Application Programmer’s Guide
Chapter 4. Introducing PKCS #11 and using PKCS #11 callable services

Chapter 5. Managing Symmetric Cryptographic Keys

Part 2. CCA Callable Services
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage Notes</td>
<td>160</td>
</tr>
<tr>
<td>Multiple Secure Key Import (CSNBSKM)</td>
<td>161</td>
</tr>
<tr>
<td>Format</td>
<td>162</td>
</tr>
<tr>
<td>Parameters</td>
<td>162</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>164</td>
</tr>
<tr>
<td>PKA Decrypt (CSNDPKD and CSNFPKD)</td>
<td>166</td>
</tr>
<tr>
<td>Format</td>
<td>166</td>
</tr>
<tr>
<td>Parameters</td>
<td>166</td>
</tr>
<tr>
<td>Restrictions</td>
<td>168</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>168</td>
</tr>
<tr>
<td>PKA Encrypt (CSNDPKE and CSNFPKE)</td>
<td>171</td>
</tr>
<tr>
<td>Format</td>
<td>171</td>
</tr>
<tr>
<td>Parameters</td>
<td>171</td>
</tr>
<tr>
<td>Restrictions</td>
<td>174</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>174</td>
</tr>
<tr>
<td>Prohibit Export (CSNBPEX)</td>
<td>175</td>
</tr>
<tr>
<td>Format</td>
<td>175</td>
</tr>
<tr>
<td>Parameters</td>
<td>175</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>176</td>
</tr>
<tr>
<td>Prohibit Export Extended (CSNBPEXX)</td>
<td>177</td>
</tr>
<tr>
<td>Format</td>
<td>177</td>
</tr>
<tr>
<td>Parameters</td>
<td>177</td>
</tr>
<tr>
<td>Restrictions</td>
<td>178</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>178</td>
</tr>
<tr>
<td>Random Number Generate (CSNBRNG, CSNERNG, CSBRNGL and CSNERNGL)</td>
<td>178</td>
</tr>
<tr>
<td>Format</td>
<td>179</td>
</tr>
<tr>
<td>Parameters</td>
<td>179</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>181</td>
</tr>
<tr>
<td>Remote Key Export (CSNDRKX)</td>
<td>182</td>
</tr>
<tr>
<td>Format</td>
<td>182</td>
</tr>
<tr>
<td>Parameters</td>
<td>182</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>188</td>
</tr>
<tr>
<td>Secure Key Import (CSNBSKI)</td>
<td>189</td>
</tr>
<tr>
<td>Format</td>
<td>189</td>
</tr>
<tr>
<td>Parameters</td>
<td>190</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>191</td>
</tr>
<tr>
<td>Symmetric Key Export (CSNDSYX and CSNFSYX)</td>
<td>192</td>
</tr>
<tr>
<td>Format</td>
<td>193</td>
</tr>
<tr>
<td>Parameters</td>
<td>193</td>
</tr>
<tr>
<td>Restrictions</td>
<td>195</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>195</td>
</tr>
<tr>
<td>Symmetric Key Generate (CSNDSYG)</td>
<td>196</td>
</tr>
<tr>
<td>Format</td>
<td>196</td>
</tr>
<tr>
<td>Parameters</td>
<td>196</td>
</tr>
<tr>
<td>Restrictions</td>
<td>199</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>199</td>
</tr>
<tr>
<td>Symmetric Key Import (CSNDSYI and CSNFSYI)</td>
<td>201</td>
</tr>
<tr>
<td>Format</td>
<td>201</td>
</tr>
<tr>
<td>Parameters</td>
<td>202</td>
</tr>
<tr>
<td>Restrictions</td>
<td>204</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>204</td>
</tr>
<tr>
<td>Transform CDMF Key (CSNBTCK)</td>
<td>206</td>
</tr>
<tr>
<td>Format</td>
<td>206</td>
</tr>
<tr>
<td>Parameters</td>
<td>206</td>
</tr>
<tr>
<td>Restrictions</td>
<td>208</td>
</tr>
</tbody>
</table>
Chapter 6. Protecting Data

Modes of Operation

Cipher Feedback (CFB) Mode

Cipher Block Chaining (CBC) Mode

Electronic Code Book (ECB) Mode

Triple DES Encryption

Processing Rules

Ciphertext Translate (CSNBCTT and CSNBCTT1)

Choosing Between CSNBCTT and CSNBCTT1

Format

Parameters

Restrictions

Usage Notes

Decipher (CSNBDEC and CSNBDEC1)

Choosing Between CSNBDEC and CSNBDEC1

Format

Parameters

Restrictions

Usage Notes

Related Information

Decode (CSNBDCO)

Considerations

Format

Parameters

Restrictions

Usage Notes

Encipher (CSNBENC and CSNBENC1)

Choosing between CSNBENC and CSNBENC1

Format

Parameters

Restrictions

Usage Notes

Related Information

Encode (CSNBECO)

Considerations

Format

Parameters

Restrictions

Usage Notes

Symmetric Algorithm Decipher (CSNBSAD, CSNBSAD1, CSNESAD and CSNESAD1)

Choosing Between CSNBSAD and CSNBSAD1 or CSNESAD and CSNESAD1

Format

Parameters

Usage Notes
Symmetric Algorithm Encipher (CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1) ........................................ 247
Choosing between CSNBSAE and CSNBSAE1 or CSNESAE and CSNESAE1 ........................................ 247
Format ................................................................................. 248
Parameters .......................................................................... 248
Usage Notes ........................................................................... 252
Symmetric Key Decipher (CSNBSYD, CSNBSYD1, CSNESYD and CSNESYD1) ........................................ 253
Choosing Between CSNBSYD and CSNBSYD1 ...................... 254
Format ................................................................................. 255
Parameters .......................................................................... 255
Usage Notes ........................................................................... 259
Related Information .............................................................. 260
Symmetric Key Encipher (CSNBSYE, CSNBSYE1, CSNESYE and CSNESYE1) ........................................ 260
Choosing between CSNBSYE and CSNBSYE1 ...................... 262
Format ................................................................................. 262
Parameters .......................................................................... 263
Usage Notes ........................................................................... 267
Related Information .............................................................. 268

Chapter 7. Verifying Data Integrity and Authenticating Messages ................................................................. 269
How MACs are Used ............................................................. 269
How Hashing Functions Are Used ......................................... 270
How MDCs Are Used ............................................................ 270
MAC Generate (CSNBMGN and CSNBMGN1) ......................... 271
Choosing Between CSNBMGN and CSNBMGN1 ...................... 271
Format ................................................................................. 272
Parameters .......................................................................... 272
Usage Notes ........................................................................... 275
Related Information .............................................................. 276
MAC Verify (CSNBMVR and CSNBMVR1) ................................. 276
Choosing Between CSNBMVR and CSNBMVR1 ...................... 276
Format ................................................................................. 277
Parameters .......................................................................... 277
Usage Notes ........................................................................... 280
Related Information .............................................................. 281
MDC Generate (CSNBMDG and CSNBMDG1) ......................... 281
Choosing Between CSNBMDG and CSNBMDG1 ...................... 282
Format ................................................................................. 282
Parameters .......................................................................... 282
Usage Notes ........................................................................... 285
One-Way Hash Generate (CSNBOWH, CSNEOWH and CSNBOWH1). .................................................. 286
Format ................................................................................. 286
Parameters .......................................................................... 287
Usage Notes ........................................................................... 289
Symmetric MAC Generate (CSNBSMG, CSNBSMG1, CSNESMG, and CSNESMG1) ................................. 290
Choosing Between CSNBSMG and CSNBSMG1 or CSNESMG and CSNESMG1 ................................. 290
Format ................................................................................. 291
Parameters .......................................................................... 291
Usage Notes ........................................................................... 294
Symmetric MAC Verify (CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1) .................................... 294
Choosing Between CSNBSMV and CSNBSMV1 or CSNESMV and CSNESMV1. ..................................... 295
Format .................................. 295
Parameters .............................. 296
Usage Notes ............................ 298

Chapter 8. Financial Services ................................................. 301
How Personal Identification Numbers (PINs) are Used ................ 301
How VISA Card Verification Values Are Used .......................... 301
Translating Data and PINs in Networks ................................. 302
Working with Europay–MasterCard–Visa smart cards ............... 302

PIN Callable Services ......................................................... 303
Generating a PIN ............................................................ 303
Encrypting a PIN ........................................................... 303
Generating a PIN Validation Value from an Encrypted PIN Block 303
Verifying a PIN .............................................................. 303
Translating a PIN ........................................................... 303
Algorithms for Generating and Verifying a PIN ....................... 303
Using PINs on Different Systems ........................................ 304
PIN-Encrypting Keys ......................................................... 304

The PIN Profile .............................................................. 305
PIN Block Format ......................................................... 305
Enhanced PIN Security Mode ............................................ 307
Format Control ............................................................. 308
Pad Digit ................................................................. 308
Current Key Serial Number ............................................... 309

Clear PIN Encrypt (CSNBCPE) ............................................ 309
Format ................................................................. 310
Parameters ............................................................... 310
Restrictions ............................................................... 312
Usage Notes ............................................................. 312

Clear PIN Generate (CSNBPGN) ........................................ 313
Format ................................................................. 313
Parameters ............................................................... 314
Restrictions ............................................................... 316
Usage Notes ............................................................. 316
Related Information ....................................................... 317

Clear PIN Generate Alternate (CSNBCPA) ................................ 317
Format ................................................................. 318
Parameters ............................................................... 318
Restrictions ............................................................... 321
Usage Notes ............................................................. 321

Encrypted PIN Generate (CSNBEPG) .................................. 323
Format ................................................................. 324
Parameters ............................................................... 324
Restrictions ............................................................... 327
Usage Notes ............................................................. 327

Encrypted PIN Translate (CSNBPTR) .................................... 328
Format ................................................................. 328
Parameters ............................................................... 328
Restrictions ............................................................... 332
Usage Notes ............................................................. 332

Encrypted PIN Verify (CSNPBPVR) ....................................... 335
Format ................................................................. 335
Parameters ............................................................... 335
Restrictions ............................................................... 339
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage Notes</td>
<td>339</td>
</tr>
<tr>
<td>Related Information</td>
<td>341</td>
</tr>
<tr>
<td>PIN Change/Unblock (CSNBPCU)</td>
<td>341</td>
</tr>
<tr>
<td>Format</td>
<td>342</td>
</tr>
<tr>
<td>Parameters</td>
<td>342</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>346</td>
</tr>
<tr>
<td>Secure Messaging for Keys (CSNBSKY)</td>
<td>347</td>
</tr>
<tr>
<td>Format</td>
<td>347</td>
</tr>
<tr>
<td>Parameters</td>
<td>347</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>350</td>
</tr>
<tr>
<td>Secure Messaging for PINs (CSNBSPN)</td>
<td>350</td>
</tr>
<tr>
<td>Format</td>
<td>351</td>
</tr>
<tr>
<td>Parameters</td>
<td>351</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>354</td>
</tr>
<tr>
<td>SET Block Compose (CSNDSCB)</td>
<td>355</td>
</tr>
<tr>
<td>Format</td>
<td>355</td>
</tr>
<tr>
<td>Parameters</td>
<td>355</td>
</tr>
<tr>
<td>Restrictions</td>
<td>359</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>359</td>
</tr>
<tr>
<td>SET Block Decompose (CSNDSD)</td>
<td>360</td>
</tr>
<tr>
<td>Format</td>
<td>360</td>
</tr>
<tr>
<td>Parameters</td>
<td>360</td>
</tr>
<tr>
<td>Restrictions</td>
<td>364</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>364</td>
</tr>
<tr>
<td>Transaction Validation (CSNBTTRV)</td>
<td>365</td>
</tr>
<tr>
<td>Format</td>
<td>366</td>
</tr>
<tr>
<td>Parameters</td>
<td>366</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>368</td>
</tr>
<tr>
<td>VISA CVV Service Generate (CSNBCSG)</td>
<td>369</td>
</tr>
<tr>
<td>Format</td>
<td>369</td>
</tr>
<tr>
<td>Parameters</td>
<td>369</td>
</tr>
<tr>
<td>Restrictions</td>
<td>372</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>372</td>
</tr>
<tr>
<td>VISA CVV Service Verify (CSNBCSV)</td>
<td>373</td>
</tr>
<tr>
<td>Format</td>
<td>373</td>
</tr>
<tr>
<td>Parameters</td>
<td>374</td>
</tr>
<tr>
<td>Restrictions</td>
<td>376</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>376</td>
</tr>
<tr>
<td>Chapter 9. Using Digital Signatures</td>
<td>379</td>
</tr>
<tr>
<td>Digital Signature Generate (CSNDDSG and CSNFDSG)</td>
<td>379</td>
</tr>
<tr>
<td>Format</td>
<td>380</td>
</tr>
<tr>
<td>Parameters</td>
<td>380</td>
</tr>
<tr>
<td>Restrictions</td>
<td>382</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>383</td>
</tr>
<tr>
<td>Digital Signature Verify (CSNDDSV and CSNFDSV)</td>
<td>385</td>
</tr>
<tr>
<td>Format</td>
<td>386</td>
</tr>
<tr>
<td>Parameters</td>
<td>386</td>
</tr>
<tr>
<td>Restrictions</td>
<td>388</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>388</td>
</tr>
<tr>
<td>Chapter 10. Managing PKA Cryptographic Keys</td>
<td>391</td>
</tr>
<tr>
<td>PKA Key Generate (CSNDPKG and CSNFPKG)</td>
<td>391</td>
</tr>
<tr>
<td>Format</td>
<td>391</td>
</tr>
<tr>
<td>Parameters</td>
<td>392</td>
</tr>
<tr>
<td>Restrictions</td>
<td>392</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>395</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>PKA Key Import (CSNDPKI and CSNFPKI)</td>
<td>395</td>
</tr>
<tr>
<td>Format</td>
<td>396</td>
</tr>
<tr>
<td>Parameters</td>
<td>396</td>
</tr>
<tr>
<td>Restrictions</td>
<td>398</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>398</td>
</tr>
<tr>
<td>PKA Key Token Build (CSNDPKB and CSNFPKB)</td>
<td>399</td>
</tr>
<tr>
<td>Format</td>
<td>400</td>
</tr>
<tr>
<td>Parameters</td>
<td>400</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>409</td>
</tr>
<tr>
<td>PKA Key Token Change (CSNDKTC)</td>
<td>409</td>
</tr>
<tr>
<td>Format</td>
<td>410</td>
</tr>
<tr>
<td>Parameters</td>
<td>410</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>411</td>
</tr>
<tr>
<td>PKA Key Translate (CSNDPKT and CSNFPKT)</td>
<td>412</td>
</tr>
<tr>
<td>Format</td>
<td>412</td>
</tr>
<tr>
<td>Parameters</td>
<td>412</td>
</tr>
<tr>
<td>Restrictions</td>
<td>415</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>415</td>
</tr>
<tr>
<td>PKA Public Key Extract (CSNDPKX and CSNFPKX)</td>
<td>415</td>
</tr>
<tr>
<td>Format</td>
<td>416</td>
</tr>
<tr>
<td>Parameters</td>
<td>416</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>417</td>
</tr>
<tr>
<td>PKDS Record Create (CSNDKRC and CSNFKRC)</td>
<td>418</td>
</tr>
<tr>
<td>Format</td>
<td>418</td>
</tr>
<tr>
<td>Parameters</td>
<td>418</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>419</td>
</tr>
<tr>
<td>PKDS Record Delete (CSNDKRD and CSNFKRD)</td>
<td>420</td>
</tr>
<tr>
<td>Format</td>
<td>420</td>
</tr>
<tr>
<td>Parameters</td>
<td>420</td>
</tr>
<tr>
<td>Restrictions</td>
<td>421</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>421</td>
</tr>
<tr>
<td>PKDS Record Read (CSNDKRR)</td>
<td>422</td>
</tr>
<tr>
<td>Format</td>
<td>422</td>
</tr>
<tr>
<td>Parameters</td>
<td>422</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>423</td>
</tr>
<tr>
<td>PKDS Record Write (CSNDKRW)</td>
<td>424</td>
</tr>
<tr>
<td>Format</td>
<td>424</td>
</tr>
<tr>
<td>Parameters</td>
<td>424</td>
</tr>
<tr>
<td>Restrictions</td>
<td>426</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>426</td>
</tr>
<tr>
<td>Retained Key Delete (CSNDRKD and CSNFRKD)</td>
<td>426</td>
</tr>
<tr>
<td>Format</td>
<td>427</td>
</tr>
<tr>
<td>Parameters</td>
<td>427</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>428</td>
</tr>
<tr>
<td>Retained Key List (CSNDRKL and CSNFRKL)</td>
<td>429</td>
</tr>
<tr>
<td>Format</td>
<td>429</td>
</tr>
<tr>
<td>Parameters</td>
<td>429</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>431</td>
</tr>
</tbody>
</table>

**Chapter 11. Utilities**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character/Nibble Conversion (CSNBXBC and CSNBXCB)</td>
<td>433</td>
</tr>
<tr>
<td>Format</td>
<td>433</td>
</tr>
<tr>
<td>Parameters</td>
<td>433</td>
</tr>
<tr>
<td>Usage Notes</td>
<td>434</td>
</tr>
<tr>
<td>Code Conversion (CSNBXEA and CSNBXAE)</td>
<td>435</td>
</tr>
</tbody>
</table>
PKCS #11 Derive key (CSFPDVK) ........................................... 496
  Format ......................................................................... 497
  Parameters ...................................................................... 497
  Authorization .................................................................. 500
  Usage Notes .................................................................... 500
PKCS #11 Get attribute value (CSFPGAV) .................................... 500
  Format ......................................................................... 500
  Parameters ...................................................................... 500
  Authorization .................................................................. 502
  Usage Notes .................................................................... 503
PKCS #11 Generate key pair (CSFPGKP) ..................................... 503
  Format ......................................................................... 503
  Parameters ...................................................................... 503
  Authorization .................................................................. 505
  Usage Notes .................................................................... 505
PKCS #11 Generate secret key (CSFPGSK) ................................... 505
  Format ......................................................................... 505
  Parameters ...................................................................... 505
  Authorization .................................................................. 507
  Usage Notes .................................................................... 507
PKCS #11 Generate HMAC (CSFPHMG) ....................................... 507
  Format ......................................................................... 508
  Parameters ...................................................................... 508
  Authorization .................................................................. 510
  Usage Notes .................................................................... 511
PKCS #11 Verify HMAC (CSFPHMV) ......................................... 511
  Format ......................................................................... 511
  Parameters ...................................................................... 511
  Authorization .................................................................. 514
  Usage Notes .................................................................... 514
PKCS #11 One-way hash generate (CSFPOWH) ............................ 514
  Format ......................................................................... 515
  Parameters ...................................................................... 515
  Authorization .................................................................. 517
  Usage Notes .................................................................... 518
PKCS #11 Private key sign (CSFPKPS) ....................................... 518
  Format ......................................................................... 518
  Parameters ...................................................................... 518
  Authorization .................................................................. 520
  Usage Notes .................................................................... 520
PKCS #11 Public key verify (CSFPKPV) ...................................... 520
  Format ......................................................................... 521
  Parameters ...................................................................... 521
  Authorization .................................................................. 523
  Usage Notes .................................................................... 523
PKCS #11 Pseudo-random function (CSFPRF) .............................. 523
  Format ......................................................................... 523
  Parameters ...................................................................... 523
  Authorization .................................................................. 525
  Usage Notes .................................................................... 525
PKCS #11 Set attribute value (CSFPSAV) .................................... 526
  Format ......................................................................... 526
  Parameters ...................................................................... 526
  Authorization .................................................................. 527
  Usage Notes .................................................................... 528
PKCS #11 Secret key decrypt (CSFPSKD) ..................................... 528
Formatting Hashes and Keys in Public-Key Cryptography .......... 695
ANSI X9.31 Hash Format .............................................. 696
PKCS #1 Formats ......................................................... 696
Visa and EMV-related smart card formats and processes .......... 697
   Deriving the smart-card-specific authentication code ........... 697
   Constructing the PIN-block for transporting an EMV smart-card PIN . 697
   Deriving the CCA TDES-XOR session key .......................... 698
   Deriving the EMV TDESEMVn tree-based session key ............... 698
   PIN-block self-encryption ........................................... 699
Key Test Verification Pattern Algorithm for DES Keys ............... 699

Appendix G. EBCDIC and ASCII Default Conversion Tables ........ 701

Appendix H. Access Control Points and Callable Services .......... 703
TKE Version 4.0 and higher ............................................ 703
TKE Version 3.1 ........................................................ 705
Callable Service Access Control Points ................................ 706

Appendix I. Accessibility .................................................. 711
Using assistive technologies ............................................ 711
Keyboard navigation of the user interface ............................ 711
z/OS information ......................................................... 711

Notices ........................................................................... 713
Programming Interface Information ..................................... 714
Trademarks ................................................................. 714

Glossary ......................................................................... 717

Index .............................................................................. 727
# Figures

1. The z/OS ICSF Library .................................................. xxx
2. Overview of trusted block contents .................................. 32
3. Simplified RKX key-token structure .................................. 36
4. Trusted block creation .................................................. 36
5. Exporting keys using a trusted block ................................. 37
6. Generating keys using a trusted block ................................. 40
7. Typical flow of callable services for remote key export ........... 41
8. PKA Key Management .................................................... 76
9. Control Vector Base Bit Map (Common Bits and Key-Encrypting Keys) ................. 635
10. Control Vector Base Bit Map (Data Operation Keys) .................. 636
11. Control Vector Base Bit Map (PIN Processing Keys and Cryptographic Variable-Encrypting Keys) ................. 637
12. Control Vector Base Bit Map (Key Generating Keys) .................. 638
13. Control Vector Translate Callable Service Mask_Array Processing ........ 645
14. Control Vector Translate Callable Service ............................ 646
15. 3624 PIN Generation Algorithm ........................................ 673
16. GBP PIN Generation Algorithm ......................................... 674
17. PIN-Offset Generation Algorithm ....................................... 675
18. PIN Verification Algorithm ............................................. 677
19. GBP PIN Verification Algorithm ....................................... 678
20. PVV Generation Algorithm ............................................. 679
22. Multiple Decipherment of Single-length Keys ....................... 687
23. Multiple Encipherment of Double-length Keys ....................... 688
24. Multiple Decipherment of Double-length Keys ....................... 689
25. Multiple Encipherment of Triple-length Keys ....................... 690
26. Multiple Decipherment of Triple-length Keys ....................... 691
27. The CDMF Key Transformation Algorithm ............................ 695
## Tables

1. Standard Return Code Values From ICSF Callable Services ........................................... 6
2. Descriptions of Key Types .................................................................................. 21
3. Access Control Points Used by ATM Remote Key Loading ........................................ 29
4. Summary of Data Encryption Standard Bits ........................................................... 46
5. Combinations of the Callable Services ....................................................................... 56
6. Summary of ICSF Callable Services ........................................................................... 61
7. Summary of PKA Key Token Sections ...................................................................... 75
8. Internal and External Private RSA Key Token Section Identifiers ............................ 78
9. Summary of PKA Callable Services ........................................................................... 79
10. Summary of PKCS #11 callable services .................................................................. 81
11. Clear key import required hardware ....................................................................... 89
12. Control vector generate required hardware ............................................................ 92
13. Keywords for Control Vector Translate .................................................................... 94
14. Control vector translate required hardware ............................................................ 95
15. Cryptographic variable encipher required hardware .............................................. 97
16. Data key export required hardware ....................................................................... 100
17. Data key import required hardware ...................................................................... 102
18. Rule Array Keywords for Diversified Key Generate ................................................ 104
19. Diversified key generate required hardware .......................................................... 107
20. Key export required hardware ............................................................................... 111
21. Key Form Values for the Key Generate Callable Service ....................................... 113
22. Key Length Values for the Key Generate Callable Service ................................... 114
23. Key lengths for DES keys - CCF systems ................................................................ 115
24. Key lengths for DES keys - PCIXCC/CEX2C/CEX3C systems ............................... 115
25. Key lengths for AES keys - CEX2C/CEX3C systems .............................................. 116
26. Key Generate Valid Key Types and Key Forms for a Single Key ............................ 120
27. Key Generate Valid Key Types and Key Forms for a Key Pair ............................... 120
28. Key generate required hardware ............................................................................ 122
29. Key import required hardware ............................................................................... 127
30. Keywords for Key Part Import Control Information ................................................. 129
31. Key part import required hardware ....................................................................... 130
32. CKDS record create required hardware .................................................................. 133
33. CKDS record delete required hardware .................................................................. 135
34. CKDS record read required hardware .................................................................... 137
35. CKDS record write required hardware .................................................................... 139
36. Keywords for Key Test Control Information ........................................................... 141
37. Key test required hardware .................................................................................... 143
38. Keywords for Key Test Extended Control Information .......................................... 145
39. Key test extended required hardware ..................................................................... 147
40. Key type keywords for key token build .................................................................. 149
41. Keywords for Key Token Build Control Information .............................................. 150
42. Key types and field lengths for AES keys ............................................................... 152
43. Control Vector Generate and Key Token Build Control Vector Keyword Combinations 154
44. Key token build required hardware ....................................................................... 156
45. Key translate required hardware ............................................................................ 158
46. Keywords for Multiple Clear Key Import Rule Array Control Information ............ 160
47. Multiple clear key import required hardware ......................................................... 161
48. Keywords for Multiple Secure Key Import Rule Array Control Information .......... 163
49. Multiple secure key import required hardware ....................................................... 165
50. Keywords for PKA Decrypt .................................................................................... 167
51. PKA decrypt required hardware ............................................................................ 170
52. Keywords for PKA Encrypt ..................................................................................... 172
53. PKA encrypt required hardware ............................................................................. 174

© Copyright IBM Corp. 1997, 2009
110. Process Rules for the Clear PIN Encryption Callable Service ........................................... 311
111. Clear PIN encrypt required hardware .............................................................................. 313
112. Process Rules for the Clear PIN Generate Callable Service ............................................ 315
113. Array Elements for the Clear PIN Generate Callable Service ......................................... 316
114. Array Elements Required by the Process Rule ................................................................. 316
115. Clear PIN generate required hardware ............................................................................. 317
116. Rule Array Elements for the Clear PIN Generate Alternate Service ............................... 320
117. Rule Array Keywords (First Element) for the Clear PIN Generate Alternate Service ....... 320
118. Data Array Elements for the Clear PIN Generate Alternate Service (IBM-PINO) .............. 321
119. Data Array Elements for the Clear PIN Generate Alternate Service (VISA-PVV) .............. 321
120. PIN Block Variant Constants (PBVCs) ................................................................................. 322
121. Clear PIN generate alternate required hardware ................................................................. 323
122. Process Rules for the Encrypted PIN Generate Callable Service .................................... 325
123. Array Elements for the Encrypted PIN Generate Callable Service .................................. 326
124. Array Elements Required by the Process Rule ................................................................. 326
125. Encrypted PIN generate required hardware. .................................................................... 327
126. Keywords for Encrypted PIN Translate ............................................................................. 330
127. Additional Names for PIN Formats ................................................................................... 333
128. PIN Block Variant Constants (PBVCs) ................................................................................. 333
129. Encrypted PIN translate required hardware ...................................................................... 334
130. Keywords for Encrypted PIN Verify .................................................................................. 337
131. Array Elements for the Encrypted PIN Verify Callable Service ...................................... 338
132. Array Elements Required by the Process Rule ................................................................. 339
133. PIN Block Variant Constants (PBVCs) ................................................................................. 340
134. Encrypted PIN verify required hardware ......................................................................... 340
135. Rule Array Keywords for PIN Change/Unblock ................................................................. 343
136. PIN Change/Unblock hardware ......................................................................................... 346
137. Rule Array Keywords for Secure Messaging for Keys ....................................................... 348
138. Secure messaging for keys required hardware ................................................................. 350
139. Rule Array Keywords for Secure Messaging for PINs ....................................................... 352
140. Secure messaging for PINs required hardware ................................................................. 354
141. Keywords for SET Block Compose Control Information ................................................. 356
142. SET block compose required hardware .......................................................................... 359
143. Keywords for SET Block Compose Control Information .................................................. 361
144. SET block decompose required hardware ....................................................................... 365
145. Rule Array Keywords for Transaction Validation ............................................................. 367
146. Output description for validation values ............................................................................ 368
147. Transaction validation required hardware ......................................................................... 368
148. CVV Generate Rule Array Keywords ............................................................................... 370
149. VISA CVV service generate required hardware ............................................................... 372
150. CVV Verify Rule Array Keywords .................................................................................... 374
151. VISA CVV service verify required hardware ..................................................................... 377
152. Keywords for Digital Signature Generate Control Information - Valid only for RSA key types. 381
153. Digital signature generate required hardware ................................................................... 384
154. Keywords for Digital Signature Verify Control Information .......................................... 387
155. Digital signature verify required hardware ......................................................................... 389
156. Keywords for PKA Key Generate Rule Array ................................................................. 393
157. PKA key generate required hardware ............................................................................... 395
158. PKA key import required hardware .................................................................................. 399
159. Keywords for PKA Key Token Build Control Information .............................................. 401
160. Key Value Structure Length Maximum Values for Key Types ........................................ 402
161. Key Value Structure Elements for PKA Key Token Build .............................................. 403
162. PKA key token build required hardware .......................................................................... 409
163. Rule Array Keywords for PKA Key Token Change (Required) ....................................... 411
164. PKA key token change required hardware ...................................................................... 411
165. Keywords for PKA Key Generate Rule Array ................................................................. 413
166. PKA key translate required hardware ........................................... 415
167. PKA public key extract build required hardware .......................... 417
168. PKDS record create required hardware ...................................... 420
169. Keywords for PKDS Record Delete ........................................... 421
170. PKDS record delete required hardware .................................... 422
171. PKDS record read required hardware ..................................... 424
172. Keywords for PKDS Record Write ........................................... 425
173. PKDS record write required hardware .................................... 426
174. Retained key delete required hardware .................................. 428
175. Retained key list required hardware ...................................... 431
176. Character/Nibble conversion required hardware ......................... 435
177. Code conversion required hardware ...................................... 437
178. Keywords for ICSF Query Algorithm ....................................... 439
179. Output for ICSF Query Algorithm .......................................... 439
180. ICSF Query Algorithm required hardware ............................... 441
181. Keywords for ICSF Query Service ......................................... 443
182. Output for option ICSFSTAT ............................................... 444
183. Output for option ICSFST2 .................................................. 446
184. Output for option STATAES ............................................... 451
185. Output for option STATCCA ............................................... 452
186. Output for option STATCCAE ............................................... 452
187. Output for option STATCARD ............................................... 453
188. Output for option STATDIAG ............................................... 455
189. Output for option STATEID ................................................ 456
190. Output for option STATEXPT .............................................. 456
191. ICSF Query Service required hardware .................................. 458
192. X9.9 data editing required hardware .................................... 461
193. Keywords for PCI Interface Callable Service ............................. 464
194. PCI Interface required hardware ......................................... 467
195. PKSC Interface required hardware ....................................... 469
196. ANSI X9.17 EDC generate required hardware ............................ 473
197. Keywords for ANSI X9.17 Key Export Rule Array ....................... 475
198. ANSI X9.17 key export required hardware ................................ 478
199. Keywords for ANSI X9.17 Key Import Rule Array ....................... 480
200. ANSI X9.17 key import required hardware ................................ 483
201. Keywords for ANSI X9.17 Key Translate Rule Array ................... 485
202. ANSI X9.17 key translate required hardware .......................... 488
203. ANSI X9.17 transport key partial notarize required hardware .......... 490
204. Keywords for derive multiple keys ....................................... 495
205. parms_list parameter format for SSL-KM and TLS-KM mechanisms. 495
206. Keywords for derive key ..................................................... 498
207. parms_list parameter format for PKCS-DH mechanism ............... 499
208. parms_list parameter format for SSL-MS, SSL-MSDH, TLS-MS, and TLS-MSDH mechanisms .............. 499
209. parms_list parameter format for EC-DH mechanism ................. 499
210. Get attribute value processing for objects possessing sensitive attributes. 502
211. Keywords for generate secret key ....................................... 506
212. parms_list parameter format for SSL and TLS mechanism .......... 507
213. Keywords for generate HMAC .............................................. 509
214. chain_data parameter format .............................................. 510
215. Keywords for verify HMAC ............................................... 512
216. chain_data parameter format .............................................. 513
217. Keywords for one-way hash generate .................................... 516
218. chain_data parameter format .............................................. 513
219. Keywords for private key sign ............................................ 517
220. Keywords for public key verify ............................................ 519
221. Keywords for derive multiple keys ...................................... 522
<table>
<thead>
<tr>
<th>Section Number</th>
<th>Section Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>222</td>
<td>parms_list parameter format for TLS-PRF mechanism</td>
<td>525</td>
</tr>
<tr>
<td>223</td>
<td>Authorization requirements for the set attribute value callable service</td>
<td>527</td>
</tr>
<tr>
<td>224</td>
<td>Keywords for secret key decrypt</td>
<td>529</td>
</tr>
<tr>
<td>225</td>
<td>initialization_vector parameter format for GCM mechanism</td>
<td>530</td>
</tr>
<tr>
<td>226</td>
<td>chain_data parameter format</td>
<td>531</td>
</tr>
<tr>
<td>227</td>
<td>Keywords for secret key encrypt</td>
<td>534</td>
</tr>
<tr>
<td>228</td>
<td>initialization_vector parameter format for GCM mechanism</td>
<td>535</td>
</tr>
<tr>
<td>229</td>
<td>chain_data parameter format</td>
<td>536</td>
</tr>
<tr>
<td>230</td>
<td>Authorization requirements for the token record create callable service</td>
<td>540</td>
</tr>
<tr>
<td>231</td>
<td>Authorization requirements for the token record delete callable service</td>
<td>542</td>
</tr>
<tr>
<td>232</td>
<td>Keywords for unwrap key</td>
<td>549</td>
</tr>
<tr>
<td>233</td>
<td>Keywords for wrap key</td>
<td>552</td>
</tr>
<tr>
<td>234</td>
<td>Return Codes</td>
<td>557</td>
</tr>
<tr>
<td>235</td>
<td>Reason Codes for Return Code 0 (0)</td>
<td>558</td>
</tr>
<tr>
<td>236</td>
<td>Reason Codes for Return Code 4 (4)</td>
<td>559</td>
</tr>
<tr>
<td>237</td>
<td>Reason Codes for Return Code 8 (8)</td>
<td>562</td>
</tr>
<tr>
<td>238</td>
<td>Reason Codes for Return Code C (12)</td>
<td>590</td>
</tr>
<tr>
<td>239</td>
<td>Reason Codes for Return Code 10 (16)</td>
<td>595</td>
</tr>
<tr>
<td>240</td>
<td>Internal Key Token Format</td>
<td>597</td>
</tr>
<tr>
<td>241</td>
<td>Internal Key Token Format</td>
<td>598</td>
</tr>
<tr>
<td>242</td>
<td>Format of External Key Tokens</td>
<td>600</td>
</tr>
<tr>
<td>243</td>
<td>External RKX DES key-token format, version X'10'</td>
<td>601</td>
</tr>
<tr>
<td>244</td>
<td>Format of Null Key Tokens</td>
<td>602</td>
</tr>
<tr>
<td>245</td>
<td>Format of PKA Null Key Tokens</td>
<td>602</td>
</tr>
<tr>
<td>246</td>
<td>RSA Public Key Token</td>
<td>602</td>
</tr>
<tr>
<td>247</td>
<td>RSA Private External Key Token Basic Record Format</td>
<td>603</td>
</tr>
<tr>
<td>248</td>
<td>RSA Private Key Token, 1024-bit Modulus-Exponent External Format</td>
<td>604</td>
</tr>
<tr>
<td>249</td>
<td>RSA Private Key Token, 4096-bit Modulus-Exponent External Format</td>
<td>605</td>
</tr>
<tr>
<td>250</td>
<td>RSA Private Key Token, 4096-bit Chinese Remainder Theorem External Format</td>
<td>606</td>
</tr>
<tr>
<td>251</td>
<td>RSA Private Internal Key Token Basic Record Format</td>
<td>608</td>
</tr>
<tr>
<td>252</td>
<td>RSA Private Internal Key Token, 1024-bit ME Form for Cryptographic Coprocessor Feature</td>
<td>609</td>
</tr>
<tr>
<td>253</td>
<td>RSA Private Internal Key Token, 1024-bit ME Form for PCICC, PCIXCC, CEX2C, or CEX3C</td>
<td>610</td>
</tr>
<tr>
<td>254</td>
<td>RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format</td>
<td>611</td>
</tr>
<tr>
<td>255</td>
<td>DSS Public Key Token</td>
<td>613</td>
</tr>
<tr>
<td>256</td>
<td>DSS Private External Key Token</td>
<td>613</td>
</tr>
<tr>
<td>257</td>
<td>DSS Private Internal Key Token</td>
<td>615</td>
</tr>
<tr>
<td>258</td>
<td>Trusted block header</td>
<td>619</td>
</tr>
<tr>
<td>259</td>
<td>Trusted block trusted RSA public-key section (X'11')</td>
<td>620</td>
</tr>
<tr>
<td>260</td>
<td>Trusted block rule section (X'12')</td>
<td>621</td>
</tr>
<tr>
<td>261</td>
<td>Summary of trusted block rule subsection</td>
<td>622</td>
</tr>
<tr>
<td>262</td>
<td>Transport key variant subsection (X'0001') of trusted block rule section (X'12)</td>
<td>623</td>
</tr>
<tr>
<td>263</td>
<td>Transport key rule reference subsection (X'0002') of trusted block rule section (X'12)</td>
<td>624</td>
</tr>
<tr>
<td>264</td>
<td>Common export key parameters subsection (X'0003') of trusted block rule section (X'12)</td>
<td>624</td>
</tr>
<tr>
<td>265</td>
<td>Source key rule reference subsection (X'0004') of trusted block rule section (X'12)</td>
<td>626</td>
</tr>
<tr>
<td>266</td>
<td>Export key CCA token parameters subsection (X'0005') of trusted block rule section (X'12)</td>
<td>626</td>
</tr>
<tr>
<td>267</td>
<td>Trusted block key label (name) section X'13'</td>
<td>628</td>
</tr>
<tr>
<td>268</td>
<td>Trusted block information section X'14'</td>
<td>629</td>
</tr>
<tr>
<td>269</td>
<td>Summary of trusted block information subsection</td>
<td>629</td>
</tr>
<tr>
<td>270</td>
<td>Protection information subsection (X'0001') of trusted block information section (X'14)</td>
<td>630</td>
</tr>
<tr>
<td>271</td>
<td>Activation and expiration dates subsection (X'0002') of trusted block information section (X'14)</td>
<td>631</td>
</tr>
<tr>
<td>272</td>
<td>Trusted block application-defined data section X'15'</td>
<td>632</td>
</tr>
<tr>
<td>273</td>
<td>Default Control Vector Values</td>
<td>633</td>
</tr>
<tr>
<td>274</td>
<td>PKA96 Clear DES Key Record</td>
<td>691</td>
</tr>
<tr>
<td>275</td>
<td>EBCDIC to ASCII Default Conversion Table</td>
<td>701</td>
</tr>
<tr>
<td>276</td>
<td>ASCII to EBCDIC Default Conversion Table</td>
<td>702</td>
</tr>
<tr>
<td>277</td>
<td>Callable service access control points</td>
<td>706</td>
</tr>
</tbody>
</table>
About this information

This information supports z/OS (5694-A01). It describes how to use the callable services provided by the Integrated Cryptographic Service Facility (ICSF). The z/OS Cryptographic Services includes these components:

- z/OS Integrated Cryptographic Service Facility (ICSF)
- z/OS Open Cryptographic Services Facility (OCSF)
- z/OS System Secure Socket Level Programming (SSL)
- z/OS Public Key Infrastructure Services (PKI)

ICSF is a software element of z/OS that works with the hardware cryptographic feature and the Security Server RACF to provide secure, high-speed cryptographic services. ICSF provides the application programming interfaces by which applications request the cryptographic services.

Who should use this information

This information is intended for application programmers who:

- Are responsible for writing application programs that use the security application programming interface (API) to access cryptographic functions.
- Want to use ICSF callable services in high-level languages such as C, COBOL, FORTRAN, and PL/I, as well as in assembler.

How to use this information

ICSF includes Advanced Encryption Standard (AES), Data Encryption Standard (DES) and public key cryptography. These are very different cryptographic systems.

These topics focus on IBM CCA programming and include:

- **Chapter 1, “Introducing Programming for the IBM CCA”** describes the programming considerations for using the ICSF callable services. It also explains the syntax and parameter definitions used in callable services.
- **Chapter 2, “Introducing Symmetric Key Cryptography and Using Symmetric Key Callable Services”** gives an overview of AES and DES cryptography and provides general guidance information on how the callable services use different key types and key forms. It also discusses how to write your own callable services called installation-defined callable services and provides suggestions on what to do if there is a problem.
- **Chapter 3, “Introducing PKA Cryptography and Using PKA Callable Services”** introduces Public Key Algorithm (PKA) support and describes programming considerations for using the ICSF PKA callable services, such as the PKA key token structure and key management.
- **Chapter 4, “Introducing PKCS #11 and using PKCS #11 callable services”** gives an overview of PKCS #11 support and management services.

These topics focus on CCA callable services and include:

- **Chapter 5, “Managing Symmetric Cryptographic Keys”** describes the callable services for generating and maintaining cryptographic keys and the random number generate callable service. It also presents utilities to build AES and DES tokens and generate and translate control vectors and describes the PKA callable services that support AES and DES key distribution.
Chapter 6, “Protecting Data” describes the callable services for deciphering ciphertext from one key and enciphering it under another key. It also describes enciphering and deciphering data with encrypted keys and encoding and decoding data with clear keys.

Chapter 7, “Verifying Data Integrity and Authenticating Messages” describes the callable services for generating and verifying message authentication codes (MACs), generating modification detection codes (MDCs) and generating hashes (SHA-1, MD5, RIPEMD-160).

Chapter 8, “Financial Services” describes the callable services for generating, verifying, and translating personal identification numbers (PINs). It also describes the callable services that support the Secure Electronic Transaction (SET) protocol and those that and generate and verify VISA card verification values and American Express card security codes.

Chapter 9, “Using Digital Signatures” describes the PKA callable services that support using digital signatures to authenticate messages.

Chapter 10, “Managing PKA Cryptographic Keys” describes the PKA callable services that generate and manage PKA keys.

Chapter 11, “Utilities” describes callable services that convert data between EBCDIC and ASCII format, convert between binary strings and character strings, and query ICSF services and algorithms.

Chapter 12, “Trusted Key Entry Workstation Interfaces” describes the PCI interface (PCI) and the Public Key Secure Cable (PKSC) interface that supports Trusted Key Entry (TKE), an optional feature available with ICSF.

Chapter 13, “Managing Keys According to the ANSI X9.17 Standard” describes the callable services that support the ANSI X9.17 key management standard \(^1\), which defines a process for protecting and exchanging DES keys.

Chapter 14, “Using PKCS #11 Tokens and Objects” describes the callable services for managing the PKCS #11 tokens and objects in the TKDS.

The appendixes include this information:

Appendix A, “ICSF and TSS Return and Reason Codes” explains the return and reason codes returned by the callable services.

Appendix B, “Key Token Formats” describes the formats for AES key tokens, DES internal, external, and null key tokens and for PKA public, private external, and private internal key tokens containing either Rivest-Shamir-Adleman (RSA) or Digital Signature Standard (DSS) information. This appendix also describes the PKA null key token.

Appendix C, “Control Vectors and Changing Control Vectors with the CVT Callable Service,” on page 633 contains a table of the default control vector values that are associated with each key type and describes the control information for testing control vectors, mask array preparation, selecting the key-half processing mode, and an example of Control Vector Translate.

Appendix D, “Coding Examples” provides examples for COBOL, assembler, and PL/I.

Appendix E, “Using ICSF with BSAFE” explains how to access ICSF services from applications written using RSA’s BSAFE cryptographic toolkit.

Appendix F, “Cryptographic Algorithms and Processes,” on page 669 describes the PIN formats and algorithms, cipher processing and segmenting rules, multiple encipherment and decipherment and their equations, the PKA92 encryption

---

\(^1\) ANSI X9.17-1985: Financial Institution Key Management (Wholesale)
process, partial notarization of an ANSI key-encrypting key (AKEK), and the algorithm for transforming a Commercial Data Masking Facility (CDMF) key.

- Appendix G, “EBCDIC and ASCII Default Conversion Tables” presents EBCDIC to ASCII and ASCII to EBCDIC conversion tables.
- Appendix H, “Access Control Points and Callable Services” lists which access control points correspond to which callable services.
- Appendix I, “Accessibility,” on page 711 contains information on accessibility features in z/OS.
- Notices contains notices, programming interface information, and trademarks.

---

**Where to find more information**

For information about the referenced ICSF publications, see Figure 1 on page xxx

Other publications referenced in this publication are:

Figure 1. The z/OS ICSF Library
Related Publications

- **z/OS Cryptographic Services ICSF TKE PCIX Workstation User's Guide**, SA23-2211
- **z/OS MVS Programming: Callable Services for High-Level Languages**, SA22-7613
- **z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU**, SA22-7611
- BSAFE User’s Manual
- BSAFE Library Reference Manual
- **z/OS Security Server RACF Command Language Reference**
- **z/OS Security Server RACF Security Administrator’s Guide**
- **IBM Common Cryptographic Architecture (CCA) Basic Services API, Release 2.53**
  
  This publication can be obtained in PDF format from the Library page at http://www.ibm.com/security/cryptocards.
- **IBM Distributed Key Management System, Installation and Customization Guide**, GG24-4406

Do You Have Problems, Comments, or Suggestions?

Your suggestions and ideas can contribute to the quality and the usability of this document. If you have problems using this document, or if you have suggestions for improving it, complete and mail the Reader’s Comment Form found at the back of the document.
Summary of changes

Summary of changes
for SA22-7522-13
z/OS Version 1 Release 11

This document contains information previously presented in z/OS ICSF Application Programmer’s Guide, SA22-7522-12, which supports z/OS Version 1 Release 10.

This document is for ICSF FMID HCR7770. This release of ICSF runs on z/OS V1R9 and z/OS V1R10 and only on zSeries hardware.

New information

• Added new callable service PKA Key Translate (CSNDPKT and CSNFPKT).
  Using this callable service, applications can translate a source CCA RSA key token into a target external smart card key token.
• Added new callable services for managing PKCS #11 tokens and objects. These additional services are:
  – PKCS #11 Derive key (CSFPDVK)
  – PKCS #11 Derive multiple keys (CSFPDMK)
  – PKCS #11 Generate HMAC (CSFPHMG)
  – PKCS #11 Generate key pair (CSFPGKP)
  – PKCS #11 Generate secret key (CSFPGSK)
  – PKCS #11 One-way hash generate (CSFPOWH)
  – PKCS #11 Private key sign (CSFPKPS)
  – PKCS #11 Pseudo-random function (CSFPPRF)
  – PKCS #11 Public key verify (CSFPKV)
  – PKCS #11 Secret key decrypt (CSFSDK)
  – PKCS #11 Secret key encrypt (CSFSEK)
  – PKCS #11 Unwrap key (CSFPUWK)
  – PKCS #11 Verify HMAC (CSFPHMV)
  – PKCS #11 Wrap key (CSFPWPK)
• Added information for the Crypto Express3 Coprocessor.

Changed information

• The Symmetric Key Export and Symmetric Key Import callable services now support invocation in AMODE(64).
  The Symmetric Key Encipher and Symmetric Key Decipher callable services now support an encrypted key in the CKDS. This enables applications to leverage the performance capabilities of CPACF when enciphering/deciphering data using encrypted symmetric keys.
• The ICSF Query Algorithm (CSFIQA and CSFIQA6) utility and ICSF Query Facility (CSFIQF and CSFIQF6) updated to return additional data.

Summary of changes
for SA22-7522-12
z/OS Version 1 Release 10

This document is for ICSF FMID HCR7751. This release of ICSF runs on z/OS V1R7, z/OS V1R8, z/OS V1R9 and z/OS V1R10 and only on zSeries hardware.

**New information**

- Added support for secure key AES
  - Added new callable service (CSNBSAD, CSNBSAD1, CSNESAD, and CSNESAD1) - Symmetric Algorithm Decipher
  - Added new callable service (CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1) - Symmetric Algorithm Encipher
- Added support for key store policy
  - new SMF 82 subtypes
- Additional SAF authorization checks when a key store policy is enabled
- Added a new utility to detect duplicate tokens in the CKDS or PKDS
- Added support for sysplex-wide consistency of PKDS
- Added support for an in-storage copy of the PKDS. This replaces the PKDS cache.
- Added support for IBM System z10 BC
- Added support for storing clear keys in the CKDS for a system without any cryptographic coprocessors
- Added support for PAN-14, PAN-15, PAN-17, PAN-18
  - CSNBCSG - VISA CVV Service Generate
  - CSNBCCSV - VISA CVV Service Verify
- Added support for IPv6
  - Added new callable service (CSNBSMG, CSNBSMG1, CSNESMG and CSNESMG1) - Symmetric MAC Generate
  - Added new callable service (CSNBSMV, CSNBSMV1, CSNESMV and CSNESMV1) - Symmetric MAC Verify
- Added new callable service (CSFIQA and CSFIQA6) - ICSF Query Algorithm
- Added SAF class XCSFKEYS for Enhanced Keylabel Access Control

**Changed information**

- Symmetric keys can now refer to DES, TDES, or AES keys
- These callable services have been changed to support secure key AES:
  - CSFIQF and CSFIQF6 - ICSF Query Function
  - CSNBKGN and CSNEKGN - Key Generate
  - CSNBKRC - Key Record Create
  - CSNBKRD - Key Record Delete
  - CSNBKRR - Key Record Read
  - CSNBKRW - Key Record Write
  - CSNBKYT - Key Test
  - CSNBKTX - Key Test Extended
  - CSNBKTB - Key Token Build
  - CSNBCKM and CSNECKM - Multiple Clear Key Import
  - CSNBSKM - Multiple Secure Key Import
  - CSNDSYX - Symmetric Key Export
  - CSNDSYG - Symmetric Key Generate
  - CSNDSYI - Symmetric Key Import
• Documentation of the Key Test and Key Test Extended callable services has been separated
• KGUP changed to support secure key AES
• Panel changes made to support the AES master key
• Updated samples shipped in SYS1.SAMPLIB

Summary of changes
for SA22-7522-11
z/OS Version 1 Release 9


This release of ICSF, HCR7750, runs on z/OS V1R7, z/OS V1R8 and z/OS V1R9 and only on zSeries hardware.

New information
• Added support for new server hardware: IBM System z10 EC
• Added new support for CPACF
• CPACF supports the AES algorithm for 192- and 256-bit keys for CSNBSYE, CSNBSYD, CSNBSYE1, CSNBSYD1

Changed information
• These callable services have been changed to support new SHA-2 algorithms:
  – CSNBOWH, CSNBOWH1, CSNEOWH and CSNEOWH1- One-Way Hash Generate
  – CSFIQF and CSFIQF6 - ICSF Query Service

Summary of changes
for SA22-7522-10
z/OS Version 1 Release 9


This release of ICSF, HCR7750, runs on z/OS V1R7, z/OS V1R8 and z/OS V1R9 and only on zSeries hardware.

New information
• Added support for 4096-bit RSA keys on
• Added support for ISO-3 PIN block format

Changed information
• These callable services have been changed to support 4096-bit RSA keys:
  – CSNDPKD and CSNFPKD - PKA Decrypt
  – CSNDPKE and CSNFPKE - PKA Encrypt
  – CSNDPKX and CSNFPKX - PKA Key Extract
  – CSNDPKG and CSNFPKG - PKA Key Generate
  – CSNDPKI and CSNFPKI - PKA Key Import
  – CSNDPKB and CSNFPKB - PKA Key Token Build
  – CSNDKTC - PKA Key Token Change
– CSNDKRC and CSNFKRC - PKDS Record Create
– CSNDKRW - PKDS Record Write
– CSNDKRD and CSNFKRD - PKDS Record Delete
– CSNDRKC - Remote Key Export
– CSNDSYX - Symmetric Key Export
– CSNDSYG - Symmetric Key Generate
– CSNDSYI - Symmetric Key Import
– CSNDTBC - Trusted Block Create
– CSNDDSG - Digital Signature Generate
– CSNDDSV - Digital Signature Verify
• These callable services have been changed to support ISO-3 PIN block format:
  – CSNBCPE - Clear PIN Encrypt
  – CSNBEPG - Encrypted PIN Generate
  – CSNBPTR - Encrypted PIN Translate
  – CSNBPVR - Encrypted PIN Verify
  – CSNBPCU - PIN Change/Unblock
  – CSNBSPN - Secure Messaging for PINs
  
• The callable service, Random Number Generate Long (CSNBRNGL and CSNERNGL), now allows a user to specify the length when generating a random number.

This document has been enabled for the following types of advanced searches in the online z/OS Library Center: examples, tasks, concepts, references, and parmlib members.

You may notice changes in the style and structure of some content in this document—for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our documents.

This document contains terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

**Summary of changes**

for SA22-7522-09

z/OS Version 1 Release 9

This document contains information previously presented in z/OS ICSF Application Programmer's Guide, SA22-7522-08, which supports z/OS Version 1 Release 8.

This release of ICSF, HCR7740, only runs on z/OS V1R9 and only on zSeries hardware.

**New information**

• These callable services have been added to support PKCS #11 token management:
  – CSFPTRC - Token Record Create
  – CSFPTRD - Token Record Delete
  – CSFPTRL - Token Record List
- CSFPSAV - Set Attribute Value
- CSFPGAV - Get Attribute Value

**Changed information**

- These callable services have been changed to support Cipher Feedback Mode (CFB) and PKCS #7 padding for encryption:
  - CSNBSYD - Symmetric Key Decipher (new CFB and PKCS-PAD keywords)
  - CSNBSYE - Symmetric Key Encipher (new CFB and PKCS-PAD keywords)
- CSNBCSV/CSNBCSG - Added that on an IBM @server zSeries 800, IBM @server zSeries 900 or higher, the user must pad out the CVV with blanks if the supplied CVV is less than 5 characters.
- Random Number Generate - Added that on a Cryptographic Coprocessor Feature only SMK-MK set is required.

This document has been enabled for the following types of advanced searches in the online z/OS Library Center: commands, examples, tasks, concepts, references, and parmlib members.

You may notice changes in the style and structure of some content in this document—for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our documents.

This document contains terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.
Part 1. IBM CCA Programming

IBM CCA Programming introduces programming for the IBM CCA, AES cryptography, DES cryptography and PKA cryptography. It explains how to use DES, AES and PKA callable services.
Chapter 1. Introducing Programming for the IBM CCA

ICSF provides access to cryptographic functions through callable services, which are also known as verbs. A callable service is a routine that receives control using a CALL statement in an application language.

Prior to invoking callable services in an application program, you must link them into the application program. See "Linking a Program with the ICSF Callable Services" on page 13.

To invoke the callable service, the application program must include a procedure call statement that has the entry point name and parameters for the callable service. The parameters that are associated with a callable service provide the only communication between the application program and ICSF.

Callable Service Syntax

This publication uses a general call format to show the name of the ICSF callable service and its parameters. An example of that format is shown here:

CALL CSNBxxx (return_code, reason_code, exit_data_length, exit_data, parameter_5, parameter_6, ... parameter_N)

where CSNBxxx is the name of the callable service. The return code, reason code, exit data length, exit data, parameter 5 through parameter N represent the parameter list. The call generates a fixed length parameter list. You must supply the parameters in the order shown in the syntax diagrams. "Parameter Definitions" on page 6 describes the parameters in more detail.

ICSF callable services can be called from application programs written in a number of high-level languages as well as assembler. The high-level languages are:

- C
- COBOL
- FORTRAN
- PL/I

The ICSF callable services comply with the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface. The services can be invoked using the generic format, CSNBxxx. Use the generic format if you want your application to work with more than one cryptographic product. The format CSFxxxx can be used in place of CSNBxxx. Otherwise, use the CSFxxxx format.

Specific formats for the languages that can invoke ICSF callable services are as follows:

C
CSNBxxx (return_code, reason_code, exit_data_length, exit_data, parameter_5, ... parameter_N)

COBOL
CALL 'CSNBxxxx' USING return_code,reason_code,exit_data_length,
exit_data,parameter_5,...parameter_N

FORTRAN
CALL CSNBxxxx (return_code,reason_code,exit_data_length,exit_data,
parameter_5,...parameter_N)

PL/I
DCL CSNBxxxx ENTRY OPTIONS(ASM);
CALL CSNBxxxx return_code,reason_code,exit_data_length,exit_data,
parameter_5,...parameter_N;

Assembler language programs must use standard linkage conventions when
invoking ICSF callable services. An example of how an assembler language
program can invoke a callable service is shown as follows:
CALL CSNBxxxx,(return_code,reason_code,exit_data_length,exit_data,
parameter_5,...parameter_N)

Coding examples using the high-level languages are shown in Appendix D, "Coding
Examples."

Callable Services with ALET Parameters

Some callable services have an alternate entry point (with ALET parameters—for
data that resides in data spaces). They are in the format of CSNBxxx1:

<table>
<thead>
<tr>
<th>Verb</th>
<th>Callable Service without ALET</th>
<th>Callable Service with ALET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ciphertext translate</td>
<td>CSNBCTT</td>
<td>CSNBCTT1</td>
</tr>
<tr>
<td>Decipher</td>
<td>CSNBDEC</td>
<td>CSNBDEC1</td>
</tr>
<tr>
<td>Encipher</td>
<td>CSNBENC</td>
<td>CSNBENC1</td>
</tr>
<tr>
<td>MAC generate</td>
<td>CSNBMGN</td>
<td>CSNBMGN1</td>
</tr>
<tr>
<td>MAC verify</td>
<td>CSNBMVIR</td>
<td>CSNBMVIR1</td>
</tr>
<tr>
<td>MDC generate</td>
<td>CSNBMDG</td>
<td>CSNBMDG1</td>
</tr>
<tr>
<td>One way hash generate</td>
<td>CSNBOWH</td>
<td>CSNBOWH1</td>
</tr>
<tr>
<td>Symmetric algorithm decipher</td>
<td>CSNBSAD</td>
<td>CSNBSAD1</td>
</tr>
<tr>
<td>Symmetric algorithm encipher</td>
<td>CSNBSAE</td>
<td>CSNBSAE1</td>
</tr>
<tr>
<td>Symmetric key decipher</td>
<td>CSNBSYD1</td>
<td>CSNBSYD1</td>
</tr>
<tr>
<td>Symmetric key encipher</td>
<td>CSNBSYE</td>
<td>CSNBSYE1</td>
</tr>
<tr>
<td>Symmetric MAC generate</td>
<td>CSNBSMG</td>
<td>CSNBSMG1</td>
</tr>
<tr>
<td>Symmetric MAC verify</td>
<td>CSNBSMV</td>
<td>CSNBSMV1</td>
</tr>
</tbody>
</table>

When choosing which service to use, consider the fact that:

- Callable services that do not have an ALET parameter require data to reside in
  the caller's primary address space. A program using these services adheres to
  the IBM Common Cryptographic Architecture: Cryptographic Application
  Programming Interface.

- Callable services that have an ALET parameter allow data to reside either in the
  caller's primary address space or in a data space. This can allow you to encipher
  more data with one call. However, a program using these services does not
  adhere to the IBM Common Cryptographic Architecture: Cryptographic
  Application Programming Interface, and may need to be modified prior to running
  with other cryptographic products that follow this programming interface.
Rules for Defining Parameters and Attributes

These rules apply to the callable services:

- Parameters are required and positional.
- Each parameter list has a fixed number of parameters.
- Each parameter is defined as an integer or a character string. Null pointers are not acceptable for any parameter.
- Keywords passed to the callable services, such as TOKEN, CUSP, and FIRST can be in lower, upper, or mixed case. The callable services fold them to uppercase prior to using them.

Each callable service defines its own list of parameters. The entire list must be supplied on every call. If you do not use a specific parameter, you must supply that parameter with hexadecimal zeros or binary zeros.

Parameters are passed to the callable service. All information that is exchanged between the application program and the callable service is through parameters passed on the call.

Each parameter definition begins with the direction that the data flows and the attributes that the parameter must possess (called "type"). This describes the direction.

<table>
<thead>
<tr>
<th>Direction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td>The application sends (supplies) the parameter to the callable service. The callable service does not change the value of the parameter.</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>The callable service returns the parameter to the application program. The callable service may have changed the value of the parameter on return.</td>
</tr>
<tr>
<td><strong>Input/Output</strong></td>
<td>The application sends (supplies) the parameter to the callable service. The callable service may have changed the value of the parameter on return.</td>
</tr>
</tbody>
</table>

This describes the attributes or type.

<table>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Integer (I)</strong></td>
<td>A 4-byte (32-bit), twos complement, binary number that has sign significance.</td>
</tr>
<tr>
<td><strong>String</strong></td>
<td>A series of bytes where the sequence of the bytes must be maintained. Each byte can take on any bit configuration. The string consists only of data bytes. No string terminators, field-length values, or type-casting parameters are included. The maximum size of a string is 'X'7FFFFFFF' or 2 gigabytes. In some of the callable services, the length of some string data has an upper bound defined by the installation. The upper bound of a string can also be defined by the service.</td>
</tr>
<tr>
<td><strong>Alphanumeric character string</strong></td>
<td>A string of bytes in which each byte represents characters from this set:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Character</th>
<th>EBCDIC Value</th>
<th>Character</th>
<th>EBCDIC Value</th>
<th>Character</th>
<th>EBCDIC Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-Z</td>
<td></td>
<td>(</td>
<td>X'40'</td>
<td>/</td>
<td>X'61'</td>
</tr>
</tbody>
</table>
Parameter Definitions

This topic describes these parameters, which are used by most of the callable services:

- **Return_code**
- **Reason_code**
- **Exit_data_length**
- **Exit_data**
- **Key_identifier**

**Note:** The **return_code** parameter, the **reason_code** parameter, the **exit_data_length** parameter, and the **exit_data** parameter are required with every callable service.

Return and Reason Codes

**Return_code** and **reason_code** parameters return integer values upon completion of the call.

**Return_code**

The return code parameter contains the general results of processing as an integer.

<table>
<thead>
<tr>
<th>Value Hex (Decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (00)</td>
<td>Successful. Normal return.</td>
</tr>
<tr>
<td>04 (04)</td>
<td>A warning. Execution was completed with a minor, unusual event encountered.</td>
</tr>
<tr>
<td>08 (08)</td>
<td>An application error occurred. The callable service was stopped due to an error in the parameters. Or, another condition was encountered that needs to be investigated.</td>
</tr>
<tr>
<td>0C (12)</td>
<td>Error. ICSF is not active or an environment error was detected.</td>
</tr>
<tr>
<td>10 (16)</td>
<td>System error. The callable service was stopped due to a processing error within the software or hardware.</td>
</tr>
</tbody>
</table>

Generally, PCF macros will receive identical error return codes if they execute on PCF or on ICSF. A single exception has been noted: if a key is installed on the ICSF CKDS with the correct label but with the wrong key type, PCF issues a return code of 8, indicating that the key type was incorrect. ICSF issues a return code of 12, indicating that the key could not be found.

**Reason_code**

The reason code parameter contains the results of processing as an integer. You can specify which set of reason codes (ICSF or TSS) are returned from callable services. The default value is ICSF. For more information about the
REASONCODES installation option, see z/OS Cryptographic Services ICSF System Programmer’s Guide. Different results are assigned to unique reason code values under a return code.

A list of reason codes is shown in Appendix A, “ICSF and TSS Return and Reason Codes.”

Exit Data Length and Exit Data
The exit_data_length and exit_data parameters are described here. The parameters are input to all callable services. Although all services require these parameters, several services ignore them.

Exit_data_length
The integer that has the string length of the data passed to the exit. The data is identified in the exit_data parameter.

Exit_data
The installation exit data string that is passed to the callable service’s preprocessing exit. The installation exit can use the data for its own processing.

ICSF provides two installation exits for each callable service. The preprocessing exit is invoked when an application program calls a callable service, but prior to when the callable service starts processing. For example, this exit is used to check or change parameters passed on the call or to stop the call. It can also be used to perform additional security checks.

The post-processing exit is invoked when the callable service has completed processing, but prior to when the callable service returns control to the application program. For example, this exit can be used to check and change return codes from the callable service or perform clean-up processing.

For more information about the exits, see z/OS Cryptographic Services ICSF System Programmer’s Guide.

Key Identifier for Key Token
A key identifier for a key token is an area that contains one of these:

- **Key label** identifies keys that are in the CKDS or PKDS. Ask your ICSF administrator for the key labels that you can use.

- **Key token** can be either an internal key token, an external key token, or a null key token. Key tokens are generated by an application (for example, using the key generate callable service), or received from another system that can produce external key tokens.

  An **internal key token** can be used only on ICSF because the master key encrypts the key value. Internal key tokens contain keys in operational form only.

  An **external key token** can be exchanged with other systems because a transport key that is shared with the other system encrypts the key value. External key tokens contain keys in either exportable or importable form.

  A **null key token** can be used to import a key from a system that cannot produce external key tokens. A null key token contains a key encrypted under an importer key-encrypting key but does not contain the other information present in an external key token.

The term **key identifier** is used to indicate that different inputs are possible for a parameter. One or more of the previously described items may be accepted by the callable service.
**Key Label:** If the first byte of the key identifier is greater than X'40', the field is considered to be holding a **key label**. The contents of a key label are interpreted as a pointer to a CKDS or PKDS key entry. The key label is an indirect reference to an internal key token.

A key label is specified on callable services with the **key_identifier** parameter as a 64-byte character string, left-justified, and padded on the right with blanks. In most cases, the callable service does not check the syntax of the key label beyond the first byte. One exception is the key record create callable service which enforces the KGUP rules for key labels unless syntax checking is bypassed by a preprocessing exit.

A key label has this form:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>00-63</td>
<td>64</td>
<td>Key label name</td>
</tr>
</tbody>
</table>

There are some general rules for creating labels for CKDS key records.

- Each label can consist of up to 64 characters. The first character must be alphabetic or a national character (#, $, @). The remaining characters can be alphanumeric, a national character (#, $, @), or a period (.)
- Labels must be unique for DATA, DATAXLAT, MAC, MACVER, DATAM, and DATAMV keys.
- For compatibility with Version 1 Release 1 function, transport and PIN keys can have duplicate labels for different key types. Keys that use the dynamic CKDS update services to create or update, however, must have unique key labels.
- Labels must be unique for any key record, including transport and PIN keys, created or updated using the dynamic CKDS update services.

**Invocation Requirements**

Applications that use ICSF callable services must meet these invocation requirements:

- Data can be located higher or lower than 16Mb but must be 31-bit addressable.
  - For these services listed which can be invoked in AMODE(64), data can be located above 2Gb if the service is invoked in AMODE(64)
- Problem or supervisor state
- Any PSW key
- Task mode or Service Request Block (SRB) mode
- No mode restrictions
- Enabled for interrupts
- No locks held

The exceptions to this list are documented with the individual callable services.

**Callable Services that support AMODE(64)**

These services support invocation in AMODE(64):

- Clear Key Import - CSNECKI
- Decipher - CSNEDEC
- Digital Signature Generate - CSNFDSG
- Digital Signature Verify - CSNFDSV
- Encipher - CSNEENC
- ICSF Query Algorithm - CSFIQA6
• ICSF Query Facility - CSFIQF6
• Key Generate - CSNEKGN
• Multiple Clear Key Import - CSNECKM
• One Way Hash - CSNEOWH
• PKA Decrypt - CSNFPKD
• PKA Encrypt - CSNFPKE
• PKA Key Generate - CSNFPKG
• PKA Key Import - CSNFPKI
• PKA Key Token Build - CSNFPKB
• PKA Public Key Extract - CSNFPKX
• PKCS #11 Derive key - CSF1DVK
• PKCS #11 Derive multiple keys - CSF1DMK
• PKCS #11 Generate HMAC - CSF1HMG
• PKCS #11 Generate key pair - CSF1GKP
• PKCS #11 Generate secret key - CSF1GSK
• PKCS #11 Get attribute value - CSF1GAV
• PKCS #11 One-way hash generate - CSF1OWH
• PKCS #11 Private key sign - CSF1PKS
• PKCS #11 Pseudo-random function - CSF1PRF
• PKCS #11 Public key verify - CSF1PKV
• PKCS #11 Secret key decrypt - CSF1SKD
• PKCS #11 Secret key encrypt - CSF1SKE
• PKCS #11 Set attribute value - CSF1SAV
• PKCS #11 token record create - CSF1TRC
• PKCS #11 token record delete - CSF1TRD
• PKCS #11 token record list - CSF1TRL
• PKCS #11 Unwrap key - CSF1UWK
• PKCS #11 Verify HMAC - CSF1HMV
• PKCS #11 Wrap key - CSF1WPK
• PKDS Record Create - CSNFKRC
• PKDS Record Delete - CSNFKRD
• Random Number Generate - CSNERNG
• Random Number Generate Long - CSNBRNGL
• Retained Key Delete - CSNFRKD
• Retained Key List - CSNFRKL
• Symmetric Algorithm Decipher - CSNESAD and CSNESAD1
• Symmetric Algorithm Encipher - CSNESAE and CSNESAE1
• Symmetric Key Export - CSNFSYX
• Symmetric Key Import - CSNFSYI
• Symmetric Key Decipher - CSNESYD
• Symmetric Key Encipher - CSNESYE
• Symmetric MAC Generate - CSNESMG and CSNESMG1
• Symmetric MAC Verify - CSNESMV and CSNESMV1
Applications which are written for AMODE(64) operation must be linked with the ICSF 64-bit service stubs, and must invoke the service with the appropriate service name. (Refer to the description of the individual callable service to determine the service name to be used.)

Security Considerations

Your installation can use the Security Server RACF or an equivalent product to control who can use ICSF callable services or key labels. Prior to using an ICSF callable service or a key label, ask your security administrator to ensure that you have the necessary authorization. For more information, see \link z/OS Security Server RACF Security Administrator's Guide. HCR7751 and later supports a key store policy using the RACF XFACILIT class. See \link z/OS Security Server RACF Security Administrator's Guide. RACF does not control all services. The usage notes topic in the callable service description will highlight those services which are not controlled.

Performance Considerations

In most cases, the z/OS operating system dispatcher provides optimum performance. However, if your application makes extensive use of ICSF functions, you should consider using one or both of these:

- **CCF Systems Only**: If your application runs in SRB mode, use the SCHEDULE macro or IEAAFFN callable service. You should consider scheduling an SRB to run on a processor with the cryptographic feature installed (using the FEATURE=CRYPTO keyword on the SCHEDULE macro). For more information on the SCHEDULE macro, refer to \link z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU. **Restriction**: The FEATURE=CRYPTO keyword should not be specified when running on an IBM @server zSeries 990.

- Use the IEAAFFN callable service (processor affinity) to avoid system overhead in selecting which processor your program (specifically, a particular TCB in the application) runs in. Note that you do not have to use the IEAAFFN service to ensure that the system runs a program on a processor with a cryptographic feature; the system ensures that automatically. However, you can avoid some of the system overhead involved in the selection process by using the IEAAFFN service, thus improving the program's performance. For more information on using the IEAAFFN callable service, refer to \link z/OS MVS Programming: Callable Services for High-Level Languages. IBM recommends that you run applications first without using these options. Consider these options when you are tuning your application for performance. Use these options only if they improve the performance of your application.

Special Secure Mode

Special secure mode is a special processing mode in which:

- The Secure Key Import and Multiple Secure Key Import callable services, which work with clear keys, can be used.

- The Clear PIN Generate callable service, which works with clear PINs, can be used.
• The Symmetric Key Generate callable service with the "IM" keyword (the DES enciphered key is enciphered by an IMPORTER key) can be used (CCF Systems Only).
• The key generator utility program (KGUP) can be used to enter clear keys into the CKDS.

To use special secure mode, several conditions must be met.
• The installation options data set must specify YES for the SSM installation option.

  For information about specifying installation options, see z/OS Cryptographic Services ICSF System Programmer's Guide.

  This is required for all systems.
• The environmental control mask (ECM) must be configured to permit special secure mode.

  The ECM is a 32-bit mask defined for each cryptographic domain during hardware installation. The second bit in this mask must have been turned on to enable special secure mode. The default is to have this bit turned on in the ECM. The bit can only be turned off/on through the optional TKE Workstation.

  This is required for systems with the Cryptographic Coprocessor Feature.
• If you are running in LPAR mode, special secure mode must be enabled.

  On the IBM @server zSeries 800 and the IBM @server zSeries 900, you enable special secure mode during activation using the Crypto page of the Customize Activation Profiles task. When activated, you can enable or disable special secure mode on the Change LPAR Crypto task. Both of these tasks can be accessed from the Hardware Management Console.

  This is required for systems with the Cryptographic Coprocessor Feature.

For the IBM @server zSeries 800 and the IBM @server zSeries 900 with TKE, TKE can disable/enable special secure mode.

---

**Using the Callable Services**

This topic discusses how ICSF callable services use the different key types and key forms. It also provides suggestions on what to do if there is a problem.

ICSF provides callable services that perform cryptographic functions. You call and pass parameters to a callable service from an application program. Besides the callable services ICSF provides, you can write your own callable services called **installation-defined callable services**. Note that only an experienced system programmer should attempt to write an installation-defined callable service.

To write an installation-defined callable service, you must first write the callable service and link-edit it into a load module. Then define the service in the installation options data set.

You must also write a service stub. To execute an installation-defined callable service, you call a service stub from your application program. In the service stub, you specify the service number that identifies the callable service.

For more information about installation-defined callable services, see z/OS Cryptographic Services ICSF System Programmer's Guide.
When the Call Succeeds

If the return code is 0, ICSF has successfully completed the call. If a reason code other than 0 is included, refer to Appendix A, "ICSF and TSS Return and Reason Codes," on page 557, for additional information. For instance, reason code 10000 indicates the key in the key identifier (or more than one key identifier, for services that use two internal key identifiers) has been reenciphered from encipherment under the old master key to encipherment under the current master key. Keys in external tokens are not affected by this processing because they contain keys enciphered under keys other than the host master key. If you manage your key identifiers on disk, then reason code 10000 should be a “trigger” to store these updated key identifiers back on disk.

Your program can now continue providing its function, but you may want to communicate the key that you used to another enterprise. This process is exporting a key.

If you want to communicate the key that you are using to a cryptographic partner, there are several methods to use:

- For DATA keys only, call the data key export callable service. You now have a DATA key type in exportable form.
- Call the key export callable service. You now have the key type in exportable form.
- When you use the key generate callable service to create your operational or importable key form, you can create an exportable form, at the same time, and you now have the key type, in exportable form, at the same time as you get the operational or importable form.

When the Call Does Not Succeed

Now you have planned your use of the ICSF callable services, made the call, but the service has completed with a return and reason codes other than zero.

If the return code is 4, there was a minor problem. For example, reason code 8004 indicates the trial MAC that was supplied does not match the message text provided.

If the return code is 8, there was a problem with one of your parameters. Check the meaning of the reason code value, correct the parameter, and call the service again. You may go through this process several times prior to succeeding.

If the return code is 12, ICSF is not active, or has no access to cryptographic units, or has an environmental problem. Check with your ICSF administrator.

If the return code is 16, the service has a serious problem that needs the help of your system programmer.

There are several reason codes that can occur when you have already fully debugged and tested your program. For example:

- Reason code 10004 indicates that you provided a key identifier that holds a key enciphered under a host master key. The host master key is not installed in the cryptographic unit. If this happens, you have to go back and import your importable key form again and call the service again. You need to build this flow into your program logic.
Reason code 10012 indicates a key corresponding to the label that you specified is not in the CKDS or PKDS. Check with your ICSF administrator to see if the label is correct.

Reason code 3063 indicates RACF failed your request to use a token.

Reason code 16000 indicates RACF failed your request to use a service.

Reason code 16004 indicates RACF failed your request to use the key label.

Examine your CSFKEYS profiles and key store policies for possible errors.

Return and reason codes are described in Appendix A, "ICSF and TSS Return and Reason Codes," on page 557.

**Linking a Program with the ICSF Callable Services**

To link the ICSF callable services into an application program, you can use these sample JCL statements. In the SYSLIB concatenation, include the CSF.SCSFMOD0 module in the link edit step. This provides the application program access to all ICSF callable services (those that can be invoked in AMODE(24)/AMODE(31) as well as those that can be invoked in AMODE(64)).

```plaintext
//LKDENC JOB
//*-----------------------------------------------------------------*
//** The JCL links the ICSF encipher callable service, CSNBENC, **
//** into an application called ENCIPHER. **
//*-----------------------------------------------------------------*
//LINK EXEC PGM=IEWL,
// PARM='XREF,LIST,LET'
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(10,10))
//SYSPRINT DD SYSOUT=*       
//SYSLIB DD DSN=CSF.SCSFMOD0,DISP=SHR * SERVICES ARE IN HERE
//SYSLMOD DD DSN=MYAPPL.LOAD,DISP=SHR * MY APPLICATION LIBRARY
//SYSLIN DD DSN=MYAPPL.ENCIPHER.OBJ,DISP=SHR * MY ENCIPHER PROGRAM
// ENTRY ENCIPHER
// NAME ENCIPHER(R)
/*
```
The Integrated Cryptographic Service Facility protects data from unauthorized disclosure or modification. ICSF protects data stored within a system, stored in a file off a system on magnetic tape, and sent between systems. ICSF also authenticates the identity of customers in the financial industry and authenticates messages from originator to receiver. It uses cryptography to accomplish these functions.

ICSF provides access to cryptographic functions through callable services. A callable service is a routine that receives control using a CALL statement in an application language. Each callable service performs one or more cryptographic functions, including:

- Generating and managing cryptographic keys
- Enciphering and deciphering data with encrypted keys using the U.S. National Institute of Standards and Technology (NIST) Data Encryption Standard (DES) or the Commercial Data Masking Facility (CDMF)
- Enciphering and deciphering data with clear keys using either the NIST Data Encryption Standard (DES), or Advanced Encryption Standard (AES)
- Transforming a CDMF DATA key to a transformed shortened DES key
- Reenciphering text from encryption under one key to encryption under another key
- Encoding and decoding data with clear keys
- Generating random numbers
- Ensuring data integrity and verifying message authentication
- Generating, verifying, and translating personal identification numbers (PINs) that identify a customer on a financial system

This topic provides an overview of the symmetric key cryptographic functions provided in ICSF, explains the functions of the cryptographic keys, and introduces the topic of building key tokens. Many services have hardware requirements. See each service for details.

Functions of the Symmetric Cryptographic Keys

ICSF provides functions to create, import, and export DES and AES keys. This topic gives an overview of these cryptographic keys. Detailed information about how ICSF organizes and protects keys is in the [z/OS Cryptographic Services ICSF Administrator’s Guide](https://www.ibm.com/support/docview.wss?uid=swg27048400).

Key Separation

The cryptographic feature controls the use of keys by separating them into unique types, allowing you to use a specific type of key only for its intended purpose. For example, a key used to protect data cannot be used to protect a key.

An ICSF system has only one DES master key and one AES master key. However, to provide for key separation, the cryptographic feature automatically encrypts each type of key under a unique variation of the master key. Each variation of the master key encrypts a different type of key. Although you enter only one master key, you have a unique master key to encrypt all other keys of a certain type.
**Note:** In PCF, key separation applies only to keys enciphered under the master key (keys in operational form). In ICSF, key separation also applies to keys enciphered under transport keys (keys in importable or exportable form). This allows the creator of a key to transmit the key to another system and to enforce its use at the other system.

**Master Key Variant**

Whenever the master key is used to encipher a key, the cryptographic coprocessor produces a variation of the master key according to the type of key the master key will encipher. These variations are called *master key variants*. The cryptographic coprocessor creates a master key variant by exclusive ORing a fixed pattern, called a *control vector*, onto the master key. A unique control vector is associated with each type of key. For example, all the different types of data-encrypting, PIN, MAC, and transport keys each use a unique control vector which is exclusive ORed with the master key in order to produce the variant. The different key types are described in "Types of Keys" on page 19.

Each master key variant protects a different type of key. It is similar to having a unique master key protect all the keys of a certain type.

The master key, in the form of master key variants, protects keys operating on the system. A key can be used in a cryptographic function only when it is enciphered under a master key. When systems want to share keys, transport keys are used to protect keys sent outside of systems. When a key is enciphered under a transport key, the key cannot be used in a cryptographic function. It must first be brought on to a system and enciphered under the system’s master key, or exported to another system where it will then be enciphered under that system’s master key.

**Transport Key Variant**

Like the master key, ICSF creates variations of a transport key to encrypt a key according to its type. This allows for key separation when a key is transported off the system. A *transport key variant*, also called *key-encrypting key variant*, is created the same way a master key variant is created. The transport key’s clear value is exclusive ORed with a control vector associated with the key type of the key it protects.

**Note:** To exchange keys with systems that do not recognize transport key variants, ICSF allows you to encrypt selected keys under a transport key itself, not under the transport key variant. For more information, see the 'Transport keys (or key-encrypting keys)' list item in "Types of Keys" on page 19.

**Key Forms**

A key that is protected under the master key is in *operational form*, which means ICSF can use it in cryptographic functions on the system.

When you store a key with a file or send it to another system, the key is enciphered under a transport key rather than the master key because, for security reasons, the key should no longer be active on the system. When ICSF enciphers a key under a transport key, the key is not in operational form and cannot be used to perform cryptographic functions.
When a key is enciphered under a transport key, the sending system considers the key in exportable form. The receiving system considers the key in importable form. When a key is reenciphered from under a transport key to under a system’s master key, it is in operational form again.

Enciphered keys appear in three forms. The form you need depends on how and when you use a key.

- **Operational** key form is used at the local system. Many callable services can use an operational key form.
  - The key token build, key generate, key import, data key import, clear key import, multiple clear key import, secure key import, and multiple secure key import callable services can create an operational key form.

- **Exportable** key form is transported to another cryptographic system. It can only be passed to another system. The ICSF callable services cannot use it for cryptographic functions. The key generate, data key export, and key export callable services produce the exportable key form.

- **Importable** key form can be transformed into operational form on the local system. The key import callable service (CSNBKIM) and the Data key import callable service (CSNBDKM) can use an importable key form. Only the key generate callable service (CSNBKGN) can create an importable key form. The secure key import (CSNBSKI) and multiple secure key import (CSNBSKM) callable services can convert a clear key into an importable key form.

For more information about the key types, see either “Functions of the Symmetric Cryptographic Keys” on page 15 or the z/OS Cryptographic Services ICSF Administrator’s Guide. See “Key Forms and Types Used in the Key Generate Callable Service” on page 56 for more information about key form.

**DES Key Flow**

The conversion from one key to another key is considered to be a one-way flow. An operational key form cannot be turned back into an importable key form. An exportable key form cannot be turned back into an operational or importable key form. The flow of ICSF key forms can only be in one direction:

**IMPORTABLE** —to— **OPERATIONAL** —to— **EXPORTABLE**

**Key Token**

A key token is a 64-byte field composed of a key value and control information. The control information is assigned to the key when ICSF creates the key. The key token can be either an internal key token, an external key token, or a null key token. Through the use of key tokens, ICSF can:

- Support continuous operation across a master key change
- Control use of keys in cryptographic services

If the first byte of the key identifier is X'01', the key identifier is interpreted as an **internal key token**. An internal key token is a token that can be used only on the ICSF system that created it (or another ICSF system with the same host master key). It contains a key that is encrypted under the master key.

An application obtains an internal key token by using one of the callable services such as those listed here. The callable services are described in detail in Chapter 5, “Managing Symmetric Cryptographic Keys.”

- Key generate
- Key import
- Secure key import
The master key may be dynamically changed between the time that you invoke a service, such as the key import callable service to obtain a key token, and the time that you pass the key token to the encipher callable service. When a change to the master key occurs, ICSF reenciphers the caller’s key from under the old master key to under the new master key. A Return Code of 0 with a reason code of 10000 notifies you that ICSF reenciphered the key. For information on reenciphering the CKDS or the PKDS, see z/OS Cryptographic Services ICSF Administrator’s Guide.

Attention: If an internal key token held in user storage is not used while the master key is changed twice, the internal key token is no longer usable. (See “Other Considerations” on page 21 for additional information.)

For debugging information, see Appendix B, “Key Token Formats” for the format of an internal key token.

If the first byte of the key identifier is X’02’, the key identifier is interpreted as an external key token. By using the external key token, you can exchange keys between systems. It contains a key that is encrypted under a key-encrypting key.

An external key token contains an encrypted key and control information to allow compatible cryptographic systems to:
- Have a standard method of exchanging keys
- Control the use of keys through the control vector
- Merge the key with other information needed to use the key

An application obtains the external key token by using one of the callable services such as these listed. They are described in detail in Chapter 5, “Managing Symmetric Cryptographic Keys.”
- Key generate
- Key export
- Data key export

For debugging information, see Appendix B, “Key Token Formats” for the format of an external key token.

If the first byte of the key identifier is X’00’, the key identifier is interpreted as a null key token. Use the null key token to import a key from a system that cannot produce external key tokens. That is, if you have an 8- to 16-byte key that has been encrypted under an importer key, but is not imbedded within a token, place the encrypted key in a null key token and then invoke the key import callable service to get the key in operational form.

For debugging information, see Appendix B, “Key Token Formats” for the format of a null key token.

**Control Vector**

A unique control vector exists for each type of key the master key enciphers. The cryptographic feature exclusive ORs the master key with the control vector.
associated with the type of key the master key will encipher. The control vector ensures that an operational key is only used in cryptographic functions for which it is intended. For example, the control vector for an input PIN-encrypting key ensures that such a key can be used only in the Encrypted PIN translate and Encrypted PIN verify functions.

**Types of Keys**

The cryptographic keys are grouped into these categories based on the functions they perform.

- **DES master key.** The DES master key is a double-length (128 bits) key used only to encrypt other keys. The ICSF administrator installs and changes the DES master key (see z/OS Cryptographic Services ICSF Administrator's Guide for details). The administrator does this by using the Master Key Entry panels or the optional Trusted Key Entry (TKE) workstation.

  The master key always remains in a secure area in the cryptographic facility. It is used only to encipher and decipher keys. Other keys also encipher and decipher keys and are mostly used to protect cryptographic keys you transmit on external links. These keys, while on the system, are also encrypted under the master key.

- **AES master key.** The AES master key is a 32–byte (256 bits) key used only to encrypt other keys. The ICSF administrator installs and changes the AES master key (see z/OS Cryptographic Services ICSF Administrator's Guide for details). The administrator does this by using the Master Key Entry panels or the optional Trusted Key Entry (TKE) workstation (TKE V5.3).

  The master key always remains in a secure area in the cryptographic facility. It is used only to encipher and decipher keys. Other keys also encipher and decipher keys and are mostly used to protect cryptographic keys you transmit on external links. These keys, while on the system, are also encrypted under the master key.

- **AES Data-encrypting keys.** The AES data-encrypting keys are 128-, 192- and 256-bits keys that protect data privacy. If you intend to use a data-encrypting key for an extended period, you can store it in the CKDS so that it will be reenciphered if the master key is changed.

- **DES Data-encrypting keys.** The DES data-encrypting keys are single-length (64-bit), double-length (128-bit), or triple-length (192-bit) keys that protect data privacy. Single-length data-encrypting keys can also be used to encode and decode data and authenticate data sent in messages. If you intend to use a data-encrypting key for an extended period, you can store it in the CKDS so that it will be reenciphered if the master key is changed.

  You can use single-length data-encrypting keys in the encipher, decipher, encode, and decode callable services to manage data and also in the MAC generation and MAC verification callable services. Double-length and triple-length data-encrypting keys can be used in the encipher and decipher callable services for more secure data privacy. DATAC is also a double-length data encrypting key.

  Single-length data-encrypting keys can be exported and imported using the ANSI X9.17 key management callable services.

- **Data-translation keys.** The data-translation keys are single-length (64 bits) keys used for the ciphertext translate callable service as either the input or the output data-transport key.

  **Restriction:** Data-translation keys are only supported on the IBM @server zSeries 800 and the IBM @server zSeries 900.
• **CIPHER keys.** These consist of CIPHER, ENCIPHER and DECIPHER keys. They are single and double length keys for enciphering and deciphering data.

  **Note:** Double-length CIPHER, ENCIPHER and DECIPHER keys are only supported on the IBM @server zSeries 990, IBM @server zSeries 890, z9 EC, z9 BC, z10 EC and z10 BC with a PCIXCC, CEX2C, or CEX3C.

• **MAC keys.** The MAC keys are single-length (64 bits - MAC and MACVER) and double-length (128 bits - DATAM and DATAMV) keys used for the callable services that generate and verify MACs.

  With a PCIXCC, CEX2C, or CEX3C, MAC and MACVER can be single or double length keys.

• **PIN keys.** The personal identification number (PIN) is a basis for verifying the identity of a customer across financial industry networks. PIN keys are used in cryptographic functions to generate, translate, and verify PINs, and protect PIN blocks. They are all double-length (128 bits) keys. PIN keys are used in the Clear PIN generate, Encrypted PIN verify, and Encrypted PIN translate callable services.

  For installations that do not support double-length 128-bit keys, effective single-length keys are provided. For a single-length key, the left key half of the key equals the right key half.

  “Managing Personal Authentication” on page 51 gives an overview of the PIN algorithms you need to know to write your own application programs.

• **Transport keys (or key-encrypting keys).** Transport keys are also known as key-encrypting keys. They are double-length (128 bits) keys used to protect keys when you distribute them from one system to another.

  There are several types of transport keys:
  - **Exporter or OKEYXLAT key-encrypting key** protects keys of any type that are sent from your system to another system. The exporter key at the originator is the same key as the importer key of the receiver.
  - **Importer or IKEYXLAT key-encrypting key** protects keys of any type that are sent from another system to your system. It also protects keys that you store externally in a file that you can import to your system at another time. The importer key at the receiver is the same key as the exporter key at the originator.
  - **NOCV Importers and Exporters** are key-encrypting keys used to transport keys with systems that do not recognize key-encrypting key variants. There are some requirements and restrictions for the use of NOCV key-encrypting keys:
    - On CCF systems, installation of NOCV enablement keys on the CKDS is required.
    - On PCIXCC, CEX2C, and CEX3C systems, use of NOCV IMPORTERs and EXPORTERs is controlled by access control points.
    - Only programs in system or supervisor state can use the NOCV key-encrypting key in the form of tokens in callable services. Any problem program may use NOCV key-encrypting key with labelnames from the CKDS.
    - NOCV key-encrypting key on the CKDS should be protected by RACF.
    - NOCV key-encrypting key can be used to encrypt single or double length keys with standard CVs for key types DATA, DATAC, DATAM, DATAXLAT, EXPORTER, IKEYXLAT, IMPORTER, IPINENC, single-length MAC, single-length MACVER, OKEYXLAT, OPINENC, PINGEN and PINVER.
- With PCIXCCs, CEX2Cs, and CEX3Cs, NOCV key-encrypting keys can be used with triple length DATA keys. Since DATA keys have 0 CVs, processing will be the same as if the key-encrypting keys are standard key-encrypting keys (not the NOCV key-encrypting key).

**Note:** Transport keys replace local, remote, and cross keys used by PCF.

You use key-encrypting keys to protect keys that are transported using any of these services: data key export, key export, key import, clear key import, multiple clear key import, secure key import, multiple secure key import, key generate, and key translate.

For installations that do not support double-length key-encrypting keys, effective single-length keys are provided. For an effective single-length key, the clear key value of the left key half equals the clear key value of the right key half.

- **ANSI X9.17 key-encrypting keys.** These bidirectional key-encrypting keys are used exclusively in ANSI X9.17 key management. They are either single-length (64 bits) or double-length (128 bits) keys used to protect keys when you distribute them from one system to another according to the ANSI X9.17 protocol.

  **Note:** ANSI X9.17 keys are only supported on the IBM @server zSeries 800 and the IBM @server zSeries 900.

- **Key-Generating Keys.** Key-generating keys are double-length keys used to derive unique-key-per-transaction keys.

**Other Considerations**

These are considerations for keys held in the cryptographic key data set (CKDS) or by applications.

- ICSF ensures that keys held in the CKDS are reenciphered during the master key change. Keys with a long life span (more than one master key change) should be stored in the CKDS.

- Keys enciphered under the host DES master key and held by applications are automatically reenciphered under a new master key as they are used. Keys with a short life span (for example, VTAM SLE data keys) do not need to be stored in the CKDS. However, if you have keys with a long life span and you do not store them in the CKDS, they should be enciphered under the importer key-encrypting key. The importer key-encrypting key itself should be stored in the CKDS.

**Table 2** describes the key types.

You can build, generate, import, or export key types DECIPHER, ENCIPHER, CIPHER, CVARDEC, and CVARPINE on a CCF system, but they are not usable on CCF systems. They will be usable by ICSF if running on a z990, z890, z9 EC, z9 BC, z10 EC and z10 BC with a PCIXCC, CEX2C, or CEX3C.

**Table 2. Descriptions of Key Types**

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>AESDATA</td>
<td>Data encrypting key. Use the AES 128-, 192- or 256-bit key to encipher and decipher data.</td>
</tr>
<tr>
<td>AESTOKEN</td>
<td>May contain an AES key.</td>
</tr>
<tr>
<td>AKEK</td>
<td>Single-length or double-length, bidirectional key-encrypting key used for the ANSI X9.17 key management callable services. AKEK keys are only supported on the IBM @server zSeries 800 and the IBM @server zSeries 900.</td>
</tr>
<tr>
<td>Key Type</td>
<td>Meaning</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CIPHER</td>
<td>This single or double-length DES key is used to encrypt or decrypt data. It can be used in the Encipher (CSNBENC) and Decipher (CSNBDEC) callable services.</td>
</tr>
<tr>
<td></td>
<td><strong>z800/z900 only:</strong> This is a single-length key and cannot be used in the Encipher and Decipher services.</td>
</tr>
<tr>
<td>CLRAES</td>
<td>Data encrypting key. The key value is not encrypted. Use this AES 128-, 192- or 256-bit key to encipher and decipher data.</td>
</tr>
<tr>
<td>CLRDES</td>
<td>Data encrypting key. The key value is not encrypted. Use this DES single-length, double-length, or triple-length key to encipher and decipher data.</td>
</tr>
<tr>
<td>CVARDEC</td>
<td>The TSS Cryptographic variable decipher verb uses a CVARDEC key to decrypt plaintext by using the Cipher Block Chaining (CBC) method. This is a single-length key.</td>
</tr>
<tr>
<td>CVARENC</td>
<td>Cryptographic variable encipher service uses a CVARENC key to encrypt plaintext by using the Cipher Block Chaining (CBC) method. This is a single-length key.</td>
</tr>
<tr>
<td>CVARPINE</td>
<td>Used to encrypt a PIN value for decryption in a PIN-printing application. This is a single-length key.</td>
</tr>
<tr>
<td>CVARXCVL</td>
<td>Used to encrypt special control values in DES key management. This is a single-length key.</td>
</tr>
<tr>
<td>CVARXCVR</td>
<td>Used to encrypt special control values in DES key management. This is a single-length key.</td>
</tr>
<tr>
<td>DATA</td>
<td>Data encrypting key. Use this DES single-length, double-length, or triple-length key to encipher and decipher data. Use the AES 128-, 192- or 256-bit key to encipher and decipher data.</td>
</tr>
<tr>
<td>DATAC</td>
<td>Used to specify a DATA-class key that will perform in the Encipher and Decipher callable services, but not in the MAC Generate or MAC Verify callable services. This is a double-length key. Only available with a PCIXCC/CEX2C/CEX3C.</td>
</tr>
<tr>
<td>DATAM</td>
<td>Double-length MAC generation key. Used to generate a message authentication code.</td>
</tr>
<tr>
<td>DATAMV</td>
<td>Double-length MAC verification key. Used to verify a message authentication code.</td>
</tr>
<tr>
<td>DATAxLAT</td>
<td>Data translation key. Use this single-length key to reencipher text from one DATA key to another. DATAxLAT keys are only supported on the IBM @server zSeries 800 and the IBM @server zSeries 900.</td>
</tr>
<tr>
<td>DECRYPT</td>
<td>This single or double-length DES key is used to decrypt data. It can be used in the Decipher (CSNBDEC) callable service.</td>
</tr>
<tr>
<td></td>
<td><strong>z800/z900 only:</strong> This is a single-length key and cannot be used in the Decipher service.</td>
</tr>
<tr>
<td>DKEYGENKY</td>
<td>Used to generate a diversified key based on the key-generating key. This is a double-length key.</td>
</tr>
<tr>
<td>ENCRYPT</td>
<td>This single or double-length DES key is used to encrypt data. It can be used in the Encipher (CSNBENC) callable service.</td>
</tr>
<tr>
<td></td>
<td><strong>z800/z900 only:</strong> This is a single-length key and cannot be used in the Encipher service.</td>
</tr>
<tr>
<td>EXPORTER</td>
<td>Exporter key-encrypting key. Use this double-length key to convert a key from the operational form into exportable form.</td>
</tr>
</tbody>
</table>
Table 2. Descriptions of Key Types (continued)

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IKEYXLAT</td>
<td>Used to decrypt an input key in the Key Translate callable service. This is a double-length key.</td>
</tr>
<tr>
<td>IMPORTER</td>
<td>Importer key-encrypting key. Use this double-length key to convert a key from importable form into operational form.</td>
</tr>
<tr>
<td>IMP-PKA</td>
<td>Double-length limited-authority importer key used to encrypt PKA private key values in PKA external tokens.</td>
</tr>
<tr>
<td>IPINENC</td>
<td>Double-length input PIN-encrypting key. PIN blocks received from other nodes or automatic teller machine (ATM) terminals are encrypted under this type of key. These encrypted PIN blocks are the input to the Encrypted PIN translate, Encrypted PIN verify, and Clear PIN Generate Alternate services. If an encrypted PIN block is contained in the output of the SET Block Decompose service, it may be encrypted by an IPINENC key.</td>
</tr>
<tr>
<td>KEYGENKY</td>
<td>Used to generate a key based on the key-generating key. This is a double-length key.</td>
</tr>
<tr>
<td>MAC</td>
<td>Single or double-length MAC generation key. Use this key to generate a message authentication code.</td>
</tr>
<tr>
<td></td>
<td>z800/z900 only: This is a single-length key.</td>
</tr>
<tr>
<td>MACVER</td>
<td>Single or double-length MAC verification key. Use this key to verify a message authentication code.</td>
</tr>
<tr>
<td></td>
<td>z800/z900 only: This is a single-length key.</td>
</tr>
<tr>
<td>OKEYXLAT</td>
<td>Used to encrypt an output key in the Key Translate callable service. This is a double-length key.</td>
</tr>
<tr>
<td>OPINENC</td>
<td>Output PIN-encrypting key. Use this double-length output key to translate PINs. The output PIN blocks from the Encrypted PIN translate, Encrypted PIN generate, and Clear PIN generate alternate callable services are encrypted under this type of key. If an encrypted PIN block is contained in the output of the SET Block Decompose service, it may be encrypted by an OPINENC key.</td>
</tr>
<tr>
<td>PINGEN</td>
<td>PIN generation key. Use this double-length key to generate PINs.</td>
</tr>
<tr>
<td>PINVER</td>
<td>PIN verification key. Use this double-length key to verify PINs.</td>
</tr>
<tr>
<td>SECMSG</td>
<td>Used to encrypt PINs or keys in a secure message. This is a double-length key.</td>
</tr>
<tr>
<td>TOKEN</td>
<td>A key token that may contain a key.</td>
</tr>
</tbody>
</table>

Clear Keys

A clear key is the base value of a key, and is not encrypted under another key. Encrypted keys are keys whose base value has been encrypted under another key.

There are four callable services you can use to convert a clear key to an encrypted key:

- To convert a clear key to an encrypted data key in operational form, use either the Clear Key Import callable service or the Multiple Clear Key Import callable service.
- To convert a clear key to an encrypted key of any type, in operational or importable form, use either the Secure Key Import callable service or the Multiple Secure Key Import callable service.
Note: The Secure Key Import and Multiple Secure Key Import callable services can only execute in special secure mode.

Clear key DATA tokens can be stored in the CKDS. These tokens can only be used by symmetric key decipher and symmetric key encipher callable services for the DES and AES algorithms.

Generating and Managing Symmetric Keys

Using ICSF, you can generate keys using either the key generator utility program or the key generate callable service. The dynamic CKDS update callable services allow applications to directly manipulate the CKDS. ICSF provides callable services that support DES and AES key management as defined by the IBM Common Cryptographic Architecture (CCA) and by the ANSI X9.17 standard.

The next few topics describe the key generating and management options ICSF provides.

Key Generator Utility Program

The key generator utility program generates data, data-translation, MAC, PIN, and key-encrypting keys, and enciphers each type of key under a specific master key variant. When the KGUP generates a key, it stores it in the cryptographic key data set (CKDS).

Note: If you specify CLEAR, KGUP uses the random number generate and secure key import callable services rather than the key generate service.

You can access KGUP using ICSF panels. The KGUP path of these panels helps you create the JCL control statements to control the key generator utility program. When you want to generate a key, you can enter the ADD control statement and information, such as the key type on the panels. For a detailed description of the key generator utility program and how to use it to generate keys, see Cryptographic Services ICSF Administrator's Guide.

Common Cryptographic Architecture DES Key Management Services

ICSF provides callable services that support CCA key management for DES keys.

Clear Key Import Callable Service (CSNBCKI)

This service imports a clear DATA key that is used to encipher or decipher data. It accepts a clear key and enciphers the key under the host master key, returning an encrypted DATA key in operational form in an internal key token.

Control Vector Generate Callable Service (CSNBCVG)

The control vector generate callable service builds a control vector from keywords specified by the key_type and rule_array parameters.

Control Vector Translate Callable Service (CSNBCVT)

The control vector translate callable service changes the control vector used to encipher an external key. Use of this service requires the optional PCI Cryptographic Coprocessor.

Cryptographic Variable Encipher Callable Service (CSNBCVE)

The cryptographic variable encipher callable service uses a CVARENC key to encrypt plaintext by using the Cipher Block Chaining (CBC) method. You can use
this service to prepare a mask array for the control vector translate service. The plaintext must be a multiple of eight bytes in length.

**Data Key Export Callable Service (CSNBDKX)**

This service reenciphers a DATA key from encryption under the master key to encryption under an exporter key-encrypting key, making it suitable for export to another system.

**Data Key Import Callable Service (CSNBDKM)**

This service imports an encrypted source DES single-length or double-length DATA key and creates or updates a target internal key token with the master key enciphered source key.

**Diversified Key Generate Callable Service (CSNBDKG)**

The diversified key generate service generates a key based on the key-generating key, the processing method, and the parameter supplied. The control vector of the key-generating key also determines the type of target key that can be generated.

**Key Export Callable Service (CSNBKEX)**

This service reenciphers any type of key (except an AKEK or IMP-PKA key) from encryption under a master key variant to encryption under the same variant of an exporter key-encrypting key, making it suitable for export to another system.

**Key Generate Callable Service (CSNBKGN)**

The key generate callable service generates data, data-translation, MAC, PIN, and key-encrypting keys. It generates a single key or a pair of keys. Unlike the key generator utility program, the key generate service does not store the keys in the CKDS where they can be saved and maintained. The key generate callable service returns the key to the application program that called it. The application program can then use a dynamic CKDS update service to store the key in the CKDS.

When you call the key generate callable service, include parameters specifying information about the key you want generated. Because the form of the key restricts its use, you need to choose the form you want the generated key to have. You can use the **key_form** parameter to specify the form. The possible forms are:

- **Operational**, if the key is used for cryptographic operations on the local system. Operational keys are protected by master key variants and can be stored in the CKDS or held by applications in internal key tokens.
- **Importable**, if the key is stored with a file or sent to another system. Importable keys are protected by importer key-encrypting keys.
- **Exportable**, if the key is transported or exported to another system and imported there for use. Exportable keys are protected by exporter key-encrypting keys and cannot be used by ICSF callable service.

Importable and exportable keys are contained in external key tokens. For more information on key tokens, refer to [*"Key Token" on page 17.*](#)

**Key Import Callable Service (CSNBKIM)**

This service reenciphers a key (except an AKEK) from encryption under an importer key-encrypting key to encryption under the master key. The reenciphered key is in the operational form.

**Key Part Import Callable Service (CSNBKPI)**

This service combines clear key parts of any key type and returns the combined key value either in an internal token or as an update to the CKDS.
**Key Test Callable Service (CSNBKYT and CSNBKYTX)**
This service generates or verifies a secure cryptographic verification pattern for keys. A parameter indicates the action you want to perform.

The key to test can be in the clear or encrypted under a master key. The key test extended callable service works on keys encrypted under a KEK.

For generating a verification pattern, the service creates and returns a random number with the verification pattern. For verifying a pattern, you supply the random number from the call to the service that generated the pattern.

**Key Token Build Callable Service (CSNBKTB)**
The key token build callable service is a utility you can use to create skeleton key tokens for AKEKs as input to the key generate or key part import callable service. You can also use this service to build CCA key tokens for all key types ICSF supports. You can also use this service to build CCA key tokens for all key types ICSF supports.

**Key Translate Callable Service (CSNBKTR)**
This service uses one key-encrypting key to decipher an input key and then enciphers this key using another key-encrypting key within the secure environment.

**Multiple Clear Key Import Callable Service (CSNBCKM)**
This service imports a single-length, double-length, or triple-length clear DATA key that is used to encipher or decipher data. It accepts a clear key and enciphers the key under the host master key, returning an encrypted DATA key in operational form in an internal key token.

**Multiple Secure Key Import Callable Service (CSNBSKM)**
This service enciphers a single-length, double-length, or triple-length clear key under the host master key or under an importer key-encrypting key. The clear key can then be imported as any of the possible key types. Triple-length keys can only be imported as DATA keys. This service can be used only when ICSF is in special secure mode and does not allow the import of an AKEK.

**Prohibit Export Callable Service (CSNBPEX)**
This service modifies an operational key so that it cannot be exported. This callable service does not support NOCV key-encrypting keys, DATA, MAC, or MACVER keys with standard control vectors (for example, control vectors supported by the Cryptographic Coprocessor Feature).

**Prohibit Export Extended Callable Service (CSNBPEXX)**
This service updates the control vector in the external token of a key in exportable form so that the receiver node can import the key but not export it. When the key import callable service imports such a token, it marks the token as non-exportable. The key export callable service does not allow export of this token.

**Random Number Generate Callable Service (CSNBRNG and CSNBRNGL)**
The random number generate callable service creates a random number value to use in generating a key. The callable service uses cryptographic hardware to generate a random number for use in encryption.

**Remote Key Export Callable Service (CSNDRKX)**
The remote key export callable service uses the trusted block to generate or export DES keys for local use and for distribution to an ATM or other remote device.
Secure Key Import Callable Service (CSNBSKI)
This service enciphers a clear key under the host master key or under an importer key-encrypting key. The clear key can then be imported as any of the possible key types. This service can be used only when ICSF is in special secure mode and does not allow the import of an AKEK.

Note: The PKA encrypt, PKA decrypt, symmetric key generate, symmetric key import, and symmetric key export callable services provide a way of distributing DES DATA keys protected under a PKA key. See Chapter 3, "Introducing PKA Cryptography and Using PKA Callable Services," on page 69 for additional information.

Symmetric Key Export Callable Service (CSNDSYX and CSNFSYX)
This service transfers an application-supplied symmetric key (a DATA key) from encryption under the DES host master key to encryption under an application-supplied RSA public key. (There are two types of PKA public key tokens: RSA and DSS. This callable service can use only the RSA type.) The application-supplied DATA key must be an ICSF DES internal key token or the label of such a token in the CKDS. The symmetric key import callable service can import the PKA-encrypted form at the receiving node.

Symmetric Key Generate Callable Service (CSNDSYG)
This service generates a symmetric key (that is, a DATA key) and returns it encrypted using DES and encrypted under an RSA public key token. (There are two types of PKA public key tokens: RSA and DSS. This callable service can use only the RSA type.)

The DES-encrypted key can be an internal token encrypted under a host DES master key, or an external form encrypted under a KEK. (You can use the symmetric key import callable service to import the PKA-encrypted form.)

Symmetric Key Import Callable Service (CSNDSYI and CSNFSYI)
This service imports a symmetric (DES) DATA key enciphered under an RSA public key. (There are two types of PKA private key tokens: RSA and DSS. This callable service can use only the RSA type.) This service returns the key in operational form, enciphered under the DES master key.

Transform CDMF Key Callable Service (CSNBTCX)
Restriction: This service is only available on the IBM @server zSeries 800 and the IBM @server zSeries 900.

It changes a CDMF DATA key in an internal or external token to a transformed shortened DES key. It ignores the input internal DES token markings and marks the output internal token for use in the DES. You need to use this service only if you have a CDMF or DES-CDMF system that needs to send CDMF-encrypted data to a DES-only system. The CDMF or DES-CDMF system must generate the key, shorten it, and pass it to the DES-only system.

If the input DATA key is in an external token, the operational KEK must be marked as DES or SYS-ENC. The service fails for an external DATA key encrypted under a KEK that is marked as CDMF.

Trusted Block Create Callable Service (CSNDTBC)
This service creates and activates a trusted block under two step process.
**User Derived Key Callable Service (CSFUDK)**

**Restriction:** This service is only available on the IBM Eserver zSeries 800 and the IBM Eserver zSeries 900.

This service generates a single-length or double-length MAC key, or updates an existing user-derived key. A single-length MAC key can be used to compute a Message Authentication Code (MAC) following the ANSI X9.9, ANSI X9.19, or the Europay, MasterCard, Visa (EMV) Specification MAC processing rules. A double-length MAC key can be used to compute a MAC following the ANSI X9.19 optional double MAC processing rule or the EMV rules.

**Common Cryptographic Architecture AES Key Management Services**

ICSF provides callable services that support CCA key management for AES keys.

**Key Generate Callable Service (CSNBKGN)**

The key generate callable service generates AES data keys. It generates a single operational key. Unlike the key generator utility program, the key generate service does not store the keys in the CKDS where they can be saved and maintained. The key generate callable service returns the key to the application program that called it. The application program can then use a dynamic CKDS update service to store the key in the CKDS.

**Key Token Build Callable Service (CSNBKTB)**

The key token build callable service is a utility you can use to create clear AES key tokens, secure AES key tokens and skeleton secure AES key tokens for use with other callable services. You can also use this service to build CCA key tokens for all key types ICSF supports. You can also use this service to build CCA key tokens for all key types ICSF supports.

**Multiple Clear Key Import Callable Service (CSNBCKM)**

This service imports a a 128-, 192- or 256-bit clear DATA key that is used to encipher or decipher data. It accepts a clear key and enciphers the key under the host master key, returning an encrypted DATA key in operational form in an internal key token.

**Multiple Secure Key Import Callable Service (CSNBSKM)**

This service enciphers 128-, 192- or 256-bit clear DATA key under the host master key. This service can be used only when ICSF is in special secure mode.

**Symmetric Key Export Callable Service (CSNDSYX and CSNFSYX)**

This service transfers an application-supplied symmetric key (a DATA key) from encryption under the AES host master key to encryption under an application-supplied RSA public key. (There are two types of PKA public key tokens: RSA and DSS. This callable service can use only the RSA type.) The application-supplied DATA key must be an ICSF AES internal key token or the label of such a token in the CKDS. The symmetric key import callable service can import the PKA-encrypted form at the receiving node.

**Symmetric Key Generate Callable Service (CSNDSYG)**

This service generates a symmetric DATA key and returns it encrypted under the host AES master key and encrypted under an RSA public key token. (There are two types of PKA public key tokens: RSA and DSS. This callable service can use only the RSA type.)
The AES-encrypted key can only be an internal token encrypted under a host AES master key. You can use the symmetric key import callable service to import the PKA-encrypted form.

**Symmetric Key Import Callable Service (CSNDSYI and CSNFSYI)**

This service imports a symmetric (AES) DATA key enciphered under an RSA public key. (There are two types of PKA private key tokens: RSA and DSS. This callable service can use only the RSA type.) This service returns the key in operational form, enciphered under the AES master key.

**Improved remote key distribution**

**Note:** This improved remote key distribute support is only available on the z9 EC, z9 BC, z10 EC and z10 BC servers.

New methods have been added for securely transferring symmetric encryption keys to remote devices, such as Automated Teller Machines (ATMs), PIN-entry devices, and point of sale terminals. These methods can also be used to transfer symmetric keys to another cryptographic system of any type, such as a different kind of Hardware Security Module (HSM) in an IBM or non-IBM computer server. This change is especially important to banks, since it replaces expensive human operations with network transactions that can be processed quickly and inexpensively. This method supports a variety of requirements, fulfilling the new needs of the banking community while simultaneously making significant interoperability improvements to related cryptographic key-management functions.

For the purposes of this description, the ATM scenario will be used to illustrate operation of the new methods. Other uses of this method are also valuable.

**Remote Key Loading**

Remote key loading refers to the process of installing symmetric encryption keys into a remotely located device from a central administrative site. This encompasses two phases of key distributions.

- Distribution of initial key encrypting keys (KEKs) to a newly installed device. A KEK is a type of symmetric encryption key that is used to encrypt other keys so they can be securely transmitted over unprotected paths.
- Distribution of operational keys or replacement KEKs, enciphered under a KEK currently installed in the device.

Access control points are assigned to roles to control keyword usage in the services provided for ATM Remote Key Loading. These access control points are used by the ATM Remote Key Loading function.

**Table 3. Access Control Points Used by ATM Remote Key Loading**

<table>
<thead>
<tr>
<th>API</th>
<th>Access Control Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNDTBC</td>
<td>0X'030F'</td>
<td>030F Create a Trusted Block in inactive form</td>
</tr>
<tr>
<td>CSNDTBC</td>
<td>0X'0310'</td>
<td>0310 Activate an inactive Trusted Block</td>
</tr>
<tr>
<td>CSNDPKI</td>
<td>0X'0311'</td>
<td>0311 Convert Trusted Block from external to internal form</td>
</tr>
<tr>
<td>CSNDPKI</td>
<td>0X'0104'</td>
<td>PKA Key Import</td>
</tr>
<tr>
<td>CSNDRKX</td>
<td>0X'0312'</td>
<td>0312 Generate or export a key for use by a non-CCA node</td>
</tr>
</tbody>
</table>
Old remote key loading example: Use an ATM as an example of the remote key loading process. A new ATM has none of the bank's keys installed when it is delivered from the manufacturer. The process of getting the first key securely loaded is difficult. This has typically been done by loading the first KEK into each ATM manually, in multiple cleartext key parts. Using dual control for key parts, two separate people must carry key part values to the ATM, then load each key part manually. Once inside the ATM, the key parts are combined to form the actual KEK. In this manner, neither of the two people has the entire key, protecting the key value from disclosure or misuse. This method is labor-intensive and error-prone, making it expensive for the banks.

New remote key loading methods: New remote key loading methods have been developed to overcome some of the shortcomings of the old manual key loading methods. These new methods define acceptable techniques using public key cryptography to load keys remotely. Using these new methods, banks will be able to load the initial KEKs without sending people to the remote device. This will reduce labor costs, be more reliable, and be much less expensive to install and change keys. The new cryptographic features added provide new methods for the creation and use of the special key forms needed for remote key distribution of this type. In addition, they provide ways to solve long-standing barriers to secure key exchange with non-IBM cryptographic systems.

Once an ATM is in operation, the bank can install new keys as needed by sending them enciphered under a KEK installed previously. This is straightforward in concept, but the cryptographic architecture in ATMs is often different from that of the host system sending the keys, and it is difficult to export the keys in a form understood by the ATM. For example, cryptographic architectures often enforce key-usage restrictions in which a key is bound to data describing limitations on how it can be used - for encrypting data, for encrypting keys, for operating on message authentication codes (MACs), and so forth. The encoding of these restrictions and the method used to bind them to the key itself differs among cryptographic architectures, and it is often necessary to translate the format to that understood by the target device prior to a key being transmitted. It is difficult to do this without reducing security in the system; typically it is done by making it possible to arbitrarily change key-usage restrictions. The methods described here provide a mechanism through which the system owner can securely control these translations, preventing the majority of attacks that could be mounted by modifying usage restrictions.

Table 3. Access Control Points Used by ATM Remote Key Loading (continued)

<table>
<thead>
<tr>
<th>API</th>
<th>Access Control Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNDRKX</td>
<td>0x’00DB’</td>
<td>Replication of a single length source key (which is either an RKX token or a CCA token) if the output symmetric encryption result is to be a CCA token, and the CV in the trusted block's Common Export Key Parameters TLV Object is 16 bytes with key form bits 'fff' set to X'010' for the left half and X'001' for the right half.</td>
</tr>
<tr>
<td>CSNDRKX</td>
<td>0x’027B’</td>
<td>The importer key identifier in the initial code release must have unique halves.</td>
</tr>
<tr>
<td>CSNDRKX</td>
<td>0x’0276’</td>
<td>The transport key identifier in the initial code release must have unique halves.</td>
</tr>
</tbody>
</table>

z/OS V1R11 ICSF Application Programmer’s Guide
A new data structure called a trusted block is defined to facilitate the remote key loading methods. The trusted block is the primary vehicle supporting these new methods.

**Trusted block**

The trusted block is the central data structure to support all remote key loading functions. It provides great power and flexibility, but this means that it must be designed and used with care in order to have a secure system. This security is provided through several features of the design.

- A two step process is used to create a trusted block.
- The trusted block includes cryptographic protection that prevents any modification when it is created.
- A number of fields in the rules of a trusted block offer the ability to limit how the block is used, reducing the risk of it being used in unintended ways or with unintended keys.

The trusted block is the enabler which requires secure approval for its creation, then enables the export or generation of DES and TDES keys in a wide variety of forms as approved by the administrators who created the trusted block. For added security, the trusted blocks themselves can be created on a separate system, such as an xSeries server with an IBM 4764 Cryptographic Coprocessor, locked in a secure room. The trusted block can subsequently be imported into the zSeries server where they will be used to support applications.

There are two new CCA callable services to manage and use trusted blocks: Trusted_Block_Create (CSNDTBC) and Remote_Key_Export (CSNDRKX). CSNDTBC creates a trusted block, and CSNDRKX uses a trusted block to generate or export DES keys according to the parameters in the trusted block. The trusted block consists of a header followed by several sections. Some elements are required, while others are optional.

Figure 2 on page 32 shows the contents of a trusted block. The elements shown in the table give an overview of the structure and do not provide all of the details of a trusted block.
Here is a brief description of the elements that are depicted.

**Structure version information** - This identifies the version of the trusted block structure. It is included so that code can differentiate between this trusted block layout and others that may be developed in the future.

**Public key** - This contains the RSA public key and its attributes. For distribution of keys to a remote ATM, this will be the root certification key for the ATM vendor, and it will be used to verify the signature on public-key certificates for specific individual ATMs. In this case, the Trusted Block will also contain Rules that will be used to generate or export symmetric keys for the ATMs. It is also possible for the Trusted Block to be used simply as a trusted public key container, and in this case the Public Key in the block will be used in general-purpose cryptographic functions such as digital signature verification. The public key attributes contain information on key usage restrictions. This is used to securely control what operations will be permitted to use the public key. If desired, the public key can be restricted to use for only digital signature operations, or for only key management operations.

**Trusted block protection information** - This topic contains information that is used to protect the Trusted Block contents against modification. According to the method in ISO 16609, a CBC-mode MAC is calculated over the Trusted Block using a randomly-generated triple-DES (TDES) key, and the MAC key itself is encrypted.
and embedded in the block. For the internal form of the block, the MAC key is encrypted with a randomly chosen fixed-value variant of the PKA master key. For the external form, the MAC key is encrypted with a fixed variant of a key-encrypting key. The MKVP field contains the master key verification pattern for the PKA master key that was used, and is filled with binary zeros if the trusted block is in external format. Various flag fields contain these boolean flags.

- **Active flag** - Contained within the flags field of the required trusted block information section, this flag indicates whether the trusted block is active and ready for use by other callable services. Combined with the use of two separate access control points, the active flag is used to enforce dual control over creation of the block. A person whose active role is authorized to create a trusted block in inactive form creates the block and defines its parameters. An inactive trusted block can only be used to make it active. A person whose active role is authorized to activate an inactive trusted block must approve the block by changing its status to active. See Figure 4 on page 36. The Remote_Key_Export callable service can only use an internal active trusted block to generate or export DES keys according to the parameters defined in the trusted block.

- **Date checking flag** - Contained within the optional activation and expiration date subsection of the required trusted block information subsection, this flag indicates whether the coprocessor checks the activation and expiration dates for the trusted block. If the date checking flag is on, the coprocessor compares the activation and expiration dates in the optional subsection to the coprocessor internal real-time clock, and processing terminates if either date is out of range. If this flag is off or the activation and expiration dates subsection is not defined, the device does no date checking. If this flag is off and the activation and expiration dates subsection is defined, date checking can still be performed outside of the device if required. The date checking flag enables use of the trusted block in systems where the coprocessor clock is not set.

**Trusted block name** - This field optionally contains a text string that is a name (key label) for the trusted block. It is included in the block for use by an external system such as a host computer, and not by the card itself. In the zSeries system, the label can be checked by RACF to determine if use of the block is authorized. It is possible to disable use of trusted blocks that have been compromised or need to be removed from use for other reasons by publishing a revocation list containing the key names for the blocks that must not be used. Code in the host system could check each trusted block prior to it being used in the cryptographic coprocessor, to ensure that the name from that block is not in the revocation list.

**Expiration date and activation dates** - The trusted block can optionally contain an expiration date and an activation date. The activation date is the first day on which the block can be used, and the expiration date is the last day when the block can be used. If these dates are present, the date checking flag in the trusted block will indicate whether the coprocessor should check the dates using its internal real-time clock. In the case of a system that does set the coprocessor clock, checking would have to be performed by an application program prior to using the trusted block. This is not quite as secure, but it is still valuable, and storing the dates in the block itself is preferable to making the application store it somewhere else and maintain the association between the separate trusted block and activation and expiration dates.

**Application-defined data** - The trusted block can hold data defined and understood only by the host application program. This data is included in the protected contents of the trusted block, but it is not used or examined in any way by
the coprocessor. By including its own data in the trusted block, an application can guarantee that the data is not changed in any way, since it is protected in the same way as the other trusted block contents.

**Rules** - A variable number of rules can be included in the block. Each rule contains information on how to generate or export a symmetric key, including values for variants to be used in order to provide keys in the formats expected by systems with differing cryptographic architectures. Use of the rules are described in the topics covering key generation and export using the RKX function. This table summarizes the required and optional values of each rule.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Required field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule ID</td>
<td>Yes</td>
<td>Specifies the 8-character name of the rule</td>
</tr>
<tr>
<td>Operation</td>
<td>Yes</td>
<td>Indicates whether this rule generates a new key or exports an existing key</td>
</tr>
<tr>
<td>Generated key length</td>
<td>Yes</td>
<td>Specifies the length of the key to be generated</td>
</tr>
<tr>
<td>Key-check algorithm ID</td>
<td>Yes</td>
<td>Specifies which algorithm to use to compute the optional key-check value (KCV). Options are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• No KCV</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Encrypt zeros with the key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Compute MDC-2 hash of the key</td>
</tr>
<tr>
<td>Symmetric-encrypted output format</td>
<td>Yes</td>
<td>Specifies the format of the required symmetric-encrypted key output. Options are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CCA key token</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• RKX key token</td>
</tr>
<tr>
<td>Asymmetric-encrypted output format</td>
<td>Yes</td>
<td>Specifies the format of the optional asymmetric-encrypted key output (key is encrypted with RSA). Options are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• No asymmetric-encrypted key output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Encrypt in PKCS1.2 format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Encrypt in RSAOAEP format</td>
</tr>
<tr>
<td>Transport-key variant</td>
<td>No</td>
<td>Specifies the variant to apply to the transport key prior to it being used to encrypt the key being generated or exported</td>
</tr>
<tr>
<td>Export key CV</td>
<td>No</td>
<td>Specifies the CCA CV to apply to the transport key prior to it being used to encrypt the key being generated or exported. The CV defines permitted uses for the exported key.</td>
</tr>
<tr>
<td>Export key length limits</td>
<td>No</td>
<td>Defines the minimum and maximum lengths of the key that can be exported with this rule.</td>
</tr>
<tr>
<td>Output key variant</td>
<td>No</td>
<td>Specifies the variant to apply to the generated or exported key prior to it being encrypted.</td>
</tr>
<tr>
<td>Export-key rule reference</td>
<td>No</td>
<td>Specifies the rule ID for the rule that must have been used to generate the key being exported, if that key is an RKX key token.</td>
</tr>
<tr>
<td>Export-key CV restrictions</td>
<td>No</td>
<td>Defines masks and templates to use to restrict the possible CV values that a source key can have when being exported with RKX. Only applies if the key is a CCA key token. This can control the types of CCA keys that can be processed using the rule.</td>
</tr>
</tbody>
</table>
Changes to the CCA API

These changes have been made to the CCA API to support remote key loading using trusted blocks:

- A new Trusted_Block_Create (CSNDTBC) callable service has been developed to securely create trusted blocks under dual control.
- A new Remote_Key_Export (CSNDRKX) callable service has been developed to generate or export DES and TDES keys under control of the rules contained in a trusted block.
- The Digital_Signature_Verify (CSNDDSV) callable service has been enhanced so that, in addition to verifying ordinary CCA RSA keys, it can use the RSA public key contained in a trusted block to verify digital signatures.
- The PKA_Key_Import (CSNDPKI) callable service has been enhanced so it can import an RSA key into the CCA domain. In addition, the verb can import an external format trusted block into an internal format trusted block, ready to be used in the local system.
- The PKA_Key_Token_Change (CSNDKTC) callable service has been enhanced so that it can update trusted blocks to the current PKA master key when the master key is changed. A trusted block contains an embedded MAC key enciphered under the PKA master key. When the PKA master key is changed, the outdated MAC key and the trusted block itself need to be updated to reflect the current PKA master key.
- The MAC_Generate (CSNBMGN) and MAC_Verify (CSNBMVR) callable services have been enhanced to add ISO 16609 TDES MAC support in which the text will be CBC-TDES encrypted using a double-length key and the MAC will be extracted from the last block.
- The PKA key storage callable services support trusted blocks.

The RKX key token

CCA normally uses key tokens that are designed solely for the purposes of protecting the key value and carrying metadata associated with the key to control its use by CCA cryptographic functions. The remote key loading design introduces a new type of key token called an RKX key token. The purpose of this token is somewhat different, and its use is connected directly with the CSNDRKX callable service added to CCA of the remote key loading design.

The RKX key token uses a special structure that binds the token to a specific trusted block, and allows sequences of CSNDRKX calls to be bound together as if they were an atomic operation. This allows a series of related key-management operations to be performed using the CSNDRKX callable service. These capabilities are made possible by incorporating these three features into the RKX key token structure:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Required field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Export-key label template</td>
<td>No</td>
<td>Specifies the key label of the key token that contains the source key to be exported. A key label is a name used to identify a key. The rule can optionally contain a key label template, which will be matched against the host-supplied key label, using a wildcard (*) so that the template can match a set of related key labels. The operation will only be accepted if the supplied label matches the wildcard template in the rule.</td>
</tr>
</tbody>
</table>
The key is enciphered using a variant of the MAC key that is in the trusted block. A fixed, randomly-derived variant is applied to the key prior to it being used. As a result, the enciphered key is protected against disclosure since the trusted block MAC key is itself protected at all times.

The structure includes the rule ID contained in the trusted block rule that was used to create the key. A subsequent call to the CSNDRKX callable service can use this key with a trusted block rule that references this rule ID, effectively chaining use of the two rules together securely.

A MAC is computed over the encrypted key and the rule ID, using the same MAC key that is used to protect the trusted block itself. This MAC guarantees that the key and the rule ID cannot be modified without detection, providing integrity and binding the rule ID to the key itself. In addition, the MAC will only verify if the RKX key token is used with the same trusted block that created the token, thus binding the key to that specific trusted block.

This figure shows a simplified conceptual view of the RKX key token structure.

Using trusted blocks
These examples illustrate how trusted blocks are used with the new and enhanced CCA callable services.

Creating a trusted block: This figure illustrates the steps used to create a trusted block.
A two step process is used to create a trusted block. Trusted blocks are structures that could be abused to circumvent security if an attacker could create them with undesirable settings, and the requirement for two separate and properly authorized people makes it impossible for a single dishonest employee to create such a block. A trusted block cannot be used for any operations until it is in the active state. Any number of trusted blocks can be created in order to meet different needs of application programs.

**Exporting keys with Remote Key Export:** This figure shows the process for using a trusted block in order to export a DES or TDES key. This representation is at a very high level in order to illustrate the basic flow.

![Diagram of trusted block process](image-url)

*Figure 5. Exporting keys using a trusted block*

Note: Importer key is only used if source key is an external CCA token.
The CSNDRKX callable service is called with these main parameters:

- A trusted block, in the active state, defines how the export operation is to be processed, including values to be used for things such as variants to apply to the keys.
- The key to be exported, shown previously as the source key. The source key takes one of two forms:
  1. A CCA DES key token
  2. An RKX key token
- A key-encrypting key, shown in the figure as the importer key. This is only used if the source key is an external CCA DES key token, encrypted under a KEK. In this case, the KEK is the key needed to obtain the cleartext value of the source key.
- A transport key, either an exporter KEK or an RKX key token, used to encrypt the key being exported.
- An optional public key certificate which, if included, contains the certified public key for a specific ATM. The certificate is signed with the ATM vendor’s private key, and its corresponding public key must be contained in the trusted block so that this certificate can be validated. The public key contained in the certificate can be used to encrypt the exported key.

The processing steps are simple at a high level, but there are many options and significant complexity in the details.

- The trusted block itself is validated. This includes several types of validation.
  - Cryptographic validation using the MAC that is embedded in the block, in which the MAC key is decrypted using the coprocessor’s master key, and the MAC is then verified using that key. This verifies the block has not been corrupted or tampered with, and it also verifies that the block is for use with this coprocessor since it will only succeed if the master key is correct.
  - Consistency checking and field validation, in which the validity of the structure itself is checked, and all values are verified to be within defined ranges.
  - Fields in the trusted block are checked to see if all requirements are met for use of this trusted block. One check which is always required is to ensure that the trusted block is in the active state prior to continuing. Another check, which is optional based on the contents of the trusted block, is to ensure the operation is currently allowed by comparing the date of the coprocessor real-time clock to the activation and expiration dates defined in the trusted block.
- Input parameters to the CSNDRKX callable service are validated against rules defined for them within the trusted block. For example:
  - The rule can restrict the length of the key to be exported.
  - The rule can restrict the control vector values for the key to be exported, so only certain key types can be exported with that rule.
- When the export key is decrypted, the rules embedded in the trusted block are then used to modify that key to produce the desired output key value. For example, the trusted block can contain a variant to be exclusive-ORed with the source key prior to when that key is encrypted. Many non-IBM cryptographic systems use variants to provide key separation to restrict a key from improper use.
- A key check value (KCV) can be optionally computed for the source key. If the KCV is computed, the trusted block allows for one of two key check algorithms to be used: (1) encrypting binary zeros with the key, or (2) computing an MDC-2 hash of the key. The KCV is returned as an output of the CSNDRKX function.
The export key, which could possibly be modified with a variant according to the rules in the trusted block, is enciphered with the transport key. The rules can specify that the key be created in one of two formats: (1) a CCA key token, or (2) the new RKX key token, described previously. With proper selection of rule options, the CCA key token can create keys that can be used in non-CCA systems. The key value can be extracted from the CCA key token resulting in a generic encrypted key, with variants and other options as defined in the rule.

Two optional fields in the trusted block may modify the transport key prior to it being used to encrypt the source key:

- The trusted block can contain a CCA control vector (CV) to be exclusive-ORed with the transport key prior to it being used to encrypt the export key. This exclusive-OR process is the standard way CCA applies a CV to a key.
- In addition to the CV described previously, the trusted block can also contain a variant to be exclusive-ORed with the transport key prior to its use.

If a variant and CV are both present in the trusted block, the variant is applied first, then the CV.

The export key can optionally be encrypted with the RSA public key identified by the certificate parameter of the CSNDRKX callable service, in addition to encrypting it with the transport key as described previously. These two encrypted versions of the export key are provided as separate outputs of the CSNDRKX callable service. The trusted block allows a choice of encrypting the key in either PKCS1.2 format or PKCSSOAEP format.

**Generating keys with Remote Key Export:** This figure shows the process for using a trusted block to generate a new DES or TDES key. This representation is at a very high level in order to illustrate the basic flow.
For key generation, the CSNDRKX callable service is called with these main parameters:

- A trusted block, in the internal active state, which defines how the key generation operation is to be processed, including values to be used for things such as variants to apply to the keys. The generated key is encrypted by a variant of the MAC key contained in a trusted block.

- An optional public key certificate which, if included, contains the certified public key for a specific ATM. The certificate is signed with the ATM vendor’s private key, and its corresponding public key must be contained in the trusted block so that this certificate can be validated. The public key contained in the certificate can be used to encrypt the generated key.
The processing steps are simple at a high level, but there are many options and significant complexity in the details. Most of the processing steps are the same as those described previously for key export. Therefore, only those processing steps that differ are described here in detail.

- Validation of the trusted block and input parameters is done as described for export previously.
- The DES or TDES key to be returned by the CSNDRKX callable service is randomly generated. The trusted block indicates the length for the generated key.
- The output key value is optionally modified by a variant as described previously for export, and then encrypted in the same way as for export using the Transport key and optionally the public key in the certificate parameter.
- The key check value (KCV) is optionally computed for the generated key using the same method as for an exported key.

**Remote key distribution scenario**

The new and modified CCA functions for remote key loading are used together to create trusted blocks, and then generate or export keys under the control of those trusted blocks. This figure summarizes the flow of the CCA functions to show how they are used:

![Diagram of the CCA functions for remote key loading](image)

**Figure 7. Typical flow of callable services for remote key export**
**Usage example:** The scenario described shows how these functions might be combined in a real-life application to distribute a key to an ATM and keep a copy for local use. Some of the terminology used reflects typical terms used in ATM networks. The example illustrates a fairly complex real-world key distribution scenario, in which these values are produced.

- A TMK (Terminal Master Key), which is the root KEK used by the ATM to exchange other keys, is produced in two forms: (1) encrypted under the ATM public key, so it can be sent to the ATM, and (2) as an RKX key token that will be used in subsequent calls to the CSNDRKX callable service to produce other keys.
- A key-encrypting key KEK1 that is encrypted under the TMK in a form that can be understood by the ATM.
- A PIN-encrypting key PINKEY be used by the ATM to encrypt customer-entered PINs and by the host to verify those PINs. The PINKEY is produced in two forms: (1) encrypted under KEK1 in a form that can be understood by the ATM, and (2) as a CCA internal DES key token with the proper PIN-key CV, encrypted under the CCA DES master key and suitable for use with the coprocessor.

It takes seven steps to produce these keys using the CSNDRKX callable service. These steps use a combination of five rules contained in a single trusted block. The rules in this example are referred to as GENERAT1, GENERAT2, EXPORT1, EXPORT2, and EXPORT3.

1. Use CSNDRKX with rule ID "GENERAT1" to generate a TMK for use with the ATM. The key will be output in two forms:
   a. $e_{Pu}(TMK)$: Encrypted under the ATM public key, supplied in the certificate parameter, CERT
   b. RKX(TMK): As an RKX key token, suitable for subsequent input to the CSNDRKX callable service
2. Use CSNDRKX with rule ID "GENERAT2" to generate a key-encrypting key (KEK1) as an RKX key token, RKX(KEK1)
3. Use CSNDRKX with rule ID "GENERAT2" to generate a PIN key (PINKEY) as an RKX key token: RKX(PINKEY).
4. Use CSNDRKX with rule ID "EXPORT1" to export KEK1 encrypted under the TMK as a CCA DES key token using a variant of zeros applied to the TMK. This produces $e_{TMK}(KEK1)$.
5. Use CSNDRKX with rule ID "EXPORT2" to export PINKEY encrypted under KEK1 as a CCA token using a variant of zeros applied to KEK1. This produces $e_{KEK1}(PINKEY)$.
6. Use CSNDRKX with rule ID "EXPORT3" to export PINKEY under KEK2, an existing CCA key-encrypting key on the local server. This produces $e_{KEK2}(PINKEY)$, with the CCA control vector for a PIN key.
7. Use CSNBKIM to import the PINKEY produced in step 6 into the local system as an operational key. This produces $e_{MK}(PINKEY)$, a copy of the key encrypted under the local DES master key (MK) and ready for use by CCA PIN API functions.

**Remote key distribution benefits**
CCA support for remote key loading solves one new problem, and one long-standing problem. This support allows the distribution of initial keys to ATMs and other remote devices securely using public-key techniques, in a flexible way that can support a wide variety of different cryptographic architectures. They also make it far easier and far more secure to send keys to non-CCA systems when
those keys are encrypted with a triple-DES key-encrypting key. These changes make it easier for customers to develop more secure systems.

Diversifying keys

CCA supports several methods for diversifying a key using the diversified key generate callable service. Key-diversification is a technique often used in working with smart cards. In order to secure interactions with a population of cards, a "key-generating key" is used with some data unique to a card to derive ("diversify") keys for use with that card. The data is often the card serial number or other quantity stored on the card. The data is often public, and therefore it is very important to handle the key-generating key with a high degree of security lest the interactions with the whole population of cards be placed in jeopardy.

In the current implementation, several methods of diversifying a key are supported: CLR8-ENC, TDES-ENC, TDES-DEC, SESS-XOR, TDES-XOR, TDESEMV2 and TDESEMV4. The first two methods triple-encrypt data using the generating_key to form the diversified key. The diversified key is then multiply-enciphered by the master key modified by the control vector for the output key. The TDES-DEC method is similar except that the data is triple-decrypted.

The SESS-XOR method provides a means for modifying an existing DATA, DATAC, MAC, DATAM, or MACVER, DATAMV single- or double-length key. The provided data is exclusive-ORed into the clear value of the key. This form of key diversification is specified by several of the credit card associations.

The TDES-ENC and TDES-DEC methods permit the production of either another key-generating key, or a final key. Control-vector bits 19 – 22 associated with the key-generating key specify the permissible type of final key. (See DKYGENKY in Figure 12 on page 638.) Control-vector bits 12 – 14 associated with the key-generating key specify if the diversified key is a final key or another in a series of key-generating keys. Bits 12 – 14 specify a counter that is decreased by one each time the diversified key generate service is used to produce another key-generating key. For example, if the key-generating key that you specify has this counter set to B'010', then you must specify the control vector for the generated_key with a DKYGENKY key type having the counter bits set to B'001' and specifying the same final key type in bits 19 – 22. Use of a generating_key with bits 12 – 14 set to B'000' results in the creation of the final key. Thus you can control both the number of diversifications required to reach a final key, and you can closely control the type of the final key.

The TDESEMV2, TDESEMV4, and TDES-XOR methods also derive a key by encrypting supplied data including a transaction counter value received from an EMV smart card. The processes are described in detail at Visa and EMV-related smart card formats and processes” on page 697. Refer to "Working with Europay–MasterCard–Visa smart cards” on page 302 to understand the various verbs you can use to operate with EMV smart cards.

Callable Services for Dynamic CKDS Update

ICSF provides the dynamic CKDS update services that allow applications to directly manipulate both the DASD copy and in-storage copy of the current CKDS.

Note: Applications using the dynamic CKDS update callable services can run concurrently with other operations that affect the CKDS, such as KGUP, CKDS conversion, REFRESH, and dynamic master key change. An operation can fail if it needs exclusive or shared access to the same DASD
copy of the CKDS that is held shared or exclusive by another operation. ICSF provides serialization to prevent data loss from attempts at concurrent access, but your installation is responsible for the effective management of concurrent use of competing operations. Consult your system administrator or system programmer for your installation guidelines.

The syntax of the key record create, key record read, and key record write services is identical with the same services provided by the Transaction Security System security application programming interface. Key management applications that use these common interface verbs can run on both systems without change.

**Key Record Create Callable Service (CSNBKRC)**

This service accepts a key label and creates a null key record in both the DASD copy and in-storage copy of the CKDS. The record contains a key token set to binary zeros and is identified by the key label passed in the call statement. The key label must be unique.

Prior to updating a key record using either the dynamic CKDS update services or KGUP, that record must already exist in the CKDS. You can use either the key record create service, KGUP, or your key entry hardware to create the initial record in the CKDS.

**Key Record Delete Callable Service (CSNBKRD)**

This service accepts a unique key label and deletes the associated key record from both the in-storage and DASD copies of the CKDS. This service deletes the entire record, including the key label from the CKDS.

**Key Record Read Callable Service (CSNBKRR)**

This service copies an internal key token from the in-storage CKDS to the application storage, where it may be used directly in other cryptographic services. Key labels specified with this service must be unique.

**Key Record Write Callable Service (CSNBKRW)**

This service accepts an internal key token and a label and writes the key token to the CKDS record identified by the key label. The key label must be unique. Application calls to this service write the key token to both the DASD copy and in-storage copy of the CKDS, so the record must already exist in both copies of the CKDS.

**Callable Services that Support Secure Sockets Layer (SSL)**

The Secure Sockets Layer (SSL) protocol, developed by Netscape Development Corporation, provides communications privacy over the Internet. Client/server applications can use the SSL protocol to provide secure communications and prevent eavesdropping, tampering, or message forgery.

ICSF provides callable services that support the RSA-encryption and RSA-decryption of PKCS 1.2-formatted symmetric key data to produce symmetric session keys. These session keys can then be used to establish an SSL session between the sender and receiver.

**PKA Decrypt Callable Service (CSNDPKD)**

The PKA decrypt callable service uses the corresponding private RSA key to unwrap the RSA-encrypted key and deformat the key value. This service then returns the clear key value to the application.
PKA Encrypt Callable Service (CSNDPKE)
The PKA encrypt callable service encrypts a supplied clear key value under an RSA public key. Currently, the supplied key can be formatted using the PKCS 1.2 or ZERO-PAD methods prior to encryption.

System Encryption Algorithm

Note: This topic only applies to systems with the Cryptographic Coprocessor Feature.

ICSF uses either the DES or AES algorithm or the Commercial Data Masking Facility (CDMF) to encipher and decipher data. The CDMF defines a scrambling technique for data confidentiality. It is a substitute for those customers prohibited from receiving IBM products that support DES data confidentiality services. The CDMF data confidentiality algorithm is composed of two processes: a key shortening process and a standard DES process to encipher and decipher data.

Your system can be one of these:
- DES
- CDMF
- DES-CDMF

A DES system protects data using a single-length, double-length, or triple-length DES data-encrypting key and the DES algorithm.

A CDMF system protects data using a single-length DES data-encrypting key and the CDMF. You input a standard single-length data-encrypting key to the encipher (CSNBENC) and decipher (CSNBDEC) callable services. The single-length data-encrypting key that is intended to be passed to the CDMF is called a CDMF key. Cryptographically, it is indistinguishable from a DES data-encrypting key. Prior to the key being used to encipher or decipher data, however, the Cryptographic Coprocessor Feature hardware cryptographically shortens the key of the CDMF process. This transformed, shortened data-encrypting key can be used only in the DES. (It must never be used in the CDMF; this would result in a double shortening of the key.) When used with the DES, a transformed, shortened data-encrypting key produces results identical to those that the CDMF would produce using the original single-length key.

A DES-CDMF system protects data using either the DES or the CDMF. The default is DES.

ICSF provides functions to mark internal IMPORTER, EXPORTER, and DATA key tokens with data encryption algorithm bits. IMPORTER and EXPORTER KEKs are marked when they are installed in operational form in ICSF. Your cryptographic key administrator does this. (See z/OS Cryptographic Services ICSF Administrator's Guide for details.) Whenever a DATA key is imported or generated in concert with a marked KEK, this marking is transferred to the DATA key token, unless the token copying function of the callable service is used to override the KEK marking with the marking of the key token passed. These data encryption algorithm bits internally drive the DES or CDMF for the ICSF encryption services. External key tokens are not marked with these data encryption algorithm bits.

IMPORTER and EXPORTER KEKs can have data encryption algorithm bit markings of CDMF (X'80'), DES (X'40'), or SYS-ENC (X'00'). DATA keys generated or imported with marked KEKs will also be marked. A CDMF-marked KEK will transfer a data encryption algorithm bit marking of CDMF (X'80') to the DATA key token. A
DES-marked KEK will transfer a data encryption algorithm bit marking of DES (X'00') to the DATA key token. A SYS-ENC-marked KEK will transfer a CDMF (X'80') marking to the DATA key token on a CDMF system, and a DES (X'00') marking to the DATA key token on DES-CDMF and DES systems.

To accomplish token copying of data encryption algorithm marks, a valid internal token of the same key type must be provided in the target key identifier field of the service. The token must have the proper token mark to be copied.

Notes:
1. For the multiple secure key import callable service the token markings on the KEK are ignored. In this case, the algorithm choice specified in the rule array determines the markings on the DATA key.
2. Propagation of data encryption algorithm bits and token copying are only performed when the ICSF callable service is performed on the Cryptographic Coprocessor Feature. The PCI Cryptographic Coprocessor, PCI X Cryptographic Coprocessor, Crypto Express2 Coprocessor, and Crypto Express3 Coprocessor do not perform these functions.

Table 4 summarizes the data encryption algorithm bits by key type, and the algorithm they drive in the ICSF encryption services.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Key Type</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDMF</td>
<td>DATA</td>
<td>X'80'</td>
</tr>
<tr>
<td></td>
<td>KEK</td>
<td>X'80'</td>
</tr>
<tr>
<td>DES</td>
<td>DATA</td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td>KEK</td>
<td>X'40'</td>
</tr>
<tr>
<td>System Default Algorithm</td>
<td>KEK</td>
<td>X'00'</td>
</tr>
</tbody>
</table>

For PCF users, your system programmer specifies a default encryption mode of DES or CDMF when installing ICSF. (See z/OS Cryptographic Services ICSF System Programmer's Guide for details.)

ANSI X9.17 Key Management Services

Restriction: ANSI X9.17 keys and ANSI key management services are only supported on the IBM @server zSeries 800 and the IBM @server zSeries 900.

The ANSI X9.17 key management standard defines a process for protecting and exchanging DES keys. The ANSI X9.17 standard defines methods for generating, exchanging, using, storing, and destroying these keys. ANSI X9.17 keys are protected by the processes of notarization and offsetting, instead of control vectors. In addition to providing services to support these processes, ICSF also defines and uses an optional process of partial notarization.

Offsetting involves exclusive-ORing a key-encrypting key with a counter. The counter, a 56-bit binary number that is associated with a key-encrypting key and contained in certain ANSI X9.17 messages, prevents either a replay or an out-of-sequence transmission of a message. When the associated AKEK is first used, the application initializes the counter. With each additional use, the application increments the counter.
Notarization associates the identities of a pair of communicating parties with a cryptographic key. The notarization process cryptographically combines a key with two 16-byte quantities, the origin identifier and the destination identifier, to produce a notarized key. The notarization process is completed by offsetting the AKEK with a counter.

ICSF makes it possible to divide the AKEK notarization process into two steps. In the first step, partial notarization, the AKEK is cryptographically combined with the origin and destination identifiers and returned in a form that can be stored in the CKDS or application storage. In the second step, the partially notarized AKEK is exclusive OR-ed with a binary counter to complete the notarization process. Partial notarization improves performance when you use an AKEK for many cryptographic service messages, each with a different counter. For details of the partial notarization calculations, refer to "ANSI X9.17 Partial Notarization Method" on page 693.

ICSF provides these callable services to support the ANSI X9.17 key management standard. Except where noted, these callable services have the identical syntax as the Transaction Security System verbs of the same name. With few exceptions, key management applications that use these common callable services, or verbs, can be executed on either system without change. Internal tokens cannot be interchanged; external tokens can be.

**Key Generate Callable Service Used to Generate an AKEK (CSNBKGN)**

The key generate callable service, described in "Key Generate Callable Service (CSNBKGN)" on page 25, can also be used to generate an AKEK in the operational form. It generates either an 8-byte or 16-byte AKEK and places it in a skeleton key token created by the key token build callable service. The length of the AKEK is determined by the key length keyword specified when building the key token.

**ANSI X9.17 EDC Generate Callable Service (CSNAEGN)**

This service generates an ANSI X9.17 error detection code on an arbitrary length string.

**ANSI X9.17 Key Export Callable Service (CSNAKEX)**

This service uses the ANSI X9.17 protocol to export a DATA key or a pair of DATA keys, with or without an AKEK. It also provides the ability to convert a single supplied DATA key or combine two supplied DATA keys into a MAC key.

**ANSI X9.17 Key Import Callable Service (CSNAKIM)**

This service uses the ANSI X9.17 protocol to import a DATA key or a pair of DATA keys, with or without an AKEK. It also provides the ability to convert a single supplied DATA key or combine two supplied DATA keys into a MAC key. The syntax is identical to the Transaction Security System verb, with these exceptions:

- Keys cannot be imported directly into the CKDS.

**ANSI X9.17 Key Translate Callable Service (CSNAKTR)**

This service translates one or two DATA keys or an AKEK from encryption under one AKEK to encryption under another AKEK, using the ANSI X9.17 protocol.

**ANSI X9.17 Transport Key Partial Notarize Callable Service (CSNATKN)**

This service preprocesses or partially notarizes an AKEK with origin and destination identifiers. The partially notarized key is supplied to the ANSI X9.17 key export,
ANSI X9.17 key import, or ANSI X9.17 key translate callable service to complete the notarization process. The syntax is identical to the Transaction Security System verb except that:

- The callable service does not update the CKDS.

**Enciphering and Deciphering Data**

The encipher and decipher callable services protect data off the host. ICSF protects sensitive data from disclosure to people who do not have authority to access it. Using algorithms that make it difficult and expensive for an unauthorized user to derive the original clear data within a practical time period assures privacy.

To protect data, ICSF can use the Data Encryption Standard (DES) algorithm to encipher or decipher data or keys. The algorithm is documented in the *Federal Information Processing Standard #46*. You can use the encipher and decipher callable services to encipher and decipher data with encrypted keys. On CCF systems, ICSF also supports the CDMF encryption mode. See "System Encryption Algorithm" on page 45 for more information.

The Symmetric Key Encipher and Symmetric Key Decipher callable services are used to encipher and decipher data in an address space or a data space using the cipher block chaining and electronic code book modes. The Advanced Encryption Standard (AES) and DES (Data Encryption Standard) are supported. AES encryption uses a 128-, 192- or 256-bit key. Only clear keys will be supported. The AES encryption is subject to the same availability restrictions as triple-DES encryption.

To protect data, ICSF can use the Advanced Encryption Standard (AES) algorithm to encipher or decipher data or keys. The algorithm is documented in the Federal Information Processing Standard #192. You can use the symmetric algorithm encipher and symmetric algorithm decipher callable services to encipher and decipher data with encrypted AES keys.

**Encoding and Decoding Data (CSNBECO and CSNBDCO)**

The encode and decode callable services perform functions with clear keys. Encode enciphers 8 bytes of data using the electronic code book (ECB) mode of the DES and a clear key. Decode does the inverse of the encode service. These services are available only on a DES-capable system. (See "System Encryption Algorithm" on page 45 for more information.)

**Translating Ciphertext (CSNBCTT and CSNBCTT1)**

**Restriction:** These services are only available on the IBM zSeries 800 and the IBM zSeries 900.

ICSF also provides a ciphertext translate callable service. It deciphers encrypted data (ciphertext) under one encryption key and reenciphers it under another key without having the data appear in the clear outside the cryptographic feature. Such a function is useful in a multiple node network, where sensitive data is passed through multiple nodes prior to it reaching its final destination. Different nodes use different keys in the process. For more information about different nodes, see "Using the Ciphertext Translate Callable Service" on page 60.
Managing Data Integrity and Message Authentication

To ensure the integrity of transmitted messages and stored data, ICSF provides:

- Message authentication code (MAC)
- Several hashing functions, including modification detection code (MDC), SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, RIPEMD-160 and MD5

(See Chapter 9, "Using Digital Signatures," on page 379 for an alternate method of message authentication using digital signatures.)

The choice of callable service depends on the security requirements of the environment in which you are operating. If you need to ensure the authenticity of the sender and also the integrity of the data, consider message authentication code processing. If you need to ensure the integrity of transmitted data in an environment where it is not possible for the sender and the receiver to share a secret cryptographic key, consider hashing functions, such as the modification detection code process.

Message Authentication Code Processing

The process of verifying the integrity and authenticity of transmitted messages is called message authentication. Message authentication code (MAC) processing allows you to verify that a message was not altered or a message was not fraudulently introduced onto the system. You can check that a message you have received is the same one sent by the message originator. The message itself may be in clear or encrypted form. The comparison is performed within the cryptographic feature. Since both the sender and receiver share a secret cryptographic key used in the MAC calculation, the MAC comparison also ensures the authenticity of the message.

In a similar manner, MACs can be used to ensure the integrity of data stored on the system or on removable media, such as tape.

ICSF provides support for both single-length and double-length MAC generation and MAC verification keys. With the ANSI X9.9-1 single key algorithm, use the single-length MAC and MACVER keys.

ICSF provides support for the use of data-encrypting keys in the MAC generation and verification callable services, and also the use of a MAC generation key in the MAC verification callable service. This support permits ICSF MAC services to interface more smoothly with non-CCA key distribution system, including those implementing the ANSI X9.17 protocol.

MAC Generation Callable Service (CSNBMGN or CSNBMGN1)

When a message is sent, an application program can generate an authentication code for it using the MAC generation callable service. The callable service computes the message authentication code using one of these methods:

- Using the ANSI X9.9-1 single key algorithm, a single-length MAC generation key or data-encrypting key, and the message text.
- Using the ANSI X9.19 optional double key algorithm, a double-length MAC generation key and the message text.
- Using Europay, MasterCard and Visa (EMV) padding rules with a single-length MAC key or double-length MAC key and the message text.
Using ISO 16609 algorithm with a double-length MAC or a double-length DATA key and the message text.

ICSF allows a MAC to be the leftmost 32 or 48 bits of the last block of the ciphertext or the entire last block (64 bits) of the ciphertext. The originator of the message sends the message authentication code with the message text.

**MAC Verification Callable Service (CSNBMVR or CSNBMVR1)**

When the receiver gets the message, an application program calls the MAC verification callable service. The callable service verifies a MAC by generating another MAC and comparing it with the MAC received with the message. If the two codes are the same, the message sent was the same one received. A return code indicates whether the MACs are the same.

The MAC verification callable service can use either of these methods to generate the MAC for authentication:

- The ANSI X9.9-1 single key algorithm, a single-length MAC verification or MAC generation key (or a data-encrypting key), and the message text.
- The ANSI X9.19 optional double key algorithm, a double-length MAC verification or MAC generation key and the message text.
- Using Europay, MasterCard and Visa (EMV) padding rules with a single-length MAC key or double-length MAC key and the message text.
- Using ISO 16609 algorithm with a double-length MAC or a double-length DATA key and the message text.

The method used to verify the MAC should correspond with the method used to generate the MAC.

ICSF provides support for MAC using clear AES keys with the symmetric MAC generate (CSNBMSG, CSNBMSG1, CSNESMG and CSNESMG1) and symmetric MAC verify (CSNBMSV, CSNBMSV1, CSNESMV and CSNESMV1). These services support the CBC-MAC and XCBC-MAC (AES-XCBC-MAC-96 and AES-XCBC-PRF-128) algorithms.

**Hashing Functions**

Hashing functions include one-way hash generation and modification detection code (MDC) processing.

**One-Way Hash Generate Callable Service (CSNBOWH and CSNBOWH1)**

This service hashes a supplied message. Supported hashing methods include:

- SHA-1
- SHA-224
- SHA-256
- SHA-384
- SHA-512
- RIPEMD-160
- MD5

2. The Secure Hash Algorithm (SHA) is also called the Secure Hash Standard (SHS), which Federal Information Processing Standard (FIPS) Publication 180 defines.
**MDC Generation Callable Service (CSNBMDG and CSNBMDG1)**

The modification detection code (MDC) provides a form of support for data integrity. The MDC allows you to verify that data was not altered during transmission or while in storage. The originator of the data ensures that the MDC is transmitted with integrity to the intended receiver of the data. For instance, the MDC could be published in a reliable source of public information. When the receiver gets the data, an application program can generate an MDC, and compare it with the original MDC value. If the MDC values are equal, the data is accepted as unaltered. If the MDC values differ the data is assumed to be bogus.

Supported hashing methods through the MDC generation callable service are:
- MDC-2
- MDC-4
- PADMDC-2
- PADMDC-4

In a similar manner, MDCs can be used to ensure the integrity of data stored on the system or on removable media, such as tape.

When data is sent, an application program can generate a modification detection code for it using the MDC generation callable service. The callable service computes the modification detection code by encrypting the data using a publicly-known cryptographic one-way function. The MDC is a 128-bit value that is easy to compute for specific data, yet it is hard to find data that will result in a given MDC.

Once an MDC has been established for a file, the MDC generation service can be run at any other time on the file. The resulting MDC can then be compared with the previously established MDC to detect deliberate or inadvertent modification.

---

**Managing Personal Authentication**

The process of validating personal identities in a financial transaction system is called *personal authentication*. The personal identification number (PIN) is the basis for verifying the identity of a customer across the financial industry networks. ICSF checks a customer-supplied PIN by verifying it using an algorithm. The financial industry needs functions to generate, translate, and verify PINs. These functions prevent unauthorized disclosures when organizations handle personal identification numbers.

ICSF supports these algorithms for generating and verifying personal identification numbers:
- IBM 3624
- IBM 3624 PIN offset
- IBM German Bank Pool
- IBM German Bank Pool PIN Offset (GBP-PINO)
- VISA PIN validation value
- Interbank

With ICSF, you can translate PIN blocks from one format to another. ICSF supports these formats:
- ANSI X9.8
- ISO formats 0, 1, 2, 3
- VISA formats 1, 2, 3, 4
- IBM 4704 Encrypting PINPAD format
- IBM 3624 formats
• IBM 3621 formats
• ECI formats 1, 2, 3

With the capability to translate personal identification numbers into different PIN block formats, you can use personal identification numbers on different systems.

Verifying Credit Card Data
The Visa International Service Association (VISA) and MasterCard International, Incorporated have specified a cryptographic method to calculate a value that relates to the personal account number (PAN), the card expiration date, and the service code. The VISA card-verification value (CVV) and the MasterCard card-verification code (CVC) can be encoded on either track 1 or track 2 of a magnetic striped card and are used to detect forged cards. Because most online transactions use track-2, the ICSF callable services generate and verify the CVV3 by the track-2 method.

The VISA CVV generate callable service calculates a 1- to 5-byte value through the DES-encryption of the PAN, the card expiration date, and the service code using two data-encrypting keys or two MAC keys. The VISA CVV service verify callable service calculates the CVV by the same method, compares it to the CVV supplied by the application (which reads the credit card’s magnetic stripe) in the CVV_value, and issues a return code that indicates whether the card is authentic.

Clear PIN Encrypt Callable Service (CSNBCPE)
To format a PIN into a PIN block format and encrypt the results, use the Clear PIN Encrypt callable service. You can also use this service to create an encrypted PIN block for transmission. With the RANDOM keyword, you can have the service generate random PIN numbers. Use of this service requires the optional PCIXCC, CEX2C, or CEX3C. An enhanced PIN security mode, on PCICC, PCIXCC, CEX2C, and CEX3C, is available for formatting an encrypted PIN block into IBM 3621 format or IBM 3624 format. See "Clear PIN Encrypt (CSNBCPE)" on page 309 for more information.

Clear PIN Generate Alternate Callable Service (CSNBCPA)
To generate a clear VISA PIN validation value from an encrypted PIN block, call the clear PIN generate alternate callable service. This service also supports the IBM-PINO algorithm to produce a 3624 offset from a customer selected encrypted PIN.

An enhanced PIN security mode is available for extracting PINs from encrypted PIN blocks. This mode only applies on PCICC, PCIXCC, CEX2C, or CEX3C, when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. See "Clear PIN Generate Alternate (CSNBCPA)" on page 317 for more information.

Note: The PIN block must be encrypted under either an input PIN-encrypting key (IPINENC) or output PIN-encrypting key (OPINENC). Using an IPINENC key requires NOCV keys to be enabled in the CKDS. Functions other than VISA PIN validation value generation require the optional PCICC, PCIXCC, CEX2C, or CEX3C.

3. The VISA CVV and the MasterCard CVC refer to the same value. CVV is used here to mean both CVV and CVC.
Clear PIN Generate Callable Service (CSNBPGN)

To generate personal identification numbers, call the Clear PIN generate callable service. Using a PIN generation algorithm, data used in the algorithm, and the PIN generation key, the callable service generates a clear PIN, a PIN verification value, or an offset. The callable service can only execute in special secure mode, which is described in “Special Secure Mode” on page 10.

Encrypted PIN Generate Callable Service (CSBEPG)

To generate personal identification numbers, call the Encrypted PIN generation callable service. Using a PIN generation algorithm, data used in the algorithm, and the PIN generation key, the callable service generates a PIN and using a PIN block format and the PIN encrypting key, formats and encrypts the PIN. Use of this service requires the optional PCICC, PCIXCC, CEX2C, or CEX3C. An enhanced PIN security mode, on PCICC, PCIXCC, CEX2C, and CEX3C, is available for formatting an encrypted PIN block into IBM 3621 format or IBM 3624 format. See “Encrypted PIN Generate (CSBEPG)” on page 323 for more information.

Encrypted PIN Translate Callable Service (CSNBPTR)

To translate a PIN from one PIN-encrypting key to another or from one PIN block format to another or both, call the Encrypted PIN translation callable service. You must identify the input PIN-encrypting key that originally enciphers the PIN. You also need to specify the output PIN-encrypting key that you want the callable service to use to encipher the PIN. If you want to change the PIN block format, specify a different output PIN block format from the input PIN block format. An enhanced PIN security mode, on PCICC, PCIXCC, CEX2C, and CEX3C, is available for formatting an encrypted PIN block into IBM 3621 format or IBM 3624 format. An enhanced security mode is also available for extracting PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. See “Encrypted PIN Translate (CSNBPTR)” on page 328 for more information.

Encrypted PIN Verify Callable Service (CSNBPVR)

To verify a supplied PIN, call the Encrypted PIN verify callable service. You need to specify the supplied enciphered PIN, the PIN-encrypting key that enciphers it, and other relevant data. You must also specify the PIN verification key and PIN verification algorithm. It compares the two personal identification numbers; if they are the same, it verifies the supplied PIN. See Chapter 8, “Financial Services,” on page 301 for additional information.

PIN Change/Unblock Callable Service (CSNBPCU)

To support PIN change algorithms specified in the VISA Integrated Circuit Card Specification, call the PIN change/unblock callable service. The service can be executed on z890/z990 and later machines.

An enhanced PIN security mode, on PCICC, PCIXCC, CEX2C, and CEX3C, is available for extracting PINs from encrypted PIN blocks. This mode only applies
when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. See “PIN Change/Unblock (CSNBPCU)” on page 341 for more information.

Transaction Validation Callable Service (CSNBTRV)

To support generation and validation of American Express card security codes, call the transaction validation callable service. The service can be executed on z890/z990 and later machines.

Secure Messaging

These services will assist applications in encrypting secret information such as clear keys and PIN blocks in a secure message. These services will execute within the secure boundary of the PCICC, PCIXCC, CEX2C, or CEX3C.

The Secure Messaging for Keys (CSNBSKY) callable service encrypts a text block, including a clear key value decrypted from an internal or external DES token.

The Secure Messaging for PINs (CSNBSPN) callable service encrypts a text block, including a clear PIN block recovered from an encrypted PIN block.

Trusted Key Entry (TKE) Support

The Trusted Key Entry (TKE) workstation is an optional feature. It offers an alternative to clear key entry. You can use the TKE workstation to load:

- DES master key, AES master key, PKA master keys, and operational keys in a secure way. CCF only supports Operational Transport and PIN keys. On the PCIXCC/CEX2C, all operational keys may be loaded with TKE V4.1 or higher.
- AES master key and AES operational keys may be loaded with TKE V5.3. On the CEX3C, all operational keys may be loaded with TKE 6.0 or later.
- DES-MK and ASYM-MK master keys on the PCICC, PCIXCC, CEX2C, or CEX3C.
- AES master keys are only on z9 and z10 systems running with the Nov. 2008 or later licensed internal code (LIC).

You can load keys remotely and for multiple PCICCs, PCIXCCs, CEX2Cs, or CEX3Cs. The TKE workstation eases the administration for using one Cryptographic Coprocessor Feature or PCIXCC/CEX2C/CEX3C as a production machine and as a test machine at the same time, while maintaining security and reliability.

The TKE workstation can be used for enabling/disabling access control points for callable services executed on PCICCs, PCIXCCs, CEX2Cs, and CEX3Cs. See Appendix H, “Access Control Points and Callable Services,” on page 703 for additional information.

For complete details about the TKE workstation see z/OS Cryptographic Services ICSF TKE PCIX Workstation User’s Guide.

TKE Version 4.0 or higher is required if using a PCIXCC/CEX2C.

TKE Version 6.0 or higher is required is using a CEX3C.
On z890, z990 z9 EC, z9 BC, z10 EC and z10 BC systems running with May 2004 or higher version of Licensed Internal Code or an z9 EC, z9 BC, z10 EC and z10 BC with MCL 029 Stream J12220 or higher of Licensed Internal Code, you must enable TKE commands for each PCIXCC/CEX2C/CEX3C card from the Support Element. This is true for new TKE users and those upgrading from TKE V4.0 to V4.1, V4.2 or V5.x when the new LIC is installed. See Support Element Operations Guide and z/OS Cryptographic Services ICSF TKE PCIX Workstation User’s Guide, SA23-2211 for more information.

Utilities

ICSF provides these utilities.

Character/Nibble Conversion Callable Services (CSNBXBC and CSNBXCB)

The character/nibble conversion callable services are utilities that convert a binary string to a character string and vice versa.

Code Conversion Callable Services (CSNBXEA and CSNBXAE)

The code conversion callable services are utilities that convert EBCDIC data to ASCII data and vice versa.

X9.9 Data Editing Callable Service (CSNB9ED)

The data editing callable service is a utility that edits an ASCII text string according to the editing rules of ANSI X9.9-4.

ICSF Query Algorithm Callable Service (CSFIQA)

The callable service provides information regarding the cryptographic and hash algorithms available.

ICSF Query Facility Callable Service (CSFIQF)

The callable service provides ICSF status information, as well as PCICC, PCIXCC, CEX2C, and CEX3C information.

Typical Sequences of ICSF Callable Services

Sample sequences in which the ICSF callable services might be called are shown in Table 5 on page 56.
### Table 5. Combinations of the Callable Services

<table>
<thead>
<tr>
<th>Combination A (DATA keys only)</th>
<th>Combination B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Random number generate</td>
<td>1. Random number generate</td>
</tr>
<tr>
<td>2. Clear key import or multiple clear key import</td>
<td>2. Secure key import or multiple secure key import</td>
</tr>
<tr>
<td>3. Encipher/decipher</td>
<td>3. Any service</td>
</tr>
<tr>
<td>4. Data key export or key export (optional step)</td>
<td>4. Data key export for DATA keys, or key export in the general case (optional step)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Combination C</th>
<th>Combination D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Key generate (OP form only)</td>
<td>1. Key generate (OPEX form)</td>
</tr>
<tr>
<td>2. Any service</td>
<td>2. Any service</td>
</tr>
<tr>
<td>3. Key export (optional)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Combination E</th>
<th>Combination F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Key generate (IM form only)</td>
<td>1. Key generate (IMEX form)</td>
</tr>
<tr>
<td>2. Key import</td>
<td>2. Key import</td>
</tr>
<tr>
<td>3. Any service</td>
<td>3. Any service</td>
</tr>
<tr>
<td>4. Key export (optional)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Combination G</th>
<th>Combination H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Key generate</td>
<td>1. Key import</td>
</tr>
<tr>
<td>2. Key record create</td>
<td>2. Key record create</td>
</tr>
<tr>
<td>3. Key record write</td>
<td>3. Key record write</td>
</tr>
<tr>
<td>4. Any service (passing label of the key just generated)</td>
<td>4. Any service (passing label of the key just generated)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Combination I</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Key token build to create key token skeleton</td>
</tr>
<tr>
<td>2. Key generate to OP form of AKEK using key token skeleton</td>
</tr>
<tr>
<td>3. Use AKEK in any ANSI X9.17 service</td>
</tr>
</tbody>
</table>

### Notes:

1. An example of "any service" is CSNBENC.
2. These combinations exclude services that can be used on their own; for example, key export or encode, or using key generate to generate an exportable key.
3. These combinations do not show key communication, or the transmission of any output from an ICSF callable service.
4. Combination I is not available on the IBM eServer zSeries 990.

The key forms are described in "Key Generate (CSNBKGN and CSNEKGN)" on page 111.

---

### Key Forms and Types Used in the Key Generate Callable Service

The key generate callable service is the most complex of all the ICSF callable services. This topic provides examples of the key forms and key types used in the key generate callable service.

### Generating an Operational Key

To generate an operational key, choose one of these methods:
For operational keys, call the key generate callable service (CSNBKGN). Table 26 on page 120 and Table 27 on page 120 show the key type and key form combinations for a single key and for a key pair.

For operational keys, call the random number generate callable service (CSNBRNG) and specify the form parameter as RANDOM. Specify ODD parity for a random number you intend to use as a key. Then pass the generated value to the secure key import callable service (CSNBSKI) with a required key type. The required key type is now in operational form.

This method requires a cryptographic unit to be in special secure mode. For more information about special secure mode, see “Special Secure Mode” on page 10.

For data-encrypting keys, call the random number generate callable service (CSNBRNG) and specify the form parameter as ODD. Then pass the generated value to the clear key import callable service (CSNBCKI) or the multiple clear key import callable service (CSNBCKM). The DATA key type is now in operational form.

You cannot generate a PIN verification (PINVER) key in operational form because the originator of the PIN generation (PINGEN) key generates the PINVER key in exportable form, which is sent to you to be imported.

Generating an Importable Key
To generate an importable key form, call the key generate callable service (CSNBKGN).

If you want a DATA, MAC, PINGEN, DATAM, or DATAC key type in importable form, obtain it directly by generating a single key. If you want any other key type in importable form, request a key pair where either the first or second key type is importable (IM). Discard the generated key form that you do not need.

Generating an Exportable Key
To generate an exportable key form, call the key generate callable service (CSNBKGN).

If you want a DATA, MAC, PINGEN, DATAM, or DATAC key type in exportable form, obtain it directly by generating a single key. If you want any other key type in exportable form, request a key pair where either the first or second key type is exportable (EX). Discard the generated key form that you do not need.

Examples of Single-Length Keys in One Form Only

<table>
<thead>
<tr>
<th>Key Form</th>
<th>Key Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP DATA</td>
<td>Encipher or decipher data. Use data key export or key export to send encrypted key to another cryptographic partner. Then communicate the ciphertext.</td>
<td></td>
</tr>
<tr>
<td>OP MAC</td>
<td>MAC generate. Because no MACVER key exists, there is no secure communication of the MAC with another cryptographic partner.</td>
<td></td>
</tr>
<tr>
<td>IM DATA</td>
<td>Key Import, and then encipher or decipher. Then key export to communicate ciphertext and key with another cryptographic partner.</td>
<td></td>
</tr>
<tr>
<td>EX DATA</td>
<td>You can send this key to a cryptographic partner, but you</td>
<td></td>
</tr>
</tbody>
</table>
can do nothing with it directly. Use it for the key
distribution service. The partner could then use key import
to get it in operational form, and use it as in OP DATA
above.

Examples of OPIM Single-Length, Double-Length, and Triple-Length
Keys in Two Forms

The first two letters of the key form indicate the form that key type 1 parameter is
in, and the second two letters indicate the form that key type 2 parameter is in.

<table>
<thead>
<tr>
<th>Key Form</th>
<th>Type 1</th>
<th>Type 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPIM</td>
<td>DATA</td>
<td>DATA</td>
</tr>
</tbody>
</table>
|          | Use the OP form in encipher. Use key export with the
|          | OP form to communicate ciphertext and key with
|          | another cryptographic partner. Use key import at a
|          | later time to use encipher or decipher with the same
|          | key again. |
| OPIM     | MAC    | MAC    |
|          | Single-length MAC generation key. Use the OP form in
|          | MAC generation. You have no corresponding MACVER key,
|          | but you can call the MAC verification service with
|          | the MAC key directly. Use the key import callable
|          | service and then compute the MAC again using the MAC
|          | verification callable service, which compares the MAC
|          | it generates with the MAC supplied with the message
|          | and issues a return code indicating whether they
|          | compare. |

Examples of OPEX Single-Length, Double-Length, and Triple-Length
Keys in Two Forms

<table>
<thead>
<tr>
<th>Key Form</th>
<th>Type 1</th>
<th>Type 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPEX</td>
<td>DATA</td>
<td>DATA</td>
</tr>
</tbody>
</table>
|          | Use the OP form in encipher. Send the EX form and
|          | the ciphertext to another cryptographic partner. |
| OPEX     | MAC    | MAC    |
|          | Single-length MAC generation key. Use the OP form in
|          | both MAC generation and MAC verification. Send the
|          | EX form to a cryptographic partner to be used in the
|          | MAC generation or MAC verification services. |
| OPEX     | MAC    | MACVER |
|          | Single-length MAC generation and MAC verification
|          | keys. Use the OP form in MAC generation. Send the EX
|          | form to a cryptographic partner where it will be put
|          | into key import, and then MAC verification, with the
|          | message and MAC that you have also transmitted. |
| OPEX     | PINGEN| PINVER |
|          | Use the OP form in Clear PIN generate. Send the
|          | EX form to a cryptographic partner where it is put
|          | into key import, and then Encrypted PIN verify,
|          | along with an IPINENC key. |
| OPEX     | IMPORTER | EXPORTER |
|          | Use the OP form in key import, key generate,
|          | or secure key import. Send the EX form to a
|          | cryptographic partner where it is used in key
|          | export, data key export, or key generate, or put in
|          | the CKDS. |
| OPEX     | EXPORTER | IMPORTER |
|          | Use the OP form in key export, data key export,
|          | or key generate. Send the EX form to a cryptographic
|          | partner where it is put into the CKDS or used in key
|          | import, key generate or secure key import. |
When you and your partner have the OPEX IMPORTER EXPORTER, OPEX EXPORTER IMPORTER pairs of keys in "Examples of OPEX Single-Length, Double-Length, and Triple-Length Keys in Two Forms" on page 58 installed, you can start key and data exchange.

Examples of IMEX Single-Length and Double-Length Keys in Two Forms

<table>
<thead>
<tr>
<th>Key Form</th>
<th>Type 1</th>
<th>Type 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMEX</td>
<td>DATA</td>
<td>DATA</td>
</tr>
<tr>
<td>IMEX</td>
<td>MAC</td>
<td>MACVER</td>
</tr>
<tr>
<td>IMEX IMPORTER</td>
<td>EXPORTER</td>
<td></td>
</tr>
<tr>
<td>IMEX</td>
<td>PINGEN</td>
<td>PINVER</td>
</tr>
</tbody>
</table>

- **IMEX DATA DATA**: Use the key import callable service to import IM form and use the OP form in encipher. Send the EX form to a cryptographic partner.
- **IMEX MAC MACVER**: Use the key import callable service to import IM form and use the OP form in MAC generate. Send the EX form to a cryptographic partner who can verify the MAC.
- **IMEX IMPORTER EXPORTER**: Use the key import callable service to import IM form and send the EX form to a cryptographic partner. This establishes a new IMPORTER/EXPORTER key between you and your partner.
- **IMEX PINGEN PINVER**: Use the key import callable service to import IM form and send the EX form to a cryptographic partner. This establishes a new PINGEN/PINVER key between you and your partner.

Examples of EXEX Single-Length and Double-Length Keys in Two Forms

For the keys shown in this list, you are providing key distribution services for other nodes in your network, or other cryptographic partners. Neither key type can be used in your installation.

<table>
<thead>
<tr>
<th>Key Form</th>
<th>Type 1</th>
<th>Type 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXEX</td>
<td>DATA</td>
<td>DATA</td>
</tr>
<tr>
<td>EXEX</td>
<td>MAC</td>
<td>MACVER</td>
</tr>
<tr>
<td>EXEX IMPORTER</td>
<td>EXPORTER</td>
<td></td>
</tr>
<tr>
<td>EXEX</td>
<td>OPINENC IPINENC</td>
<td></td>
</tr>
</tbody>
</table>

- **EXEX DATA DATA**: Send the first EX form to a cryptographic partner with the corresponding IMPORTER and send the second EX form to another cryptographic partner with the corresponding IMPORTER. This exchange establishes a key between two partners.

Generating AKEKs

**Restriction**: AKEKs are only supported on the IBM @server zSeries 800 and the IBM @server zSeries 900.

AKEKs are bidirectional and are OP-form-only keys that can be used in both import and export. Prior to using the key generate callable service to create an AKEK, you need to use the key token build callable service to create a key token for receiving the AKEK. The steps involved in this process are:

1. Use the key token build callable service with these parameter values:
   - **Parameter**: Key_type
   - **Value**: AKEK
   - **Parameter**: Rule_array
   - **Value**: INTERNAL NO-KEY {SINGLE or DOUBLE-O}

2. Use the key generate callable service with these parameter values:
Using the Ciphertext Translate Callable Service

Restriction: The ciphertext translate callable service does not work in CDMF-only systems (see “System Encryption Algorithm” on page 45). The ciphertext translate callable service does not work on the PCI X Cryptographic Coprocessor, Crypto Express2 Coprocessor, or Crypto Express3 Coprocessor.

This topic describes a scenario using the encipher, ciphertext translate, and decipher callable services with four network nodes: A, B, C, and D. You want to send data from your network node A to a destination node D. You cannot communicate directly with node D, and nodes B and C are situated between you. You do not want nodes B and C to decipher your data.

At node A, you use the encipher callable service (CSNBENC or CSNBENC1). Node D uses the decipher callable service (CSNBDEC or CSNBDEC1).

Node B and C will use the ciphertext translate callable service. Consider the keys that are needed to support this process:

1. At your node, generate one key in two forms: OPEX DATA DATAXLAT
2. Send the exportable DATAXLAT key to node B.
3. Node B and C need to share a DATAXLAT key, so generate a different key in two forms: EXEX DATAXLAT DATAXLAT.
4. Send the first exportable DATAXLAT key to node B.
5. Send the second exportable DATAXLAT key to node C.
6. Node C and node D need to share a DATAXLAT key and a DATA key. Node D can generate one key in two forms: OPEX DATA DATAXLAT.
7. Node D sends the exportable DATAXLAT key to node C.

The communication process is shown as:

<table>
<thead>
<tr>
<th>Node: A B C D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Callable Service: Encipher Ciphertext Translate Ciphertext Translate Decipher</td>
</tr>
<tr>
<td>Keys: DATA DATAXLAT DATAXLAT DATAXLAT DATAXLAT DATA</td>
</tr>
<tr>
<td>Key Pairs: ___ = ___</td>
</tr>
</tbody>
</table>

Therefore, you need three keys, each in two different forms. You can generate two of the keys at node A, and node D can generate the third key. Note that the key used in the decipher callable service at node D is not the same key used in the encipher callable service at node A.

Summary of Callable Services

Table 6 on page 61 lists the callable services described in this publication, and their corresponding verbs. The figure also references the topic that describes the callable service.
<table>
<thead>
<tr>
<th>Verb</th>
<th>Service Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNBCKI</td>
<td>Clear key import</td>
<td>Imports an 8-byte clear DATA key, enciphers it under the master key, and places the result into an internal key token. CSNBCKI converts the clear key into operational form as a DATA key.</td>
</tr>
<tr>
<td>CSNECKI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNBCVG</td>
<td>Control vector generate</td>
<td>Builds a control vector from keywords specified by the key_type and rule_array parameters.</td>
</tr>
<tr>
<td>CSNBCVT</td>
<td>Control vector translate</td>
<td>Changes the control vector used to encipher an external key.</td>
</tr>
<tr>
<td>CSNBCVE</td>
<td>Cryptographic variable encipher</td>
<td>Uses a CVARENC key to encrypt plaintext by using the Cipher Block Chaining (CBC) method. The plaintext must be a multiple of eight bytes in length.</td>
</tr>
<tr>
<td>CSNBDKX</td>
<td>Data key export</td>
<td>Converts a DATA key from operational form into exportable form.</td>
</tr>
<tr>
<td>CSNBDKM</td>
<td>Data key import</td>
<td>Imports an encrypted source DES single- or double-length DATA key and creates or updates a target internal key token with the master key enciphered source key.</td>
</tr>
<tr>
<td>CSNBDKG</td>
<td>Diversified key generate</td>
<td>Generates a key based upon the key-generating key, the processing method, and the parameter data that is supplied.</td>
</tr>
<tr>
<td>CSNBKEX</td>
<td>Key export</td>
<td>Converts any key from operational form into exportable form. (However, this service does not export a key that was marked non-exportable when it was imported.)</td>
</tr>
<tr>
<td>CSNBKGN</td>
<td>Key generate</td>
<td>Generates a 64-bit, 128-bit, or 192-bit odd parity key, or a pair of keys; and returns them in encrypted forms (operational, exportable, or importable). CSNBKGN does not produce keys in plaintext.</td>
</tr>
<tr>
<td>CSNEKGN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNBKIM</td>
<td>Key import</td>
<td>Converts any key from importable form into operational form.</td>
</tr>
<tr>
<td>CSNBKPI</td>
<td>Key part import</td>
<td>Combines the clear key parts of any key type and returns the combined key value in an internal key token or an update to the CKDS.</td>
</tr>
<tr>
<td>CSNBKRC</td>
<td>Key record create</td>
<td>Adds a key record containing a key token set to binary zeros to both the in-storage and DASD copies of the CKDS.</td>
</tr>
<tr>
<td>CSNBKRD</td>
<td>Key record delete</td>
<td>Deletes a key record from both the in-storage and DASD copies of the CKDS.</td>
</tr>
<tr>
<td>CSNBKRR</td>
<td>Key record read</td>
<td>Copies an internal key token from the in-storage copy of the CKDS to application storage.</td>
</tr>
<tr>
<td>CSNBKRW</td>
<td>Key record write</td>
<td>Writes an internal key token to the CKDS record specified in the key label parameter. Updates both the in-storage and DASD copies of the CKDS currently in use.</td>
</tr>
</tbody>
</table>
Table 6. Summary of ICSF Callable Services (continued)

<table>
<thead>
<tr>
<th>Verb</th>
<th>Service Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNBKYT or CSNBK渝TX</td>
<td>Key test service</td>
<td>Generates or verifies (depending on keywords in the rule array) a secure verification pattern for keys. CSNBKYT requires the tested key to be in the clear or encrypted under the master key. CSNBK渝TX also allows the tested key to be encrypted under a key-encrypting key.</td>
</tr>
<tr>
<td>CSNBKTB</td>
<td>Key token build</td>
<td>Builds an internal or external token from the supplied parameters. You can use this callable service to build an internal token for an AKEK for input to the key generate and key part import callable services. You can also use this service to build CCA key tokens for all key types ICSF supports. You can also use this service to build CCA key tokens for all key types ICSF supports.</td>
</tr>
<tr>
<td>CSNBKTR</td>
<td>Key translate</td>
<td>Uses one key-encrypting key to decipher an input key and then enciphers this key using another key-encrypting key within the secure environment.</td>
</tr>
<tr>
<td>CSNBCKM</td>
<td>Multiple clear key import</td>
<td>Imports a single-, double-, or triple-length clear DATA key, enciphers it under the master key, and places the result into an internal key token. CSNBCKM converts the clear key into operational form as a DATA key.</td>
</tr>
<tr>
<td>CSNBSKM</td>
<td>Multiple secure key import</td>
<td>Enciphers a single-, double-, or triple-length clear key under the master key or an input importer key, and places the result into an internal or external key token as any key type. Triple-length keys can only be imported as DATA keys. CSNBSKM executes only in special secure mode.</td>
</tr>
<tr>
<td>CSNDPKD</td>
<td>PKA decrypt</td>
<td>Uses an RSA private key to decrypt the RSA-encrypted key value and return the clear key value to the application.</td>
</tr>
<tr>
<td>CSNFPKD</td>
<td>PKA encrypt</td>
<td>Encrypts a supplied clear key value under an RSA public key.</td>
</tr>
<tr>
<td>CSNBPEX</td>
<td>Prohibit export</td>
<td>Modifies an operational key so that it cannot be exported.</td>
</tr>
<tr>
<td>CSNPBEXX</td>
<td>Prohibit export extended</td>
<td>Changes the external token of a key in exportable form so that it can be imported at the receiver node but not exported from that node.</td>
</tr>
<tr>
<td>CSNBRNG</td>
<td>Random number generate</td>
<td>Generates an 8-byte random number or a random number with a user-specified length. The output can be specified in three forms of parity: RANDOM, ODD, and EVEN.</td>
</tr>
<tr>
<td>Verb</td>
<td>Service Name</td>
<td>Function</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CSNDRKX</td>
<td>Remote key export</td>
<td>Generates or exports DES keys for local use and for distribution to an ATM or other remote device. RKX uses a special structure to hold encrypted symmetric keys in a way that binds them to the trusted block and allows sequences of RKX calls to be bound together as if they were an atomic operation.</td>
</tr>
<tr>
<td>CSNBSKI</td>
<td>Secure key import</td>
<td>Enciphers a clear key under the master key, and places the result into an internal or external key token as any key type. CSNBSKI executes only in special secure mode.</td>
</tr>
<tr>
<td>CSNDXYX</td>
<td>Symmetric key export</td>
<td>Transfers an application-supplied symmetric key (a DATA key) from encryption under the DES host master key to encryption under an application-supplied RSA public key. The application-supplied DATA key must be an ICSF DES internal key token or the label of such a token in the CKDS.</td>
</tr>
<tr>
<td>CSNDXYI</td>
<td>Symmetric key import</td>
<td>Imports a symmetric DATA key enciphered under an RSA public key into operational form enciphered under a DES master key.</td>
</tr>
<tr>
<td>CSNDSYG</td>
<td>Symmetric key generate</td>
<td>Generates a symmetric DATA key and returns the key in two forms: enciphered under the DES master key or KEK and under a PKA public key.</td>
</tr>
<tr>
<td>CSNDSYI</td>
<td>Symmetric key import</td>
<td>Imports a symmetric DATA key enciphered under an RSA public key into operational form enciphered under a DES master key.</td>
</tr>
<tr>
<td>CSNBTCY</td>
<td>Transform CDMF key</td>
<td>Changes a CDMF DATA key in an internal or external token to a transformed shortened DES key.</td>
</tr>
<tr>
<td>CSNDTBC</td>
<td>Trusted block create</td>
<td>Creates a trusted block in a two step process. The block will be in external form, encrypted under an IMP-PKA transport key. This means that the MAC key contained within the trusted block will be encrypted under the IMP-PKA key.</td>
</tr>
<tr>
<td>CSFUDK</td>
<td>User Derived Key</td>
<td>Generates single-length or double-length MAC keys, or updates an existing user derived key.</td>
</tr>
<tr>
<td>CSNBCTT or CSNBCTT1</td>
<td>Ciphertext translate</td>
<td>Translates the user-supplied ciphertext from one key and enciphers the ciphertext to another key. (This is for DES encryption only.) CSNBCTT requires the ciphertext to reside in the caller’s primary address space. CSNBCTT1 allows the ciphertext to reside in the caller’s primary address space or in a z/OS data space.</td>
</tr>
</tbody>
</table>
Table 6. Summary of ICSF Callable Services (continued)

<table>
<thead>
<tr>
<th>Verb</th>
<th>Service Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNBDEC or CSNBDEC1</td>
<td>Decipher</td>
<td>Deciphers data using either the CDMF or the cipher block chaining mode of the DES. (The method depends on the token marking or keyword specification.) The result is called plaintext. CSNBDEC requires the plaintext and ciphertext to reside in the caller’s primary address space. CSNBDEC1 allows the plaintext and ciphertext to reside in the caller’s primary address space or in a z/OS data space.</td>
</tr>
<tr>
<td>CSNEDEC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNBDCO</td>
<td>Decode</td>
<td>Decodes an 8-byte string of data using the electronic code book mode of the DES. (This is for DES encryption only.)</td>
</tr>
<tr>
<td>CSNBENC or CSNBENC1</td>
<td>Encipher</td>
<td>Enciphers data using either the CDMF or the cipher block chaining mode of the DES. (The method depends on the token marking or keyword specification.) The result is called ciphertext. CSNBENC requires the plaintext and ciphertext to reside in the caller’s primary address space. CSNBENC1 allows the plaintext and ciphertext to reside in the caller’s primary address space or in a z/OS data space.</td>
</tr>
<tr>
<td>CSNEENC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNBSAD, CSNBSAD1,</td>
<td>Symmetric algorithm decipher</td>
<td>Deciphers data using the AES algorithm in an address space or a data space using the cipher block chaining or electronic code book modes. CSNBSAD and CSNESAD require the plaintext and ciphertext to reside in the caller’s primary address space. CSNBSAD1 and CSNESAD1 allows the plaintext and ciphertext to reside in the caller’s primary address space or in a z/OS data space.</td>
</tr>
<tr>
<td>CSNESAD, CSNESAD1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNBSAE, CSNBSAE1,</td>
<td>Symmetric algorithm encipher</td>
<td>Enciphers data using the AES algorithm in an address space or a data space using the cipher block chaining or electronic code book modes. CSNBSAE and CSNESAE require the plaintext and ciphertext to reside in the caller’s primary address space. CSNBSAE1 and CSNESAE1 allows the plaintext and ciphertext to reside in the caller’s primary address space or in a z/OS data space.</td>
</tr>
<tr>
<td>CSNESAE, CSNESAE1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verb</td>
<td>Service Name</td>
<td>Function</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CSNBSYD or CSNBSYD1</td>
<td>Symmetric key decipher</td>
<td>Deciphers data using the AES or DES algorithm in an address space or a data space using the cipher block chaining or electronic code book modes. Only clear keys are supported. CSNBSYD requires the plaintext and ciphertext to reside in the caller’s primary address space. CSNBSYD1 allows the plaintext and ciphertext to reside in the caller’s primary address space or in a z/OS data space.</td>
</tr>
<tr>
<td>CSNESYD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNBSYE or CSNBSYE1</td>
<td>Symmetric key encipher</td>
<td>Enciphers data using the AES or DES algorithm in an address space or a data space using the cipher block chaining or electronic code book modes. Only clear keys are supported. CSNBSYE requires the plaintext and ciphertext to reside in the caller’s primary address space. CSNBSYE1 allows the plaintext and ciphertext to reside in the caller’s primary address space or in a z/OS data space.</td>
</tr>
<tr>
<td>CSNESYE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNBMGN or CSNBMGN1</td>
<td>MAC generate</td>
<td>Generates a 4-, 6-, or 8-byte message authentication code (MAC) for a text string that the application program supplies. The MAC is computed using the ANSI X9.9-1 algorithm, ANSI X9.19 optional double key algorithm, the EMV padding rules or the ISO 16609 TDES algorithm. CSNBMGN requires data to reside in the caller’s primary address space. CSNBMGN1 allows data to reside in the caller’s primary address space or in a z/OS data space.</td>
</tr>
<tr>
<td>CSNBMVR or CSNBMVR1</td>
<td>MAC verify</td>
<td>Verifies a 4-, 6-, or 8-byte message authentication code (MAC) for a text string that the application program supplies. The MAC is computed using the ANSI X9.9-1 algorithm, ANSI X9.19 optional double key algorithm, the EMV padding rules or the ISO 16609 TDES algorithm. CSNBMVR requires data to reside in the caller’s primary address space. CSNBMVR1 allows data to reside in the caller’s primary address space or in a z/OS data space.</td>
</tr>
</tbody>
</table>

Chapter 7, “Verifying Data Integrity and Authenticating Messages”
<table>
<thead>
<tr>
<th>Verb</th>
<th>Service Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNBMDG or CSNBMDG1</td>
<td>MDC generate</td>
<td>Generates a 128-bit modification detection code (MDC) for a text string that the application program supplies. CSNBMDG requires data to reside in the caller's primary address space. CSNBMDG1 allows data to reside in the caller's primary address space or in a z/OS data space.</td>
</tr>
<tr>
<td>CSNBOWH or CSNBOWH1</td>
<td>One way hash generate</td>
<td>Generates a one-way hash on specified text.</td>
</tr>
<tr>
<td>CSNBSMG, CSNBSMG1, CSNESMG, CSNESMG1</td>
<td>Symmetric MAC Generate</td>
<td>Use the symmetric MAC generate callable service to generate a 96- or 128-bit message authentication code (MAC) for an application-supplied text string using a clear AES key. CSNBSMG1 allows data to reside in the caller's primary address space or in a z/OS data space.</td>
</tr>
<tr>
<td>CSNBSMV, CSNBSMV1, CSNESMV, CSNESMV1</td>
<td>Symmetric MAC Verify</td>
<td>Use the symmetric MAC verify callable service to verify a 96- or 128-bit message authentication code (MAC) for an application-supplied text string using a clear AES key. CSNBSMV1 allows data to reside in the caller's primary address space or in a z/OS data space.</td>
</tr>
<tr>
<td>CSNBCPE</td>
<td>Clear PIN encrypt</td>
<td>Formats a PIN into a PIN block format and encrypts the results.</td>
</tr>
<tr>
<td>CSNBPGN</td>
<td>Clear PIN generate</td>
<td>Generates a clear personal identification number (PIN), a PIN verification value (PVV), or an offset using one of these algorithms: IBM 3624 (IBM-PIN or IBM-PINO) IBM German Bank Pool (GBP-PIN or GBP-PINO) VISA PIN validation value (VISA-PVV) Interbank PIN (INBK-PIN) CSNBPGN executes only in special secure mode.</td>
</tr>
<tr>
<td>CSNBCPA</td>
<td>Clear PIN generate alternate</td>
<td>Generates a clear VISA PIN validation value (PVV) from an input encrypted PIN block. The PIN block may have been encrypted under either an input or output PIN encrypting key. The IBM-PINO algorithm is supported to produce a 3624 offset from a customer selected encrypted PIN.</td>
</tr>
<tr>
<td>CSNBEPG</td>
<td>Encrypted PIN generate</td>
<td>Generates and formats a PIN and encrypts the PIN block.</td>
</tr>
</tbody>
</table>
Table 6. Summary of ICSF Callable Services (continued)

<table>
<thead>
<tr>
<th>Verb</th>
<th>Service Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNBPTR</td>
<td>Encrypted PIN translate</td>
<td>Reenciphers a PIN block from one PIN-encrypting key to another and, optionally, changes the PIN block format. UKPT keywords are supported.</td>
</tr>
</tbody>
</table>
| CSNBPVIR  | Encrypted PIN verify    | Verifies a supplied PIN using one of these algorithms:  
IBM 3624 (IBM-PIN or IBM-PINO)  
IBM German Bank Pool (GBP-PIN or GBP-PINO)  
VISA PIN validation value (VISA-PVV)  
Interbank PIN (INBK-PIN)  
UKPT keywords are supported. |
| CSNBPCU   | PIN Change/Unblock      | Supports the PIN change algorithms specified in the VISA Integrated Circuit Card Specification; only available on a z890 or Requires May 2004 or later version of Licensed Internal Code (LIC). |
| CSNBBSKY  | Secure messaging for keys | Encrypts a text block, including a clear key value decrypted from an internal or external DES token.            |
| CSNBBSNP  | Secure messaging for PINs | Encrypts a text block, including a clear PIN block recovered from an encrypted PIN block.                      |
| CSNDSCB   | SET block compose       | Composes the RSA-OAEP block and the DES-encrypted block in support of the SET protocol.                       |
| CSNDSDBD  | SET block decompose     | Decomposes the RSA-OAEP block and the DES-encrypted block to provide unencrypted data back to the caller.      |
| CSNBTRV   | Transaction Validation  | Supports the generation and validation of American Express card security codes; only available on a z890 or Requires May 2004 or later version of Licensed Internal Code (LIC). |
| CSNBCSG   | VISA CVV service generate | Generates a VISA Card Verification Value (CVV) or a MasterCard Card Verification Code (CVC).                  |
| CSNBCSV   | VISA CVV service verify  | Verifies a VISA Card Verification Value (CVV) or a MasterCard Card Verification Code (CVC).                    |

Chapter 11, “Utilities”

<table>
<thead>
<tr>
<th>Verb</th>
<th>Service Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNBXBC or CSNBXCB</td>
<td>Character/nibble conversion</td>
<td>Converts a binary string to a character string or vice versa.</td>
</tr>
<tr>
<td>CSNBXEA or CSNBXAE</td>
<td>Code conversion</td>
<td>Converts EBCDIC data to ASCII data or vice versa.</td>
</tr>
<tr>
<td>CSFIQA</td>
<td>ICSF Query Algorithm</td>
<td>Use this utility to retrieve information about the cryptopraphic and hash algorithms available. You can control the amount of data that is returned by passing in different rule_array keywords.</td>
</tr>
<tr>
<td>CSFIQF</td>
<td>ICSF Query Service</td>
<td>Provides ICSF status, as well as PCICC, PCIXCC, CEX2C, and CEX3C information.</td>
</tr>
<tr>
<td>Verb</td>
<td>Service Name</td>
<td>Function</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CSNB9ED</td>
<td>X9.9 data editing</td>
<td>Edits an ASCII text string according to the editing rules of ANSI X9.9–4.</td>
</tr>
<tr>
<td>CSNPCI</td>
<td>PCI interface</td>
<td>Puts a request to a specific PCI Cryptographic Coprocessor / PCI X Cryptographic Coprocessor / Crypto Express2 Coprocessor / Crypto Express3 Cryptographic Coprocessor queue and removes the corresponding response when complete. Only the Trusted Key Entry (TKE) workstation uses this service.</td>
</tr>
<tr>
<td>CSFPKSC</td>
<td>PKSC interface</td>
<td>Puts a request to a specific cryptographic module and removes the corresponding response when complete. Only the Trusted Key Entry (TKE) workstation uses this service.</td>
</tr>
<tr>
<td>CSNAEGN</td>
<td>ANSI X9.17 EDC generate</td>
<td>Generates an ANSI X9.17 error detection code on an arbitrary length string using the special MAC key (x'0123456789ABCDEF').</td>
</tr>
<tr>
<td>CSNAKEX</td>
<td>ANSI X9.17 key export</td>
<td>Uses the ANSI X9.17 protocol to export a DATA key or a pair of DATA keys with or without an AKEK. Supports the export of a CCA IMPORTER or EXPORTER KEK. Converts a single DATA key or combines two DATA keys into a single MAC key.</td>
</tr>
<tr>
<td>CSNAKIM</td>
<td>ANSI X9.17 key import</td>
<td>Uses the ANSI X9.17 protocol to import a DATA key or a pair of DATA keys with or without an AKEK. Supports the import of a CCA IMPORTER or EXPORTER KEK. Converts a single DATA key or combines two DATA keys into a single MAC key.</td>
</tr>
<tr>
<td>CSNAKTR</td>
<td>ANSI X9.17 key translate</td>
<td>Uses the ANSI X9.17 protocol to translate, in a single service call, either one or two DATA keys or a single KEK from encryption under one AKEK to encryption under another AKEK. Converts a single DATA key or combines two DATA keys into a single MAC key.</td>
</tr>
<tr>
<td>CSNATKN</td>
<td>ANSI X9.17 transport key partial notarize</td>
<td>Permits the preprocessing of an AKEK with origin and destination identifiers to create a partially notarized AKEK.</td>
</tr>
</tbody>
</table>
Chapter 3. Introducing PKA Cryptography and Using PKA Callable Services

The preceding topic focused on DES cryptography or secret-key cryptography. This is symmetric—senders and receivers use the same key (which must be exchanged securely in advance) to encipher and decipher data.

Public key cryptography does not require exchanging a secret key. It is asymmetric—the sender and receiver each have a pair of keys, a public key and a different but corresponding private key.

You can use PKA support to exchange DES secret keys securely and to compute digital signatures for authenticating messages to users. You can also use public key cryptography in support of secure electronic transactions over open networks, using SET protocols.

PKA Key Algorithms

Public key cryptography uses a key pair consisting of a public key and a private key. The PKA public key uses one of two algorithms:

- Rivest-Shamir-Adleman (RSA)
- Digital Signature Standard (DSS)

The RSA Algorithm

The RSA algorithm is the most widely used and accepted of the public key algorithms. It uses three quantities to encrypt and decrypt text: a public exponent (PU), a private exponent (PR), and a modulus (M). Given these three and some cleartext data, the algorithm generates ciphertext as follows:

\[
\text{ciphertext} = \text{cleartext}^{PU} \pmod{M}
\]

Similarly, this operation recovers cleartext from ciphertext:

\[
\text{cleartext} = \text{ciphertext}^{PR} \pmod{M}
\]

An RSA key consists of an exponent and a modulus. The private exponent must be secret, but the public exponent and modulus need not be secret.

Digital Signature Standard (DSS)

The U.S. National Institute of Standards and Technology (NIST) defines DSS in Federal Information Processing Standard (FIPS) Publication 186.

PKA Master Keys

PKA master keys protect private keys. On the Cryptographic Coprocessor Feature, there are two PKA master keys: the Signature Master Key (SMK) and the RSA Key Management Master Key (KMMK). The SMK protects PKA private keys used only in digital signature services. The KMMK protects PKA private keys used in digital signature services and in the DES DATA key distribution functions.
**PCI Cryptographic Coprocessor**

On the PCI Cryptographic Coprocessor, PKA keys are protected by the Asymmetric-Keys Master Key (ASYM-MK). The ASYM-MK is a triple-length key used to encipher and decipher PKA keys.

In order for the PCI Cryptographic Coprocessor to function, the hash pattern of the ASYM-MK must match the hash pattern of the SMK on the Cryptographic Coprocessor Feature. The ICSF administrator installs the PKA master keys on the Cryptographic Coprocessor Feature and the ASYM-MK on the PCI Cryptographic Coprocessor by using either the pass phrase initialization routine, the Clear Master Key Entry panels, or the optional Trusted Key Entry (TKE) workstation.

Prior to PKA services being enabled on the PCI Cryptographic Coprocessor, these conditions must be met:

- The Symmetric-Keys Master Key (SYM-MK) must be installed on the PCI Cryptographic Coprocessor. It must match the Cryptographic Coprocessor Feature DES master key and match the master key that the CKDS was enciphered with.
- The PKDS is required for OS/390 V2 R9 ICSF and higher.
- The PKA master keys (SMK and KMMK) on the Cryptographic Coprocessor Feature must be installed and valid.
- The ASYM-MK PKA master key on the PCI Cryptographic Coprocessor must be installed and valid.
- The hash pattern of the ASYM-MK on the PCI Cryptographic Coprocessor must match the hash pattern of the SMK on the Cryptographic Coprocessor Feature.
- The PKDS must be initialized with the PKA master keys installed on the Cryptographic Coprocessor Feature.

**PCI X Cryptographic Coprocessor, Crypto Express2 Coprocessor, or Crypto Express3 Coprocessor**

On the PCIXCC, CEX2C, or CEX3C, PKA keys are protected by the Asymmetric-Keys Master Key (ASYM-MK). The ASYM-MK is a triple-length key used to encipher and decipher PKA keys.

In order for PKA services to function on the PCIXCC, CEX2C, or CEX3C, the Asymmetric-Keys Master Key must be installed. The ICSF administrator installs the ASYM-MK on the PCIXCC, CEX2C, or CEX3C by using either the pass phrase initialization routine, the Clear Master Key Entry panels, or the optional Trusted Key Entry (TKE) workstation.

Prior to PKA services being enabled on the PCIXCC, CEX2C, or CEX3C, these conditions must be met:

- The DES (DES-MK) must be installed on the PCIXCC.
- The DES and/or AES Master Key (DES-MK, AES-MK) must be installed on the CEX2C or CEX3C.
- The ASYM-MK master key on the PCIXCC, CEX2C, or CEX3C must be installed.
- The PKDS must be initialized with the ASYM-MK master key installed on the PCIXCC, CEX2C, or CEX3C.

**Operational private keys**

Operational private keys are protected under two layers of DES encryption. They are encrypted under an Object Protection Key (OPK) that in turn is encrypted under
the SMK/ASYM-MK or KMMK. You dynamically generate the OPK for each private key at import time or when the private key is generated on a PCI Cryptographic Coprocessor, PCI X Cryptographic Coprocessor, Crypto Express2 Coprocessor, or Crypto Express3 Coprocessor. ICSF provides a public key data set (PKDS) for the storage of application PKA keys. Although you cannot change PKA master keys dynamically, the PKA Key Token Change callable service can be executed to change a private PKA token (RSA or DSS) from encryption under the old ASYM-MK to encryption under the current ASYM-MK. This service requires a PCI Cryptographic Coprocessor, PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor, or Crypto Express3 Coprocessor and PKA callable services must be enabled. Private tokens encrypted under the KMMK will only be reenciphered if the KMMK was equal to the SMK. Private tokens in the PKDS are reenciphered when the SMK and ASYM-MK keys are changed by executing the Reencipher PKDS panel option. The reenciphered PKDS is then activated through the Activate PKDS panel option.

### PKA Callable Services

The Cryptographic Coprocessor Feature available on the IBM @server zSeries 800 and the IBM @server zSeries 900, provides RSA and DSS digital signature functions, key management functions, and DES key distribution functions.

The IBM @server zSeries 800 and the IBM @server zSeries 900 provide the ability to generate RSA keys on the PCI Cryptographic Coprocessor. ICSF provides application programming interfaces to these functions through callable services.

The PCIXCC (available on the z890 and z990), the CEX2C (available on the z9 EC, z9 BC, z10 EC and z10 BC), and the CEX3C (available on the z10 EC and z10 BC) provide RSA digital signature functions, key management functions, and DES key distribution functions, PIN, MAC and data encryption functions, and application programming interfaces to these functions through callable services. You can also generate RSA keys on the PCIXCC/CEX2C/CEX3C.

### Callable Services Supporting Digital Signatures

ICSF provides these services that support digital signatures.

**Restriction:** DSS is only supported on the IBM @server zSeries 800 and the IBM @server zSeries 900.

**Digital Signature Generate Callable Service (CSNDDSG)**

This service generates a digital signature. (There are two types of PKA public key tokens: RSA and DSS. This service can use either type.) It supports these methods:

- ANSI X9.30 (DSS)
- ANSI X9.31 (RSA)
- ISO 9796-1 (RSA)
- RSA DSI PKCS 1.0 and 1.1 (RSA)
- Padding on the left with zeros (RSA)

The input text must have been previously hashed using the one-way hash generate callable service or the MDC generation service.

**Digital Signature Verify Callable Service (CSNDDSV)**

This service verifies a digital signature using a PKA public key. (There are two types of PKA public key tokens: RSA and DSS. This service can use either type.) It supports these methods:

- ANSI X9.30 (DSS)
Callable Services for PKA Key Management

ICSF provides these services for PKA key management.

**PKA Key Generate Callable Service (CSNDPKG)**

This service generates a PKA internal token for use with the DSS algorithm in digital signature services. You can then use the PKA public key extract callable service to extract a DSS public key token from the internal key token. This service also supports the generation of RSA keys on the PCICC, PCIXCC, CEX2C, or CEX3C.

Input to the PKA key generate callable service is either a skeleton key token created by the PKA key token build callable service or a valid key token. Upon examination of the input skeleton key token, the PKA key generate service routes the key generation request as follows:

- If the skeleton is for a DSS key token, ICSF routes the request to a Cryptographic Coprocessor Feature.
- If the skeleton is for an RSA key, ICSF routes the request to any available PCICC, PCIXCC, CEX2C, or CEX3C.
- If the skeleton is for a retained RSA key, ICSF routes the request to a PCICC, PCIXCC, CEX2C, or CEX3C where the key is generated and retained for additional security.

**PKA Key Import Callable Service (CSNDPKI)**

This service imports a PKA private key, which may be RSA or DSS.

The key token to import can be in the clear or encrypted. The PKA key token build utility creates a clear PKA key token. The PKA key generate callable service generates either a clear or an encrypted PKA key token.

**PKA Key Token Build Callable Service (CSNDPKB)**

The PKA key token build callable service is a utility you can use to create an external PKA key token containing an unenciphered private RSA or DSS key. You can supply this token as input to the PKA key import callable service to obtain an operational internal token containing an enciphered private key. You can also use this service to input a clear unenciphered public RSA or DSS key and return the public key in a token format that other PKA services can use directly.

Use this service to build skeleton key tokens for input to the PKA key generate callable service for creation of RSA keys on the PCICC, PCIXCC, CEX2C, or CEX3C.

**PKA Key Token Change Callable Service (CSNDKTC)**

The PKA key token change callable service is a utility you can use to change PKA key tokens (RSA and DSS) from encipherment with the old PCICC, PCIXCC, CEX2C, or CEX3C asymmetric-keys master key to encipherment with the current PCICC, PCIXCC, CEX2C, or CEX3C asymmetric-keys master key. This callable
service only changes private internal tokens. An active PCICC, PCIXCC, CEX2C, or
CEX3C is required and PKA callable services must be enabled.

**PKA Key Translate (CSNDPKT and CSNFPKT)**
This service translates a CCA RSA key token to an external smart card key token.
An active CEX2C or CEX3C is required and PKA callable services must be
enabled.

**PKA Public Key Extract Callable Service (CSNDPKX)**
This service extracts a PKA public key token from a PKA internal (operational) or
external (importable) private key token. It performs no cryptographic verification
of the PKA private key token.

**Callable Services to Update The Public Key Data Set (PKDS)**
The Public Key Data Set (PKDS) is a repository for RSA and DSS public and
private keys. An application can store keys in the PKDS and refer to them by label
when using any of the callable services which accept public key tokens as input.
The PKDS update callable services provide support for creating and writing records
to the PKDS and reading and deleting records from the PKDS.

**PKDS Record Create Callable Service (CSNDKRC)**
This service accepts an RSA or DSS private key token in either external or internal
format, or an RSA or DSS public key token and writes a new record to the PKDS.
An application can create a null token in the PKDS by specifying a token length of
zero. The key label must be unique.

**PKDS Record Delete Callable Service (CSNDKRD)**
This service deletes a record from the PKDS. An application can specify that the
total record be deleted, or that only the contents of the record be deleted. If only
the contents of the record are deleted, the record will still exist in the PKDS but will
contain only binary zeros. The key label must be unique.

**Callable Services for Working with Retained Private Keys**
Private keys can be generated, retained, and used within the secure boundary of a
PCICC, PCIXCC, CEX2C, or CEX3C. Retained keys are generated by the PKA Key
Generate (CSNDPKG) callable service. The private key values of retained keys
never appear in any form outside the secure boundary. All retained keys have an
entry in the PKDS that identifies the PCICC, PCIXCC, CEX2C, or CEX3C where
the retained private key is stored. ICSF provides these callable services to list and
delete retained private keys.

Retained Key Delete Callable Service (CSNDRKD)
The retained key delete callable service deletes a key that has been retained within
a PCICC, PCIXCC, CEX2C, or CEX3C and also deletes the record containing the
key token from the PKDS.

Retained Key List Callable Service (CSNDRKL)
The retained key list callable service lists the key labels of private keys that are
retained within the boundaries of PCICC, PCIXCC, CEX2C, or CEX3C installed on
your server.

Clearing the retained keys on a coprocessor
The retained keys on a PCICC, PCIXCC, CEX2C, or CEX3C may be cleared.
These are the conditions under which the retained key will be lost:

• If the PCICC, PCIXCC, CEX2C, or CEX3C detects tampering (the intrusion latch
is tripped), ALL installation data is cleared: master keys, retained keys for all
domains, as well as roles and profiles.

• If the PCICC, PCIXCC, CEX2C, or CEX3C detects tampering (the secure
boundary of the card is compromised), it self-destructs and can no longer be
used.

• If you issue a command from the TKE workstation to zeroize a domain
This command zeroizes the data specific to a domain: master keys and retained
keys.

• If you issue a command from the Support Element panels to zeroize all domains.
This command zeroizes ALL installation data: master keys, retained keys and
access control roles and profiles.

Callable Services for SET Secure Electronic Transaction
SET is an industry-wide open standard for securing bankcard transactions over
open networks. The SET protocol addresses the payment phase of a transaction
from the individual, to the merchant, to the acquirer (the merchant’s current
bankcard processor). It can be used to help ensure the privacy and integrity of real
time bankcard payments over the Internet. In addition, with SET in place, everyone
in the payment process knows who everyone else is. The card holder, the
merchant, and the acquirer can be fully authenticated because the core protocol of
SET is based on digital certificates. Each participant in the payment transaction
holds a certificate that validates his or her identity. The public key infrastructure
allows these digital certificates to be exchanged, checked, and validated for every
transaction made over the Internet. The mechanics of this operation are transparent
to the application.

Under the SET protocol, every online purchase must be accompanied by a digital
certificate which identifies the card-holder to the merchant. The buyer’s digital
certificate serves as an electronic representation of the buyer’s credit card but does
not actually show the credit card number to the merchant. Once the merchant’s
SET application authenticates the buyer’s identity, it then decrypts the order
information, processes the order, and forwards the still-encrypted payment
information to the acquirer for processing. The acquirer’s SET application
authenticates the buyer’s credit card information, identifies the merchant, and
arranges settlement. With SET, the Internet becomes a safer, more secure environment for the use of payment cards.

ICSF provides these callable services that can be used in developing SET applications that make use of the S/390 and IBM zSeries cryptographic hardware at the merchant and acquirer payment gateway.

**SET Block Compose Callable Service (CSNDSBC)**
The SET Block Compose callable service performs DES encryption of data, OAEP-formatting through a series of SHA-1 hashing operations, and the RSA-encryption of the Optimal Asymmetric Encryption Padding (OAEP) block.

**SET Block Decompose Callable Service (CSNDSBD)**
The SET Block Decompose callable service decrypts both the RSA-encrypted and the DES-encrypted data.

---

**PKA Key Tokens**

PKA key tokens contain RSA or DSS private or public keys. Although DES and AES tokens are 64 bytes, PKA tokens are variable length because they contain either RSA or DSS key values, which are variable in length. Consequently, length parameters precede all PKA token parameters. The maximum allowed size is 3500 bytes. PKA key tokens consist of a token header, any required sections, and any optional sections. Optional sections depend on the token type. PKA key tokens can be public or private, and private key tokens can be internal or external. Therefore, there are three basic types of tokens, each of which can contain either RSA or DSS information:

- A public key token
- A private external key token
- A private internal key token

Public key tokens contain only the public key. Private key tokens contain the public and private key pair. Table 7 summarizes the sections in each type of token.

<table>
<thead>
<tr>
<th>Section</th>
<th>Public External Key Token</th>
<th>Private External Key Token</th>
<th>Private Internal Key Token</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RSA or DSS private key information</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>RSA or DSS public key information</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Key name (optional)</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal information</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

As with DES and AES key tokens, the first byte of a PKA key token contains the token identifier which indicates the type of token.

A first byte of X’1E’ indicates an external token with a cleartext public key and optionally a private key that is either in cleartext or enciphered by a transport key-encrypting key. An external key token is in importable key form. It can be sent on the link.

A first byte of X’1F’ indicates an internal token with a cleartext public key and a private key that is enciphered by the PKA master key and ready for internal use. An
internal key token is in operational key form. A PKA private key token must be in operational form for ICSF to use it. (PKA public key tokens are used directly in the external form.)

Formats for public and private external and internal RSA and DSS key tokens begin in "RSA Public Key Token" on page 602.

**PKA Key Management**

You can also generate PKA keys in several ways.
- Using the ICSF PKA key generate callable service.
- Using the Transaction Security System PKA key generate verb, or a comparable product from another vendor.

With a PCI X Cryptographic Coprocessor, Crypto Express2 Coprocessor, or Crypto Express3 Coprocessor, you can use the ICSF PKA key generate callable service to generate internal and external PKA tokens. You can also generate RSA keys on another system. To input a clear RSA key to ICSF, create the token with the PKA key token build callable service and import it using the PKA key import callable service. To input an encrypted RSA key, generate the key on the Transaction Security System and import it using the PKA key import callable service.

In either case, use the PKA key token build callable service to create a skeleton key token as input (see "PKA Key Token Build (CSNDPKB and CSNFPKB)" on page 399).

You can generate DSS keys on another system or on ICSF. You need to supply DSS network quantities to the PKA key generate callable service. If you generate DSS keys on another system, you can import them the same way as RSA keys. If you generate a DSS key on ICSF, you can never export it. You can use it on another ICSF host only if the same PKA master keys are installed on both systems.

The PKA key import callable service uses the clear token from the PKA key token build service or a clear or encrypted token from the Transaction Security System to securely import the key token into operational form for ICSF to use. ICSF does not permit the export of the imported PKA key.
The PKA public key extract callable service builds a public key token from a private key token.

Application RSA and DSS public and private keys can be stored in the public key data set (PKDS), a VSAM data set.

Security and Integrity of the Token

PKA private key tokens may optionally have a 64-byte private_key_name field. If private_key_name exists, ICSF uses RACHECK to verify it prior to using the token in a callable service. For additional security, the processor also validates the entire private key token.

Key Identifier for PKA Key Token

A key identifier for a PKA key token is a variable length (maximum allowed size is 3500 bytes) area that contains one of these:

- **Key label** identifies keys that are in the PKDS. Ask your ICSF administrator for the key labels that you can use.
- **Key token** can be either an internal key token, an external key token, or a null key token. Key tokens are generated by an application (for example, using the PKA key generate callable service), or received from another system that can produce external key tokens.

An internal key token can be used only on ICSF, because a PKA master key encrypts the key value. Internal key tokens contain keys in operational form only.

An external key token can be exchanged with other systems because a transport key that is shared with the other system encrypts the key value. External key tokens contain keys in either exportable or importable form.

A null key token consists of 8 bytes of binary zeros. The PKDS Record Create service can be used to write a null token to the PKDS. This PKDS record can subsequently be identified as the target token for the PKA key import or PKA key generate service.

The term key identifier is used when a parameter could be one of the previously discussed items and to indicate that different inputs are possible. For example, you may want to specify a specific parameter as either an internal key token or a key label. The key label is, in effect, an indirect reference to a stored internal key token.

Key Label

If the first byte of the key identifier is greater than X’40’, the field is considered to be holding a key label. The contents of a key label are interpreted as a pointer to a public key data set (PKDS) key entry. The key label is an indirect reference to an internal key token.

A key label is specified on callable services with the key_identifier parameter as a 64-byte character string, left-justified, and padded on the right with blanks. In most cases, the callable service does not check the syntax of the key label beyond the first byte. One exception is the key record create callable service which enforces the KGUP rules for key labels unless syntax checking is bypassed by a preprocessing exit.
A key label has this form:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>00-63</td>
<td>64</td>
<td>Key label name</td>
</tr>
</tbody>
</table>

**Key Token**

A key token is a variable length (maximum allowed size is 3500 bytes) field composed of key value and control information. PKA keys can be either public or private RSA or DSS keys. Each key token can be either an internal key token (the first byte of the key identifier is X'1F'), an external key token (the first byte of the key identifier is X'1E'), or a null PKA private key token (the first byte of the key identifier is X'00'). This is a list of private key section identifiers for internal and external private RSA key tokens:

| Table 8. Internal and External Private RSA Key Token Section Identifiers |
|-----------------|--------------------------|
| Key token        | Section identifier       |
| RSA Private Key Token, 1024-bit Modulus-Exponent External Form | X'02' |
| RSA Private Key Token, 4096-bit Chinese Remainder Theorem External Form | X'08' |
| RSA Private Key Token, 1024-bit Modulus-Exponent Internal Form (Cryptographic Coprocessor Feature) | X'02' |
| RSA Private Key Token, 1024-bit Modulus-Exponent Internal Form (PCICC, PCIXCC, CEX2C, or CEX3C) | X'06' |
| RSA Private Key Token, 4096-bit Chinese Remainder Theorem Internal Form (PCICC, PCIXCC, CEX2C, or CEX3C) | X'08' |
| RSA Private Key Token, 4096-bit Modulus-Exponent External Form (CEX2C or CEX3C) | X'09' |

**Note:** 4096-bit modulus support is only available on z9 EC, z9 BC, z10 EC and z10 BC.

See Appendix B, “Key Token Formats,” on page 597 for descriptions of the PKA key tokens.

An internal key token is a token that can be used only on the ICSF system that created it (or another ICSF system with the same PKA master key). It contains a key that is encrypted under the PKA master key.

An application obtains an internal key token by using one of the callable services such as those listed. The callable services are described in detail in **Chapter 10, “Managing PKA Cryptographic Keys.”**

- PKA key generate
- PKA key import

The PKA Key Token Change callable service can reencipher private internal tokens from encryption under the old ASYM-MK to encryption under the current ASYM-MK. PKDS Reencipher/Activate options are available to reencipher RSA and DSS internal tokens in the PKDS when the SMK/ASYM-MK keys are changed.

PKA master keys may not be changed dynamically.
For debugging information, see Appendix B, “Key Token Formats” for the format of an internal key token.

If the first byte of the key identifier is X’1E’, the key identifier is interpreted as an external key token. An external PKA key token contains key (possibly encrypted) and control information. By using the external key token, you can exchange keys between systems.

An application obtains the external key token by using one of the callable services such as those listed. They are described in detail in Chapter 10, “Managing PKA Cryptographic Keys.”

- PKA public key extract
- PKA key token build
- PKA key generate

For debugging information, see Appendix B, “Key Token Formats” for the format of an external key token.

If the first byte of the key identifier is X’00’, the key identifier is interpreted as a null key token.

For debugging information, see Appendix B, “Key Token Formats” for the format of a null key token.

---

**The Transaction Security System and ICSF Portability**

The Transaction Security System PKA verbs from releases prior to 1996 can run only on the Transaction Security System. The PKA96 release of the Transaction Security System PKA verbs generally runs on ICSF without change. As with DES cryptography, you cannot interchange internal PKA tokens but can interchange external tokens.

---

**Summary of the PKA Callable Services**

Table 9 lists the PKA callable services, described in this publication, and their corresponding verbs. (The PKA services start with CSNDxxx and have corresponding CSFxxx names.) This table also references the topic that describes the callable service.

<table>
<thead>
<tr>
<th>Verb</th>
<th>Service Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 8, “Financial Services”</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNDSBC</td>
<td>SET block compose</td>
<td>Composes the RSA-OAEP block and the DES-encrypted block in support of the SET protocol.</td>
</tr>
<tr>
<td>CSNDSBD</td>
<td>SET block decompose</td>
<td>Decomposes the RSA-OAEP block and the DES-encrypted block to provide unencrypted data back to the caller.</td>
</tr>
<tr>
<td>Chapter 9, “Using Digital Signatures”</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNDDSG</td>
<td>Digital signature generate</td>
<td>Generates a digital signature using a PKA private key supporting RSA and DSS algorithms.</td>
</tr>
<tr>
<td>CSNFDMSG</td>
<td>Digital signature verify</td>
<td>Verifies a digital signature using a PKA public key supporting RSA and DSS algorithms.</td>
</tr>
<tr>
<td>Chapter 10, “Managing PKA Cryptographic Keys”</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 9. Summary of PKA Callable Services (continued)

<table>
<thead>
<tr>
<th>Verb</th>
<th>Service Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNDPKG</td>
<td>PKA key generate</td>
<td>Generates a DSS internal token for use in digital signature services and RSA keys for use on the PCICC, PCIXCC, CEX2C, or CEX3C.</td>
</tr>
<tr>
<td>CSNFPKG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNDPKI</td>
<td>PKA key import</td>
<td>Imports a PKA key token containing either a clear PKA key or a PKA key enciphered under a limited authority IMP-PKA KEK.</td>
</tr>
<tr>
<td>CSNFPKI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNDPKB</td>
<td>PKA key token build</td>
<td>Creates an external PKA key token containing a clear private RSA or DSS key. Using this token as input to the PKA key import callable service returns an operational internal token containing an enciphered private key. Using CSNDPKB on a clear public RSA or DSS key, returns the public key in a token format that other PKA services can directly use. CSNDPKB can also be used to create a skeleton token for input to the PKA Key Generate service for the generation of an internal DSS or RSA key token.</td>
</tr>
<tr>
<td>CSNFPKB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNDKTC</td>
<td>PKA key token change</td>
<td>Changes PKA key tokens (RSA and DSS) from encipherment with the old PCICC, PCIXCC, CEX2C, or CEX3C asymmetric-keys master key to encipherment with the current PCICC, PCIXCC, CEX2C, or CEX3C asymmetric-keys master key. This callable service only changes private internal tokens.</td>
</tr>
<tr>
<td>CSNDPKT</td>
<td>PKA key translate</td>
<td>Translates a CCA RSA key token to a smart card format.</td>
</tr>
<tr>
<td>CSNFPKT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNDPKX</td>
<td>PKA public key extract</td>
<td>Extracts a PKA public key token from a supplied PKA internal or external private key token. Performs no cryptographic verification of the PKA private token.</td>
</tr>
<tr>
<td>CSNDPKX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNDKRC</td>
<td>PKDS record create</td>
<td>Writes a new record to the PKDS.</td>
</tr>
<tr>
<td>CSNFKRC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNDKRD</td>
<td>PKDS record delete</td>
<td>Delete a record from the PKDS.</td>
</tr>
<tr>
<td>CSNFKRD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNDKRR</td>
<td>PKDS record read</td>
<td>Read a record from the PKDS and return the contents of that record.</td>
</tr>
<tr>
<td>CSNDKRW</td>
<td>PKDS record write</td>
<td>Write over an existing record in the PKDS.</td>
</tr>
<tr>
<td>CSNDRKL</td>
<td>Retained key list</td>
<td>Lists key labels of keys that have been retained within all currently active PCICCs, PCIXCCs, CEX2Cs, or CEX3Cs.</td>
</tr>
<tr>
<td>CSNFRKL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSNDRKD</td>
<td>Retained key delete</td>
<td>Deletes a key that has been retained within the PCICCs, PCIXCCs, CEX2Cs, or CEX3Cs.</td>
</tr>
<tr>
<td>CSNFRKD</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 4. Introducing PKCS #11 and using PKCS #11 callable services

The Integrated Cryptographic Service Facility has implemented callable service in support of PKCS #11. A callable service is a routine that receives control using a CALL statement in an application language. Each callable service performs one or more functions, including:

- initializing and deleting PKCS11 tokens
- creating, reading, updating and deleting PKCS11 objects

Many services have hardware requirements. See each service for details. All new callable services will be invocable in AMODE(24), AMODE(31), or AMODE(64).

For more information about PKCS #11 see z/OS Cryptographic Services ICSF Writing PKCS #11 Applications.

PKCS #11 Management Services

ICSF provides callable services that support PKCS #11 token and object management. The following table summarizes these callable services. For complete syntax and reference information, refer to Part 3, “PKCS #11 Callable Services,” on page 491.

Table 10. Summary of PKCS #11 callable services

<table>
<thead>
<tr>
<th>Verb</th>
<th>Service Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSFPDVK</td>
<td>PKCS #11 Derive key</td>
<td>Generate a new secret key object from an existing key object</td>
</tr>
<tr>
<td>CSFPDMK</td>
<td>PKCS #11 Derive multiple keys</td>
<td>Generate multiple secret key objects and protocol dependent keying material from an existing secret key object</td>
</tr>
<tr>
<td>CSFPHMG</td>
<td>PKCS #11 Generate HMAC</td>
<td>Generate a hashed message authentication code (MAC)</td>
</tr>
<tr>
<td>CSFPGKP</td>
<td>PKCS #11 Generate key pair</td>
<td>Generate an RSA, DSA, Elliptic Curve, or Diffie-Hellman key pair</td>
</tr>
<tr>
<td>CSFPGSK</td>
<td>PKCS #11 Generate secret key</td>
<td>Generate a secret key or set of domain parameters</td>
</tr>
<tr>
<td>CSFPGAV</td>
<td>PKCS #11 Get attribute value</td>
<td>List the attributes of a PKCS11 object</td>
</tr>
<tr>
<td>CSFPOWH</td>
<td>PKCS #11 One-way hash generate</td>
<td>Generate a one-way hash on specified text</td>
</tr>
</tbody>
</table>
| CSFPPKS      | PKCS #11 Private key sign | • Decrypt or sign data using an RSA private key using zero-pad or PKCS #1 v1.5 formatting  
                        |                                                                 | • Sign data using a DSA private key                                          |
                        |                  | • Sign data using an Elliptic Curve private key in combination with DSA |
| CSFPPRF      | PKCS #11 Pseudo-random function | Generate pseudo-random output of arbitrary length.                      |
### Table 10. Summary of PKCS #11 callable services (continued)

<table>
<thead>
<tr>
<th>Verb</th>
<th>Service Name</th>
<th>Function</th>
</tr>
</thead>
</table>
| CSFPPKV  | PKCS #11 Public key verify| • Encrypt or verify data using an RSA public key using zero-pad or PKCS #1 v1.5 formatting. For encryption, the encrypted data is returned  
  • Verify a signature using a DSA public key. No data is returned  
  • Verify a signature using an Elliptic Curve public key in combination with DSA. No data is returned |
| CSFPSKD  | PKCS #11 Secret key decrypt| Decipher data using a clear symmetric key                                  |
| CSFPSKE  | PKCS #11 Secret key encrypt| Encipher data using a clear symmetric key                                  |
| CSFPSAV  | PKCS #11 Set attribute value| Update the attributes of a PKCS11 object                                  |
| CSFPTRC  | PKCS #11 Token record create| Initialize or re-initialize a z/OS PKCS #11 token, creates or copies a token object in the token data set and creates or copies a session object for the current PKCS #11 session |
| CSFPTRD  | PKCS #11 Token record delete| Delete a z/OS PKCS #11 token, token object, or session object             |
| CSFPRRL  | PKCS #11 Token record list| Obtain a list of z/OS PKCS #11 tokens. The caller must have SAF authority to the token. Also obtains a list of token and session objects for a token. Use a search template to restrict the search for specific attributes. |
| CSFPWK   | PKCS #11 Unwrap key Unwrap and create a key object using another key   |
| CSFPWHV  | PKCS #11 Verify HMAC Verify a hash message authentication code (MAC)    |
| CSFPWPK  | PKCS #11 Wrap key Wrap a key with another key                           |

### Attribute List

The attributes of an object can be the input and the output of a service. The format of the attributes is shown here and applies to all PKCS #11 callable services. For the token record list service, the search_template has the same format as an attribute list. The lengths in the attribute list and attribute structures are unsigned integers.

An attribute list is a structure in this format:

<table>
<thead>
<tr>
<th>Number of attributes</th>
<th>Attribute</th>
<th>Attribute</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 bytes</td>
<td>4 + 2 + length of value bytes</td>
<td>4 + 2 + length of value bytes</td>
<td>...</td>
</tr>
</tbody>
</table>
Each attribute is a structure in this format:

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Length of value ((n))</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 bytes</td>
<td>2 bytes</td>
<td>(n) bytes</td>
</tr>
</tbody>
</table>

### Handles

A handle is a 44-byte identifier for a token or an object. The format of the handle is as follows:

<table>
<thead>
<tr>
<th>Name of token or object</th>
<th>Sequence number</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 bytes</td>
<td>8 bytes</td>
<td>4 bytes</td>
</tr>
</tbody>
</table>

The token name in the first 32 bytes of the handle is provided by the PKCS #11 application when the token or object is created. The first character of the name must be alphabetic or a national character (“#”, “$”, or “@”). Each of the remaining characters can be alphanumeric, a national character (“#”, “$”, or “@”), or a period (“.”).

The sequence number is a hexadecimal number stored as the EBCDIC representation of 8 hexadecimal numbers. The sequence number field in a token is EBCDIC blanks. The token record contains a last-used sequence number field, which is incremented each time an object associated with the token is created. This sequence number value is placed in the handle of the newly-created object.

The ID field is 4 characters. The first character contains an EBCDIC “T” if the handle belongs to a token object, “S” if the handle belongs to a session object, or blank if the handle belongs to a token. The last three characters must be EBCDIC blanks.
Part 2. CCA Callable Services

This publication introduces DES, AES and PKA callable services.
Chapter 5. Managing Symmetric Cryptographic Keys

This topic describes the callable services that generate and maintain cryptographic keys.

Using ICSF, you can generate keys using either the key generator utility program or the key generate callable service. ICSF provides a number of callable services to assist you in managing and distributing keys and maintaining the cryptographic key data set (CKDS).

This topic describes these callable services:
- "Clear Key Import (CSNBCKI and CSNECKI)" on page 89
- "Control Vector Generate (CSNBCVG)" on page 92
- "Control Vector Translate (CSNBCVT)" on page 92
- "Cryptographic Variable Encipher (CSNBCVE)" on page 95
- "Data Key Export (CSNBDRX)" on page 98
- "Data Key Import (CSNBDKM)" on page 100
- "Diversified Key Generate (CSNBDKG)" on page 102
- "Key Export (CSNBKEX)" on page 107
- "Key Generate (CSNBKG and CSNEKGN)" on page 111
- "Key Import (CSNBKIM)" on page 123
- "Key Part Import (CSNBKPI)" on page 127
- "Key Record Create (CSNBKRC)" on page 131
- "Key Record Delete (CSNBKRD)" on page 133
- "Key Record Read (CSNBKRR)" on page 135
- "Key Record Write (CSNBKRW)" on page 137
- "Key Test (CSNBKYT)" on page 139
- "Key Test Extended (CSNBKETYX)" on page 144
- "Key Token Build (CSNBKTB)" on page 147
- "Key Translate (CSNBKTR)" on page 156
- "Multiple Clear Key Import (CSNBCKM and CSNECKM)" on page 159
- "Multiple Secure Key Import (CSNBSKM)" on page 161
- "PKA Decrypt (CSNDPKD and CSNPFPK)" on page 166
- "PKA Encrypt (CSNDPKE and CSNPFPE)" on page 171
- "Prohibit Export (CSNBPEX)" on page 175
- "Prohibit Export Extended (CSNBPXXE)" on page 177
- "Random Number Generate (CSNBRNG, CSNERNG, CSNBRNL and CSNERNL)" on page 178
- "Remote Key Export (CSNDRKX)" on page 182
- "Secure Key Import (CSNBSKI)" on page 189
- "Symmetric Key Export (CSNDSYX and CSNFYSX)" on page 192
- "Symmetric Key Generate (CSNDSYG)" on page 196
- "Symmetric Key Import (CSNDSYI and CSNFYSY)" on page 201
- "Transform CDMF Key (CSNBTCMK)" on page 208
- "Trusted Block Create (CSNDTBC)" on page 208
- "User Derived Key (CSFUDK)" on page 211

Clear Key Import (CSNBCKI and CSNECKI)

Use the clear key import callable service to import a clear DATA key that is to be used to encipher or decipher data. This callable service can import only DATA keys. Clear key import accepts an 8-byte clear DATA key, enciphers it under the master key, and returns the encrypted DATA key in operational form in an internal key token.
Clear Key Import (CSNBCKI and CSNECKI)

If the clear key value does not have odd parity in the low-order bit of each byte, the service returns a warning value in the `reason_code` parameter. The callable service does not adjust the parity of the key.

**Note:** To import 16-byte or 24-byte DATA keys, use the multiple clear key import callable service that is described in "Multiple Clear Key Import (CSNBCKM and CSNECKM)" on page 159. The multiple clear key import service supports AES DATA keys.

This callable service supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSNECKI.

Format

```call
CALL CSNBCKI(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    clear_key,
    key_identifier )
```

Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFFF' (2 gigabytes). The data is identified in the `exit_data` parameter.

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**clear_key**

Direction: Input  
Type: String
Clear Key Import (CSNBCKI and CSNECKI)

The clear_key specifies the 8-byte clear key value to import.

**key_identifier**

Direction: Input/Output

Type: String

A 64-byte string that is to receive the internal key token. "Key Identifier for Key Token" on page 7 describes the internal key token.

**Usage Notes**

These usage notes only apply to CCF systems.

This service produces an internal DATA token with a control vector which is usable on the Cryptographic Coprocessor Feature. If a valid internal token is supplied as input to the service in the key_identifier field, that token's control vector will not be used in the encryption of the clear key value.

This service marks this internal key token CDMF or DES, according to the system's default encryption algorithm, unless token copying overrides this. The service marks this internal key token CDMF or DES, according to the system's default encryption algorithm, unless token copying overrides this. See "System Encryption Algorithm" on page 45 for details.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

**Table 11. Clear key import required hardware**

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>PCI X Cryptographic Coprocessor/Crypto Express2 Coprocessor</td>
<td>There are no internal token markings for CDMF or DES. There is no token copying.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express2 Coprocessor</td>
<td>There are no internal token markings for CDMF or DES. There is no token copying.</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express2 Coprocessor</td>
<td>There are no internal token markings for CDMF or DES. There is no token copying.</td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

Control Vector Generate (CSNBCVG)

The Control Vector Generate callable service builds a control vector from keywords specified by the key_type and rule_array parameters.
Control Vector Generate (CSNBCVG)

Format

CALL CSNBCVG(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  key_type,
  rule_array_count,
  rule_array,
  reserved,
  control_vector )

Parameters

return_code
Direction: Output Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code
Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

exit_data_length
Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFFFF' (2 gigabytes). The data is defined in the exit_data parameter.

exit_data
Direction: Input/Output Type: String

The data that is passed to the installation exit.

key_type
Direction: Input Type: String

A string variable containing a keyword for the key type. The keyword is 8 bytes in length, left justified, and padded on the right with space characters. It is taken from this list:

CIPHER DATAM IKEYXLAT OPINENC
CVARDEC DATAMV IMPORTER PINGEN
CVARENC DECRYPT IPINENC PINVER
CVARPINE DKYGENK KEYGENKY SECMSG
CVARXCVL ENCIPHER MAC
CVARXCVR EXPORTER MACVER
DATA OKEYXLAT
Control Vector Generate (CSNBCVG)

rule_array_count
Direction: Input Type: Integer

The number of keywords you are supplying in the rule_array parameter.

rule_array
Direction: Input Type: Character String

Keywords that provide control information to the callable service. Each keyword is left justified in 8-byte fields, and padded on the right with blanks. All keywords must be in contiguous storage. "Key Token Build (CSNBKTB)" on page 147 illustrates the key type and key usage keywords that can be combined in the Control Vector Generate and Key Token Build callable services to create a control vector. The rule array keywords are:

CLR8-ENC DKYL5 IBM-PIN NOOFFSET
CPINENC DKYL6 IBM-PINO OPEX
CPINGEN DKYL7 IMEX OPEX
CPINGENA DMAC IMIM REFORMAT
DALL DMKEY IMPORT SINGLE
DATA DMPIN INBK-PIN SMKEY
DDATA DMV KEY-PART SMPIN
DEXP DOUBLE KEYLN8 TRANSLAT
DIMP DPVR KEYLN16 UKPT
DKYL0 EPINGEN MIXED VISA-PVV
DKYL1 EPINVER NO-SPEC XLATE
DKYL2 EXEX NO-XPORT XPORT-OK
DKYL3 EXPORT
DKYL4 GBP-PIN
GBP-PINO

Note: CLR8-ENC or UKPT must be coded in rule_array when the KEYGENKY key type is coded. When the SECMSG key_type is coded, either SMKEY or SMPIN must be specified in the rule_array.

reserved
Direction: Input Type: String

The reserved parameter must be a variable of 8 bytes of X'00'.

control_vector
Direction: Output Type: String

A 16-byte string variable in application storage where the service returns the generated control vector.

Usage Notes
See Table 43 on page 154 for an illustration of key type and key usage keywords that can be combined in the Control Vector Generate and Key Token Build callable services to create a control vector.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
Control Vector Generate (CSNBCVG)

### Table 12. Control vector generate required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>None.</td>
<td></td>
</tr>
</tbody>
</table>

Control Vector Translate (CSNBCVT)

The Control Vector Translate callable service changes the control vector used to encipher an external key.

See "Changing Control Vectors with the Control Vector Translate Callable Service" on page 643 for additional information about this service.

### Format

```plaintext
CALL CSNBCVT(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    KEK_key_identifier,
    source_key_token,
    array_key_left,
    mask_array_left,
    array_key_right,
    mask_array_right,
    rule_array_count,
    rule_array,
    target_key_token )
```

### Parameters

- **return_code**
  
  **Direction:** Output  
  **Type:** Integer

  The return code specifies the general result of the callable service. [Appendix A, "ICSF and TSS Return and Reason Codes"] lists the return codes.
Control Vector Translate (CSNBCVT)

**reason_code**
Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit_data_length**
Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFFFF' (2 gigabytes). The data is defined in the exit_data parameter.

**exit_data**
Direction: Input/Output Type: String

The data that is passed to the installation exit.

**KEK_key_identifier**
Direction: Input/Output Type: String

The 64-byte string variable containing an internal key token or the key label of an internal key token record containing the key-encrypting key. The control vector in the internal key token must specify the key type of IMPORTER, EXPORTER, IKEYXLAT, or OKEYXLAT.

**source_key_token**
Direction: Input Type: String

A 64-byte string variable containing the external key token with the key and control vector to be processed.

**array_key_left**
Direction: Input/Output Type: String

A 64-byte string variable containing an internal key token or a key label of an internal key token record that deciphers the left mask array. The internal key token must contain a control vector specifying a CVARXCVL key type.

**mask_array_left**
Direction: Input Type: String

A string of seven 8-byte elements containing the mask array enciphered under the left array key.

**array_key_right**
Direction: Input/Output Type: String
Control Vector Translate (CSNBCVT)

A 64-byte string variable containing an internal key token or a key label of an internal key token record that deciphers the right mask array. The internal key token must contain a control vector specifying a CVARXCVR key type.

**mask_array_right**
- **Direction:** Input
- **Type:** String

A string of seven 8-byte elements containing the mask array enciphered under the right array key.

**rule_array_count**
- **Direction:** Input
- **Type:** Integer

An integer containing the number of elements in the rule array. The value of the `rule_array_count` must be zero, one, or two for this service. If the rule array count is zero, the default keywords ADJUST and LEFT are used.

**rule_array**
- **Direction:** Input
- **Type:** Character String

The `rule_array` parameter is an array of keywords. The keywords are 8 bytes in length, and must be left-justified and padded on the right with space characters. The `rule_array` keywords are:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADJUST</td>
<td>Ensures that all target key bytes have odd parity. This is the default.</td>
</tr>
<tr>
<td>NOADJUST</td>
<td>Prevents the parity of the target being altered.</td>
</tr>
</tbody>
</table>

**Key-portion Rule (optional)**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOTH</td>
<td>Causes both halves of a 16-byte source key to be processed with the result placed into corresponding halves of the target key. When you use the BOTH keyword, the mask array must be able to validate the translation of both halves.</td>
</tr>
<tr>
<td>LEFT</td>
<td>Causes an 8-byte source key, or the left half of a 16-byte source key, to be processed with the result placed into both halves of the target key. This is the default.</td>
</tr>
<tr>
<td>RIGHT</td>
<td>Causes the right half of a 16-byte source key to be processed with the result placed into the right half of the target key. The left half is copied unchanged (still enciphered) from the source key.</td>
</tr>
<tr>
<td>SINGLE</td>
<td>Causes the left half of the source key to be processed with the result placed into the left half of the target key token. The right half of the target key is unchanged.</td>
</tr>
</tbody>
</table>

**target_key_token**
- **Direction:** Input/Output
- **Type:** String
Control Vector Translate (CSNBCVT)

A 64-byte string variable containing an external key token with the new control vector. This key token contains the key halves with the new control vector.

Restrictions

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

If KEK_key_identifier is a label of an IMPORTER or EXPORTER key, the label must be unique in the CKDS.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 14. Control vector translate required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>PCI Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>PCI X Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

Cryptographic Variable Encipher (CSNBCVE)

The Cryptographic Variable Encipher callable service uses a CVARENC key to encrypt plaintext by using the Cipher Block Chaining (CBC) method. You can use this service to prepare a mask array for the Control Vector Translate service. The plaintext must be a multiple of eight bytes in length.
Cryptographic Variable Encipher (CSNBCVE)

Format

CALL CSNBCVE(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    c-variable_encrypting_key_identifier,
    text_length,
    plaintext,
    initialization_vector,
    ciphertext )  

Parameters

return_code
Direction: Output  Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code
Direction: Output  Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

exit_data_length
Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is defined in the exit_data parameter.

exit_data
Direction: Input/Output  Type: String

The data that is passed to the installation exit.

c-variable_encrypting_key_identifier
Direction: Input/Output  Type: String

The 64-byte string variable containing an internal key or a key label of an internal key token record in the CKDS. The internal key must contain a control vector that specifies a CVARENC key type.

text_length
Direction: Input  Type: Integer

An integer variable containing the length of the plaintext and the returned ciphertext.
Cryptographic Variable Encipher (CSNBCVE)

**plaintext**

*Direction*: Input  
*Type*: String

A string of length 8 to 256 bytes which contains the plaintext. The data must be a multiple of 8 bytes.

**initialization_vector**

*Direction*: Input  
*Type*: String

A string variable containing the 8-byte initialization vector that the service uses in encrypting the plaintext.

**ciphertext**

*Direction*: Output  
*Type*: String

The field which receives the ciphertext. The length of this field is the same as the length of the plaintext.

**Restrictions**

- The text length must be a multiple of 8 bytes.
- The maximum length of text that the security server can process is 256 bytes.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 15. Cryptographic variable encipher required hardware*

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>PCI Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>PCI Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>
Data Key Export (CSNBDKX)

Use the data key export callable service to reencrypt a data-encrypting key (key type of DATA only) from encryption under the master key to encryption under an exporter key-encrypting key. The reencrypted key is in a form suitable for export to another system.

The data key export service generates a key token with the same key length as the input token’s key.

Format

```
CALL CSNBDKX(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    source_key_identifier,
    exporter_key_identifier,
    target_key_identifier )
```

Parameters

**return_code**
- Direction: Output
- Type: Integer

The return code specifies the general result of the callable service. [Appendix A, “ICSF and TSS Return and Reason Codes”](#) lists the return codes.

**reason_code**
- Direction: Output
- Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. [Appendix A, “ICSF and TSS Return and Reason Codes”](#) lists the reason codes.

**exit_data_length**
- Direction: Input/Output
- Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the `exit_data` parameter.

**exit_data**
- Direction: Input/Output
- Type: String

The data that is passed to the installation exit.

**source_key_identifier**
- Direction: Input/Output
- Type: String

The input key identifier of the source key that is to be reencrypted.
Data Key Export (CSNBDKX)

A 64-byte string for an internal key token or label that contains a data-encrypting key to be reenciphered. The data-encrypting key is encrypted under the master key.

**exporter_key_identifier**

Direction: Input/Output  
Type: String  

A 64-byte string for an internal key token or key label that contains the exporter key-encrypting key. The data-encrypting key previously discussed will be encrypted under this exporter key-encrypting key.

**target_key_identifier**

Direction: Input/Output  
Type: String  

A 64-byte field that is to receive the external key token, which contains the reenciphered key that has been exported. The reenciphered key can now be exchanged with another cryptographic system.

**Restrictions**

For existing TKE V3.1 (or higher) users, you may have to explicitly enable new access control points. Current applications will fail if they use an equal key halves exporter to export a key with unequal key halves. You must have access control point 'Data Key Export - Unrestricted' explicitly enabled.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

When the service is processed on the CCF, ICSF examines the data encryption algorithm bits on the exporter key-encrypting key and DATA key for consistency. It does not export a CDMF key under a DES-marked key-encrypting key or a DES key under a CDMF-marked key-encrypting key. ICSF does not propagate the data encryption marking on the operational key to the external token.

Token marking for DES/CDMF on DATA and key-encrypting keys is ignored on a PCICC, PCIXCC, CEX2C, or CEX3C.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
# Data Key Export (CSNBDKX)

## Table 16. Data key export required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>PCI Cryptographic Coprocessor</td>
<td>ICSF routes the request to a PCI Cryptographic Coprocessor if the control vector of the exporter_key_identifier cannot be processed on the Cryptographic Coprocessor Feature.</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

## Data Key Import (CSNBDKM)

Use the data key import callable service to import an encrypted source DES single-length, double-length or triple-length DATA key and create or update a target internal key token with the master key enciphered source key.

### Format

```call
CALL CSNBDKM(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    source_key_token,
    importer_key_identifier,
    target_key_identifier)
```

### Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. [Appendix A. ICSF and TSS Return and Reason Codes](#) lists the return codes.

**reason_code**

Direction: Output  
Type: Integer
Data Key Import (CSNBDKM)

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

exit_data_length

Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

exit_data

Direction: Input/Output Type: String

The data that is passed to the installation exit.

source_key_token

Direction: Input Type: String

64-byte string variable containing the source key to be imported. The source key must be an external token or null token. The external key token must indicate that a control vector is present; however, the control vector is usually valued at zero. A double-length key that should result in a default DATA control vector must be specified in a version X'01' external key token. Otherwise, both single and double-length keys are presented in a version X'00' key token. For the null token, the service will process this token format as a DATA key encrypted by the importer key and a null (all zero) control vector.

importer_key_identifier

Direction: Input/Output Type: String

A 64-byte string variable containing the (IMPORTER) transport key or key label of the transport key used to decipher the source key.

target_key_identifier

Direction: Output Type: String

A 64-byte string variable containing a null key token or an internal key token. The key token receives the imported key.

Restrictions

For existing TKE V3.1 (or higher) users, you may have to explicitly enable new access control points. Current applications will fail if they use an equal key halves importer to import a key with unequal key halves. You must have access control point 'Data Key Import - Unrestricted' explicitly enabled.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).
Data Key Import (CSNBDKM)

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

This service does not adjust the key parity of the source key.

CDMF/DES token markings will be ignored.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 17. Data key import required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>PCI Cryptographic Coprocessor</td>
<td>Does not support triple length DATA keys.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

Diversified Key Generate (CSNBDKG)

Use the diversified key generate service to generate a key based on the key-generating key, the processing method, and the parameter supplied. The control vector of the key-generating key also determines the type of target key that can be generated.

To use this service, specify:

- The rule array keyword to select the diversification process.
- The operational key-generating key from which the diversified keys are generated. The control vector associated with this key restricts the use of this key to the key generation process. This control vector also restricts the type of key that can be generated.
- The data and length of data used in the diversification process.
- The generated-key may be an internal token or a skeleton token containing the desired CV of the generated-key. The generated key CV must be one that is permitted by the processing method and the key-generating key. The generated-key will be returned in this parameter.
- A key generation method keyword. Some keywords require Requires May 2004 or later version of Licensed Internal Code (LIC) or a z890.
Diversified Key Generate (CSNBDKG)

This service generates diversified keys as follows:
- Determines if it can support the process specified in rule array.
- Recovers the key-generating key and checks the key-generating key class and the specified usage of the key-generating key.
- Determines that the control vector in the generated-key token is permissible for the specified processing method.
- Determines that the control vector in the generated-key token is permissible by the control vector of the key-generating key.
- Determines the required data length from the processing method and the generated-key CV. Validates the data_length.
- Generates the key appropriate to the specific processing method. Adjusts parity of the key to odd. Creates the internal token and returns the generated diversified key.

Format

```
CALL CSNBDKG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    generating_key_identifier,
    data_length,
    data,
    key_identifier,
    generated_key_identifier)
```

Parameters

**return_code**

Direction: Output Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**

Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.
Diversified Key Generate (CSNBDKG)

exit_data
Direction: Input/Output Type: String

The data that is passed to the installation exit.

rule_array_count
Direction: Input Type: Integer

The number of keywords you supplied in the rule_array parameter. The only valid value is 1.

rule_array
Direction: Input Type: String

The keyword that provides control information to the callable service. The processing method is the algorithm used to create the generated key. The keyword is left justified and padded on the right with blanks.

Table 18. Rule Array Keywords for Diversified Key Generate

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR8-ENC</td>
<td>Specifies that 8-bytes of clear data shall be multiply encrypted with the generating key. The generating_key_identifier must be a KEYGENKY key type with bit 19 of the control vector set to 1. The control vector in generated_key_identifier must specify a single-length key. The key type may be DATA, MAC, or MACVER.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: CIPHER class keys are not supported.</td>
</tr>
<tr>
<td>TDES-DEC</td>
<td>Data supplied may be 8 or 16 bytes of clear data. If the generating_key_identifier specifies a single length key, then 8-bytes of data is TDES decrypted under the generating_key_identifier. If the generating_key_identifier specifies a double length key, then 16-bytes of data is TDES ECB mode decrypted under the generating_key_identifier. No formatting of data is done prior to encryption. The generating_key_identifier must be a DKYGENKY key type, with appropriate usage bits for the desired generated key.</td>
</tr>
<tr>
<td>TDES-ENC</td>
<td>Data supplied may be 8 or 16 bytes of clear data. If the generating_key_identifier specifies a single length key, then 8-bytes of data is TDES encrypted under the generating_key_identifier. If the generating_key_identifier specifies a double length key, then 16-bytes of data is TDES ECB mode encrypted under the generating_key_identifier. No formatting of data is done prior to encryption. The generating_key_identifier must be a DKYGENKY key type, with appropriate usage bits for the desired generated key. The generating_key_identifier may be a single or double length key with a CV that is permitted by the generating_key_identifier.</td>
</tr>
</tbody>
</table>
Table 18. Rule Array Keywords for Diversified Key Generate (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDES-XOR</td>
<td>Requires May 2004 or later version of Licensed Internal Code (LIC). It combines the function of the existing TDES-ENC and SESS-XOR into one step. The generating key must be a level 0 DKYGENKY and cannot have replicated halves. The session key generated must be double length and the allowed key types are DATA, DATAC, MAC, MACVER, DATAM, DATAMV, SMPIN and SMKEY. Key type must be allowed by the generating key control vector.</td>
</tr>
<tr>
<td>TDESEMV2</td>
<td>Requires May 2004 or later version of Licensed Internal Code (LIC): supports generation of a session key by the EMV 2000 algorithm (This EMV2000 algorithm uses a branch factor of 2). The generating key must be a level 0 DKYGENKY and cannot have replicated halves. The session key generated must be double length and the allowed key types are DATA, DATAC, MAC, MACVER, DATAM, DATAMV, SMPIN and SMKEY. Key type must be allowed by the generating key control vector.</td>
</tr>
<tr>
<td>TDESEMV4</td>
<td>Requires May 2004 or later version of Licensed Internal Code (LIC): supports generation of a session key by the EMV 2000 algorithm (This EMV2000 algorithm uses a branch factor of 4). The generating key must be a level 0 DKYGENKY and cannot have replicated halves. The session key generated must be double length and the allowed key types are DATA, DATAC, MAC, MACVER, DATAM, DATAMV, SMPIN and SMKEY. Key type must be allowed by the generating key control vector.</td>
</tr>
</tbody>
</table>

**Processing Method for updating a diversified key**

**SESS-XOR**

Specifies the VISA method for session key generation. Data supplied may be 8 or 16 bytes of data depending on whether the generating_key_identifier is a single or double length key. The 8 or 16 bytes of data is XORed with the clear value of the generating_key_identifier. The generated_key_identifier has the same control vector as the generating_key_identifier. The generating_key_identifier may be DATA/DATAC, MAC/DATAM or MACVER/DATAMV key types.

**generating_key_identifier**

Direction: Input/Output  
Type: String

The label or internal 64 byte token of a key that is a DKYLO, DKYGENKY or a subtype appropriate to the session key to be derived. The type of key-generating key depends on the processing method.

**data_length**

Direction: Input  
Type: Integer
Diversified Key Generate (CSNBDKG)

The length of the data parameter that follows. Length depends on the processing method and the generated key. The data length for TDESEMV4 or TDESEMV2 is either 18 or 34.

data
Direction: Input
Type: String

Data input to the diversified key or session key generation process. Data depends on the processing method and the generated_key_identifier.

For TDESEMV4 or TDESEMV2 the data is either 18 bytes (36 digits) or 34 bytes (68 digits) or data comprised of:

- 16 bytes (32 digits) of card specific data used to create the card specific intermediate key (UDK) as per the TDES-ENC method. This will typically be the PAN and PAN Sequence number as per the EMV specifications
- 2 bytes (4 digits) of ATC (Application Transaction Count)
- (optional) 16 bytes (32 digits) of IV (Initial Value) used in the EMV

key_identifier
Direction: Input/Output
Type: String

This parameter is currently not used. It must be a 64-byte null token.

generated_key_identifier
Direction: Input/Output
Type: String

The internal token of an operational key, a skeleton token containing the control vector of the key to be generated, or a null token. A null token can be supplied if the generated_key_identifier will be a DKYGENKY with a CV derived from the generating_key_identifier. A skeleton token or internal token is required when generated_key_identifier will not be a DKYGENKY key type or the processing method is not SESS-XOR. For SESS-XOR, this must be a null token. On output, this parameter contains the generated key.

Restrictions
This callable service does not support version X’10’ external DES key tokens (RXX key tokens).

Usage Notes
SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Refer to Appendix C, “Control Vectors and Changing Control Vectors with the CVT Callable Service,” on page 633 for information on the control vector bits for the DKG key generating key.

Additional usage notes for Session key algorithm (EMV Smartcard specific):

Given an MDK, it can be used in two ways:

- To calculate the Card Specific Key (or UDK) in the personalization process, call CSNBBDKG with the TDES-ENC method using an output token that has been
Diversified Key Generate (CSNBDKG)

primed with the CV of the final session key, for instance, if the MDK is a DMPIN, the token should have the CV of an SMPIN key; DMAC; a double length MAC; DDATA, a double length DATA key, etc.

The result would then be exported in the personalization file. This key is not usable in this form for any other calculations.

- To use the session key, Call CSNBDKG with the TDESEMV4 method. Provide, for input, the same card data that was used to create the UDK as well as the ATC and optionally the IV value. This is the key that will be used in EMV related Smartcard processing.

This same processing applies to those API's the generate the session key on your behalf, like CSNBPCU.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 19. Diversified key generate required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>PCI Cryptographic Coprocessor</td>
<td>Keywords TDES-XOR, TDESEMV2 and TDESEMV4 are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>PCI X Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Cryptographic Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

Key Export (CSNBKEX)

Use the key export callable service to reencipher any type of key (except an AKEK or an IMP-PKA) from encryption under a master key variant to encryption under the same variant of an exporter key-encrypting key. The reenciphered key can be exported to another system.

If the key to be exported is a DATA key, the key export service generates a key token with the same key length as the input token's key.

This service supports the no-export bit that the prohibit export service sets in the internal token.
Key Export (CSNBKEX)

Format

```
CALL CSNBKEX(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  key_type,
  source_key_identifier,
  exporter_key_identifier,
  target_key_identifier )
```

Parameters

**return_code**

Direction: Output Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**

Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**

Direction: Input/Output Type: String

The data that is passed to the installation exit.

**key_type**

Direction: Input Type: Character string

The parameter is an 8-byte field that contains either a key type value or the keyword TOKEN. The keyword is left-justified and padded on the right with blanks.

If the key type is TOKEN, ICSF determines the key type from the control vector (CV) field in the internal key token provided in the source_key_identifier parameter. If the control vector is invalid on the Cryptographic Coprocessor Feature, the key export request will be routed to the PCI Cryptographic Coprocessor.
Key Export (CSNBKEX)

Key type values for the Key Export callable service are: CIPHER, DATA, DATAC, DATAM, DATAMV, DATAXLAT, DECIPHER, ENCIPHER, EXPORTER, IKEYXLAT, IMPORTER, IPINENC, MAC, MACD, MACVER, OKEYXLAT, OPINENC, PINGEN and PINVER. For information on the meaning of the key types, see Table 2 on page 21.

**source_key_identifier**

Direction: Input
Type: String

A 64-byte string of the internal key token that contains the key to be reenciphered. This parameter must identify an internal key token in application storage, or a label of an existing key in the cryptographic key data set.

If you supply TOKEN for the `key_type` parameter, ICSF looks at the control vector in the internal key token and determines the key type from this information. If you supply TOKEN for the `key_type` parameter and supply a label for this parameter, the label must be unique in the cryptographic key data set.

**exporter_key_identifier**

Direction: Input/Output
Type: String

A 64-byte string of the internal key token or key label that contains the exporter key-encrypting key. This parameter must identify an internal key token in application storage, or a label of an existing key in the cryptographic key data set.

If the NOCV bit is on in the internal key token containing the key-encrypting key, the key-encrypting key itself (not the key-encrypting key variant) is used to encipher the generated key. For example, the key has been installed in the cryptographic key data set through the key generator utility program or the key entry hardware using the NOCV parameter, or you are passing the key-encrypting key in the internal key token with the NOCV bit on and your program is running in supervisor state or in key 0-7.

Control vectors are explained in "Control Vector" on page 18 and the NOCV bit is shown in Table 241 on page 598.

**target_key_identifier**

Direction: Input/Output
Type: String

The 64-byte field external key token that contains the reenciphered key. The reenciphered key can be exchanged with another cryptographic system.

**Restrictions**

For existing TKE V3.1 (or higher) users, you may have to explicitly enable new access control points. Current applications will fail if they use an equal key halves exporter to export a key with unequal key halves. You must have access control point 'Key Export - Unrestricted' explicitly enabled.

This service cannot be used to export AKEKs. Refer to "ANSI X9.17 Key Export (CSNAKEK)" on page 473 for information on exporting AKEKs.

This callable service does not support version X‘10’ external DES key tokens (RKX key tokens).
Key Export (CSNBKEX)

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

For key export, you can use these combinations of parameters:

- A valid key type in the `key_type` parameter and an internal key token in the `source_key_identifier` parameter. The key type must be equivalent to the control vector specified in the internal key token.
- A `key_type` parameter of TOKEN and an internal key token in the `source_key_identifier` parameter. The `source_key_identifier` can be a label with TOKEN only if the labelname is unique on the CKDS. The key type is extracted from the control vector contained in the internal key token.
- A valid key type in the `key_type` parameter, and a label in the `source_key_identifier` parameter.

If internal key tokens are supplied in the `source_key_identifier` or `exporter_key_identifier` parameters, the key in one or both tokens can be reenciphered. This occurs if the master key was changed since the internal key token was last used. The return and reason codes that indicate this do not indicate which key was reenciphered. Therefore, assume both keys have been reenciphered.

**Systems with the Cryptographic Coprocessor Feature.**

ICSF examines the data encryption algorithm bits on the exporter key-encrypting key and the key being exported for consistency. It does not export a CDMF key under a DES-marked key-encrypting key or a DES key under a CDMF-marked key-encrypting key. ICSF does not propagate the data encryption marking on the operational key to the external token.

If the key type is MACD, the control vectors of the input keys must be the standard control vectors supported by the Cryptographic Coprocessor Feature, since the key export service will be processed on the Cryptographic Coprocessor Feature in this case.

To use NOCV key-encrypting keys or to export double-length DATAM and DATAMV keys, the NOCV-enablement keys must be installed in the CKDS. NOCV-enablement keys are only needed with the Cryptographic Coprocessor Feature.

For a double-length MAC key with a key type of DATAM, the service uses the data compatibility control vector to create an external token. For a double-length MAC key with a key type of MACD, the service uses the single-length control vector for both the left and right half of the key to create an external token (MAC||MAC). For a table of control vectors, refer to Control Vector Table.

Key Export operations which specify a NOCV key-encrypting key as the exporter key and also specify a source or key-encrypting key which contains a control vector not supported by the Cryptographic Coprocessor Feature will fail.

To export a double-length MAC generation or MAC verification key, it is recommended that a key type of TOKEN be used.
Key Export (CSNBKEX)

Systems with a PCI X Cryptographic Coprocessor, Crypto Express2 Coprocessor, or Crypto Express3 Coprocessor

If running with a PCIXCC, CEX2C, or CEX3C, existing internal tokens created with key type MACD must be exported with either a TOKEN or DATAM key type. The external CV will be DATAM CV. The MACD key type is not supported.

To export a double-length MAC generation or MAC verification key, it is recommended that a key type of TOKEN be used.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 20. Key export required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>Key_type MACD is processed on a Cryptographic Coprocessor Feature. DATAC key type is not supported.</td>
</tr>
</tbody>
</table>
| IBM @server zSeries 900 | PCI Cryptographic Coprocessor | ICSF routes the request to a PCI Cryptographic Coprocessor if:  
  - The key_type specified is one of these: DECIPHER, ENCRYPT, IKEYXLAT, OKEYXLAT or CIPHER.  
  - The control vector of either the exporter_key_identifier or the source_key_identifier cannot be processed on the Cryptographic Coprocessor Feature.  
  - Token markings for DES/CDMF on DATA and KEKs are ignored. |
| IBM @server zSeries 990 | PCI X Cryptographic Coprocessor | Key_type MACD and DATA XLAT are not supported. Token markings for DES/CDMF on DATA and KEKs are ignored. |
| IBM @server zSeries 890 | Crypto Express2 Coprocessor | Key_type MACD and DATA XLAT are not supported. Token markings for DES/CDMF on DATA and KEKs are ignored. |
| IBM System z9 EC | Crypto Express2 Coprocessor | Key_type MACD and DATA XLAT are not supported. Token markings for DES/CDMF on DATA and KEKs are ignored. |
| IBM System z9 BC | Crypto Express2 Coprocessor | Key_type MACD and DATA XLAT are not supported. Token markings for DES/CDMF on DATA and KEKs are ignored. |
| IBM System z10 EC | Crypto Express2 Coprocessor | Key_type MACD and DATA XLAT are not supported. Token markings for DES/CDMF on DATA and KEKs are ignored. |
| IBM System z10 BC | Crypto Express3 Coprocessor | Key_type MACD and DATA XLAT are not supported. Token markings for DES/CDMF on DATA and KEKs are ignored. |

Key Generate (CSNBKGN and CSNEKGN)

Use the key generate callable service to generate either one or two odd parity DES keys of any type. The keys can be single-length (8 bytes), double-length (16 bytes), or, in the case of DATA keys, triple-length (24 bytes). The callable service does not produce keys in clear form and all keys are returned in encrypted form. When two keys are generated, each key has the same clear value, although this clear value is not exposed outside the secure cryptographic feature.
Key Generate (CSNBKGN and CSNEKGN)

Use the key generate callable service to generate an AES key of DATA type. The callable service does not produce AES keys in clear form and all AES keys are returned in encrypted form. Only one AES key is generated. Its clear value is not exposed outside the secure cryptographic feature.

This callable service supports invocation in AMODE (64). The callable service name for AMODE (64) invocation is CSNEKGN.

Format

```
CALL CSNBKGN(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_form,
    key_length,
    key_type_1,
    key_type_2,
    KEK_key_identifier_1,
    KEK_key_identifier_2,
    generated_key_identifier_1,
    generated_key_identifier_2 )
```

Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. [Appendix A, "ICSF and TSS Return and Reason Codes"] lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. [Appendix A, "ICSF and TSS Return and Reason Codes"] lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**key_form**

Direction: Input  
Type: Character string

The data that is passed to the installation exit.
A 4-byte keyword that defines the type of key(s) you want to generate. This parameter also specifies if each key should be returned for either operational, importable, or exportable use. The keyword must be in a 4-byte field, left-justified, and padded with blanks.

The first two characters refer to `key_type_1`. The next two characters refer to `key_type_2`.

These keywords are allowed: OP, IM, EX, OPIM, OPEX, IMEX, EXEX, OPOP, and IMIM. See Table 21 for their meanings.

If the `key_form` is OP, EX or IM, the KEK_key_identifier_2, generated_key_identifier_1 and generated_key_identifier_2 should be set to NULL.

**Table 21. Key Form Values for the Key Generate Callable Service**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EX</td>
<td>One key that can be sent to another system.</td>
</tr>
<tr>
<td>EXEX</td>
<td>A key pair; both keys to be sent elsewhere, possibly for exporting to two different systems. The key pair has the same clear value.</td>
</tr>
<tr>
<td>IM</td>
<td>One key that can be locally imported. The key can be imported onto this system to make it operational at another time.</td>
</tr>
<tr>
<td>IMEX</td>
<td>A key pair to be imported; one key to be imported locally and one key to be sent elsewhere. Both keys have the same clear value.</td>
</tr>
<tr>
<td>IMIM</td>
<td>A key pair to be imported; both keys to be imported locally at another time.</td>
</tr>
<tr>
<td>OP</td>
<td>One operational key. The key is returned to the caller in the key token format. Specify the OP key form when generating AKEKs and AES keys.</td>
</tr>
<tr>
<td>OPEX</td>
<td>A key pair; one key that is operational and one key to be sent from this system. Both keys have the same clear value.</td>
</tr>
<tr>
<td>OPIM</td>
<td>A key pair; one key that is operational and one key to be imported to the local system. Both keys have the same clear value. On the other system, the external key token can be imported to make it operational.</td>
</tr>
<tr>
<td>OPOP</td>
<td>A key pair; normally with different control vector values.</td>
</tr>
</tbody>
</table>

The key forms are defined as follows:

**Operational (OP)**

The key value is enciphered under a master key. The result is placed into an internal key token. The key is then operational at the local system. For AKEKs, the result is placed in a skeleton token created by the key token build callable service. AES AKEKs are not supported.

**Importable (IM)**

The key value is enciphered under an importer key-encrypting key. The result is placed into an external key token.

**Exportable (EX)**

The key value is enciphered under an exporter key-encrypting key. The result is placed into an external key token. The key can then be...
Key Generate (CSNBKGN and CSNEKGN)

transported or exported to another system and imported there for use. This key form cannot be used by any ICSF callable service.

The keys are placed into tokens that the generated_key_identifier_1 and generated_key_identifier_2 parameters identify.

Valid key type combinations depend on the key form. See Table 27 for valid key combinations.

key_length

Direction: Input Type: Character string

An 8-byte value that defines the length of the key. The keyword must be left-justified and padded on the right with blanks. You must supply one of the key length values in the key_length parameter.

Table 22. Key Length Values for the Key Generate Callable Service

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>SINGLE, SINGLE-R or KEYLN8</td>
<td>The key should be a single length (8-byte or 64-bit) key.</td>
<td>DES</td>
</tr>
<tr>
<td>DOUBLE or KEYLN16</td>
<td>The key should be a double length (16-byte or 128-bit) key</td>
<td>AES or DES</td>
</tr>
<tr>
<td>KEYLN24</td>
<td>The key should be a 24-byte (192-bit) key.</td>
<td>AES or DES</td>
</tr>
<tr>
<td>KEYLN32</td>
<td>The key should be a 32-byte (256-bit) key.</td>
<td>AES</td>
</tr>
</tbody>
</table>

DES Keys:

Double-length (16-byte) keys have an 8-byte left half and an 8-byte right half. Both halves can have identical clear values or not. If you want the same value to be used in both key halves (referred to as replicated key values), specify key_length as SINGLE, SINGLE-R or KEYLN8. If you want different values to be the basis of each key half, specify key_length as DOUBLE or KEYLN16.

Triple-length (24-byte) keys have three 8-byte key parts. This key length is valid for DATA keys only. To generate a triple-length DATA key with three different values to be the basis of each key part, specify key_length as KEYLN24.

Use SINGLE/SINGLE-R if you want to create a DES transport key that you would use to exchange DATA keys with a PCF system. Because PCF does not use double-length transport keys, specify SINGLE so that the effects of multiple encipherment are nullified. When generating an AKEK, the key_length parameter is ignored. The AKEK key length (8-byte or 16-byte) is determined by the skeleton token created by the key token build callable service and provided in the generated_key_identifier_1 parameter.

Note: SINGLE-R is not supported on IBM @server zSeries 800 or IBM @server zSeries 900 servers.

AES Keys:

AES only allows KEYLN16, KEYLN24, KEYLN32. To generate a 128-bit AES key, specify key_length as KEYLN16. For 192-bit AES keys specify key_length as KEYLN24. A 256-bit AES key requires a key_length of KEYLN32. All AES keys are DATA keys.
Key Generate (CSNBKGN and CSNEKGN)

Systems with CCFs (with or without PCICCs)

This table shows the valid key lengths for each key type supported by DES keys. An X indicates that a key length is permitted for a key type. A Y indicates that the key generated will be a double-length key with replicated key values.

**Note:** When generating a double-length key with replicated key values and the key_form parameter as IMEX, the KEK_key_identifier_1 parameter must contain a NOCV IMPORTER key-encrypting key either as a key label or an internal key token. Also the CKDS must contain NOCV enablement keys.

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Single - KEYLN8</th>
<th>Double - KEYLN16</th>
<th>KEYLN24</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACVER</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DATAM</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DATAMV</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXPORTER</td>
<td>Y</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>IMPORTER</td>
<td>Y</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>IKEYXLAT</td>
<td>Y</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>OKEYXLAT</td>
<td>Y</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CIPHER#</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECIPHER#</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENCIIPHER#</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPINENC</td>
<td>Y</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>OPINENC</td>
<td>Y</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PINGEN</td>
<td>Y</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PINVER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARDEC*#</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CVARENC*#</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CVARPINE*#</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CVARXCVL*#</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CVARXCVR*#</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DKYGENKY*#</td>
<td>Y</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>KEYGENKY*#</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
1. * — key types marked with an asterisk (*) are requested through the use of the TOKEN keyword and specifying a proper control-vector in a key token
2. # — key types marked with a pound sign (#) require a PCICC

Systems with PCIXCCs/CEX2C/CEX3C

This table shows the valid key lengths for each key type supported by DES keys. An X indicates that a key length is permitted for a key type. A Y indicates that the key generated will be a double-length key with replicated key values. It is preferred that SINGLE-R be used for this result.

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Single - KEYLN8</th>
<th>Single-R KEYLN16</th>
<th>Double - KEYLN16</th>
<th>KEYLN24</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACVER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATAM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATAMV</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXPORTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMPORTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IKEYXLAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OKEYXLAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CIPHER#</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECIPHER#</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENCIIPHER#</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPINENC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPINENC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PINGEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PINVER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARDEC*#</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARENC*#</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARPINE*#</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARXCVL*#</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARXCVR*#</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DKYGENKY*#</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEYGENKY*#</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Key Generate (CSNBKGN and CSNEKGN)

**Table 24. Key lengths for DES keys - PCIXCC/CEX2C/CEX3C systems (continued)**

|             | MAC | MACVER | DATA | DATAC* | DATAM | DATAMV | EXPORTER | IMPORTER | IKEYXLAT | OKEYXLAT | CIPHER | DECIPHER | ENCIPHER | IPINENC | OPINENC | PINGEN | PINVER | CVARDEC* | CVARENC* | CVARPINE* | CVARXCVL* | CVARXCVR* | DKYGENKY* | KEYGENKY* |
|-------------|-----|--------|------|--------|-------|--------|----------|----------|----------|----------|--------|----------|----------|---------|---------|--------|--------|---------|---------|----------|---------|----------|---------|
|             | X   | X      | X    | X      | X     |        | X        | X        | X        | X        | X      | X        | X        | Y       | Y       | Y      | Y      | X       | X       | X        | X       | X        | X        |

This table shows the valid key lengths for each key type supported by AES keys. An X indicates that a key length is permitted for that key type.

**Table 25. Key lengths for AES keys - CEX2C/CEX3C systems**

<table>
<thead>
<tr>
<th>Key Type</th>
<th>128-byte</th>
<th>192-byte</th>
<th>256-byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>AESTOKEN</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>AESDATA</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**key_type_1**

Direction: Input  
Type: Character string

Use the `key_type_1` parameter for the first, or only key, that you want generated. The keyword must be left-justified and padded with blanks. Valid type combinations depend on the key form.

The 8-byte keyword for the `key_type_1` parameter can be one of the following:

- AESDATA, AESTOKEN, CIPHER, DATA, DATAC, DATAM, DATAMV, DATAXLAT, DECIPHER, ENCIPHER, EXPORTER, IKEYXLAT, IMPORTER, IPINENC, MAC, MACVER, OKEYXLAT, OPINENC, PINGEN and PINVER
- or the keyword TOKEN

For information on the meaning of the key types, see Table 2 on page 21.

If `key_type_1` is TOKEN, ICSF examines the control vector (CV) field in the `generated_key_identifier_1` parameter to derive the key type. When `key_type_1`
Key Generate (CSNBKGN and CSNEKGN)

is TOKEN, ICSF does not check for the length of the key for DATA keys. Instead, ICSF uses the key_length parameter to determine the length of the key.

To generate a DES AKEK, specify a key_type_1 of TOKEN. The generated_key_identifier_1 parameter must be a skeleton token of an AKEK created by the key token build (CSNBKTB) callable service. The token cannot be a partially notarized AKEK or an AKEK key part.

If key_type_1 is AESDATA or AES TOKEN, the key generated will be an AES key of type DATA. When key_type_1 is AES TOKEN, ICSF uses the key_length parameter to determine the length of the key.

See Table 26 and Table 27 for valid key type and key form combinations.

key_type_2
Direction: Input
Type: Character string

Use the key_type_2 parameter for a key pair, which is shown in Table 27 on page 120. The keyword must be left-justified and padded with blanks. Valid type combinations depend on the key form. key_type_2 is only used when DES keys are generated.

key_type_2 is only use when DES keys are generated. The 8-byte keyword for the key_type_2 parameter can be one of the following:

- CIPHER, DATA, DATAC, DATAM, DATAMV, DATAXLAT, DECIPHER, ENCIPHER, EXPORTER, IKEYXLAT, IMPORTER, IPINENC, MAC, MACVER, OKEYXLAT, OPINENC, PINGEN and PINVER
- or the keyword TOKEN

For information on the meaning of the key types, see Description of Key Types, Table 2 on page 21.

If key_type_2 is TOKEN, ICSF examines the control vector (CV) field in the generated_key_identifier_2 parameter to derive the key type. When key_type_2 is TOKEN, ICSF does not check for the length of the key for DATA keys. Instead, ICSF uses the key_length parameter to determine the length of the key.

If only one key is to be generated, key_type_2 and KEK_key_identifier_2 are ignored.

See Table 26 on page 120 and Table 27 on page 120 for valid key type and key form combinations.

KEK_key_identifier_1
Direction: Input/Output
Type: String

A 64-byte string of a DES internal key token containing the importer or exporter key-encrypting key, or a key label. If you supply a key label that is less than 64-bytes, it must be left-justified and padded with blanks. KEK_key_identifier_1 is required for a key_form of IM, EX, IMEX, EXEX, or IMIM.

When key_form OP is used, parameters KEK_key_identifier_1 and KEK_key_identifier_2 are ignored. In this case, it is recommended that the parameters are initialized to 64-bytes of X’00’.

If the NOCV bit is on in the internal key token containing the key-encrypting key, the key-encrypting key itself (not the key-encrypting key variant) is used to
encipher the generated key. For example, the key has been installed in the cryptographic key data set through the key generator utility program or the key entry hardware using the NOCV parameter; or you are passing the key-encrypting key in the internal key token with the NOCV bit on and your program is running in supervisor state or key 0-7.

Control vectors are explained in “Control Vector” on page 18 and the NOCV bit is shown in Table 241 on page 598.

KEK_key_identifier_1 cannot be an AES key token or label.

**KEK_key_identifier_2**

Direction: Input/Output

Type: String

A 64-byte string of a DES internal key token containing the importer or exporter key-encrypting key, or a key label of an internal token. If you supply a key label that is less than 64-bytes, it must be left-justified and padded with blanks. *KEK_key_identifier_2* is required for a key_form of OPIM, OPEX, IMEX, IMIM, or EXEX. This field is ignored for key_form keywords OP, IM and EX. When key_form OP is used, parameter *KEK_key_identifier_2* is ignored. In this case, it is recommended that the parameter is initialized to 64-bytes of X’00’.

If the NOCV bit is on in the internal key token containing the key-encrypting key, the key-encrypting key itself (not the key-encrypting key variant) is used to encipher the generated key. For example, the key has been installed in the cryptographic key data set through the key generator utility program or the key entry hardware using the NOCV parameter; or you are passing the key-encrypting key in the internal key token with the NOCV bit on and your program is running in supervisor state or in key 0-7.

Control vectors are explained in “Control Vector” on page 18 and the NOCV bit is shown in Table 241 on page 598.

KEK_key_identifier_2 cannot be an AES key token or label.

**generated_key_identifier_1**

Direction: Input/Output

Type: String

This parameter specifies either a generated:

- Internal DES or AES key token for an operational key form, or
- External DES key tokens containing a key enciphered under the *KEK_key_identifier_1* parameter.

If you specify a key_type_1 of TOKEN, then this field contains a valid DES token of the key type you want to generate. Otherwise, on input, this parameter must be binary zeros. See key_type_1 for a list of valid key types.

If you specify a key_type_1 of IMPORTER or EXPORTER and a key_form of OPEX, and if the generated_key_identifier_1 parameter contains a valid DES internal token of the SAME type, the NOCV bit, if on, is propagated to the generated key token.

When generating an AKEK, specify the skeleton key token created by the key token build callable service (CSNBKTB) as input for this parameter.

When key_type_1 parameter is AESDATA, then generated_key_identifier_1 is ignored. In this case, it is recommended that the parameter be initialized to 64-bytes of X’00’. If you specify a key_type_1 of AESTOKEN, the
**Key Generate (CSNBKGN and CSNEKGN)**

The `generated_key_identifier_2` parameter must be an internal AES key token or a clear AES key token. Information in this token can be used to determine the key type:

- The key type parameter overrides the type in the token.
- The `key_length` parameter overrides the length value in the generated key token.

`generated_key_identifier_2`

**Direction:** Input/Output  
**Type:** String

This parameter specifies a generated external key token containing a key enciphered under the `KEK_key_identifier_2` parameter.

When `key_type_1` parameter is AESDATA or AESTOKEN, then `generated_key_identifier_2` is ignored. In this case, it is recommended that the parameters are initialized to 64-bytes of X'00'.

If you specify a `key_type_2` of TOKEN, then this field contains a valid token of the key type you want to generate. Otherwise, on input, this parameter must be binary zeros. See `key_type_1` for a list of valid key types.

The token can be an internal or external token.

**Restrictions**

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

**System Encryption Algorithm Marks (CCF systems only)**

This applies to requests processed on a system with CCFs and only if the request is processed by the CCF. Processing on a PCICC does not cause tokens to be marked.

Internal DATA, IMPORTER and EXPORTER tokens are marked with the system encryption algorithm. No external tokens generated by this service are marked.

When the key form is OP, the token is marked with the system default algorithm. This marking can be overridden by specifying a valid token in the `generated_key_identifier_1` parameter with the marking required.

When the key form is OPEX or OPIM, the operational token is marked with the markings of the key-encrypting key (`KEK_key_identifier_2`). This marking can be overridden by specifying a valid token in the `generated_key_identifier_1` parameter with the marking required.

It is possible to generate an operational DES-marked DATA key on a CDMF-only system or a CDMF-marked DATA key on a DES-only system. However, the encipher (CSNBENC) and decipher (CSNBDDEC) callable services fail when you use these keys on the systems where they were generated unless overridden by keyword.
Key Generate (CSNBKGN and CSNEKGN)

**Key type and key form combinations**

Table 26 shows the valid key type and key form combinations for a single DES or AES key. Key types marked with an "*" must be requested through the specification of a proper control vector in a key token and through the use of the TOKEN keyword.

**Note:** Not all keytypes are valid on all hardware. See Table 2 on page 21.

*Table 26. Key Generate Valid Key Types and Key Forms for a Single Key*

<table>
<thead>
<tr>
<th>Key Type 1</th>
<th>Key Type 2</th>
<th>OP</th>
<th>IM</th>
<th>EX</th>
</tr>
</thead>
<tbody>
<tr>
<td>AESDATA</td>
<td>Not applicable</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AESTOKEN</td>
<td>Not applicable</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>Not applicable</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DATA*</td>
<td>Not applicable</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DATAM</td>
<td>Not applicable</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DKYGENKY*</td>
<td>Not applicable</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>KEYGENKY*</td>
<td>Not applicable</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>MAC</td>
<td>Not applicable</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PINGEN</td>
<td>Not applicable</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 27 shows the valid key type and key form combinations for a DES key pair. Key types marked with an "*" must be requested through the specification of a proper control vector in a key token and through the use of the TOKEN keyword.

*Table 27. Key Generate Valid Key Types and Key Forms for a Key Pair*

<table>
<thead>
<tr>
<th>Key Type 1</th>
<th>Key Type 2</th>
<th>OPEX</th>
<th>EXEX</th>
<th>OPIM, OPOP, IMIM</th>
<th>IMEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIPHER</td>
<td>CIPHER</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CIPHER</td>
<td>DECIPHER</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CIPHER</td>
<td>ENCPHER</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CVARDEC*</td>
<td>CVARENC*</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARDEC*</td>
<td>CVARPCIN*</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARENC*</td>
<td>CVARDEC*</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARENC*</td>
<td>CVARPCIN*</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARENC*</td>
<td>CVARXCVL*</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARENC*</td>
<td>CVARXCVR*</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARXCVL*</td>
<td>CVARENC*</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARXCVR*</td>
<td>CVARENC*</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARPINE*</td>
<td>CVARDEC*</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>DATA</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>DATAXLAT</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA*</td>
<td>DATAXLAT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DATAM</td>
<td>DATAM</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DATAM</td>
<td>DATAMV</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DATAXLAT</td>
<td>DATAXLAT</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECPHER</td>
<td>DECPHER</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

120 z/OS V1R11 ICSF Application Programmer's Guide
### Table 27. Key Generate Valid Key Types and Key Forms for a Key Pair (continued)

<table>
<thead>
<tr>
<th>Key Type 1</th>
<th>Key Type 2</th>
<th>OPEX</th>
<th>EXEX</th>
<th>OPIM, OPOP, IMIM</th>
<th>IMEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECIPHER</td>
<td>ENCIPHER</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DKYGENKY*</td>
<td>DKYGENKY*</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ENCIPHER</td>
<td>CIPHER</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ENCIPHER</td>
<td>DECIPHER</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>EXPORTER</td>
<td>IKEYXLAT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>EXPORTER</td>
<td>IMPORTER</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>IKEYXLAT</td>
<td>EXPORTER</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>IKEYXLAT</td>
<td>OKEYXLAT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>IMPORTER</td>
<td>EXPORTER</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>IMPORTER</td>
<td>OKEYXLAT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>IPINENC</td>
<td>OPINENC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>KEYGENKY*</td>
<td>KEYGENKY*</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>MAC</td>
<td>MAC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>MAC</td>
<td>MACVER</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>OKEYXLAT</td>
<td>IKEYXLAT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>OKEYXLAT</td>
<td>IMPORTER</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>OPINENC</td>
<td>IPINENC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>OPINENC</td>
<td>OPINENC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PINVER</td>
<td>PINGEN</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PINGEN</td>
<td>PINVER</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

If you are running with the Cryptographic Coprocessor Feature and the `key_form` is IMEX, the `key_length` is SINGLE, and `key_type_1` is IPINENC, OPINENC, PINGEN, IMPORTER, or EXPORTER, you must specify the `KEK_key_identifier_1` parameter as NOCV IMPORTER.

If you are running with the Cryptographic Coprocessor Feature and need to use NOCV key-encrypting keys, NOCV-enablement keys must be installed in the CKDS. If you running with the PCI X Cryptographic Coprocessor, Crypto Express2 Coprocessor, or Crypto Express3 Coprocessor and need to use NOCV key-encrypting keys, you need to enable NOCV IMPORTER and NOCV EXPORTER access control points.

If you are running with the Cryptographic Coprocessor Feature and need to generate DATAM and DATAMV keys in the importable form, the ANSI system keys must be installed in the CKDS.

Table 28 on page 122 lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
### Key Generate (CSNBKGN and CSNEKGN)

Table 28. Key generate required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>OPIM is valid on the Cryptographic Coprocessor Feature for key forms DATA/DATA, DATAM/DATAM and MAC/MAC. All other OPIM key forms are routed to the PCI Cryptographic Coprocessor. In <code>key_form</code> and <code>generated_key_identifier_1</code>, marking of data encryption algorithm bits and token copying are only performed if this service is processed on a Cryptographic Coprocessor Feature. In <code>KEK_key_identifier_2</code>, propagation of token markings is only relevant when this service is processed on the Cryptographic Coprocessor Feature. In <code>generated_key_identifier_1</code>, propagation of the NOCV bit is performed only if the service is processed on the Cryptographic Coprocessor Feature. AKEKs are processed on CCFs DATAC is not supported. Secure AES keys are not supported.</td>
</tr>
</tbody>
</table>
| IBM @server zSeries 900       | PCI Cryptographic Coprocessor                        | ICSF routes the request to a PCI Cryptographic Coprocessor if:  
  - OPIM key forms are not DATA/DATA, DATAM/DATAM or MAC/MAC.  
  - The key type specified in `key_type_1` or `key_type_2` is not valid for the Cryptographic Coprocessor Feature or if the control vector in a supplied token cannot be processed on the Cryptographic Coprocessor Feature.  
  - A key length of SINGLE-R is specified, or if a key form of OPIM, OPOP or IMIM is specified.  
  - Tokens are not marked with the system encryption algorithm. The NOCV flag is not propagated to key-encrypting keys. Secure AES keys are not supported. |
| IBM System z9 EC              | PCI X Cryptographic Coprocessor                      | `Key_type` DATAXLAT is not supported. AKEK key type is not supported. Secure AES keys are not supported. |
| IBM System z9 BC              | Crypto Express2 Coprocessor                         | `Key_type` DATAXLAT is not supported. AKEK key type is not supported. Secure AES keys require the Nov. 2008 or later licensed internal code (LIC). |
Key Generate (CSNBKGN and CSNEKGN)

Table 28. Key generate required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td><em>Key_type</em> DATAXLAT is not supported.</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td>AKEK key type is not supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Secure AES keys require the Nov. 2008 or later licensed internal code (LIC).</td>
</tr>
</tbody>
</table>

Key Import (CSNBKIM)

Use the key import callable service to reencipher a key (except an AKEK) from encryption under an importer key-encrypting key to encryption under the master key. The reenciphered key is in operational form.

Choose one of these options:
- Specify the *key_type* parameter as TOKEN and specify the external key token in the *source_key_identifier* parameter. The key type information is determined from the control vector in the external key token.
- Specify a key type in the *key_type* parameter and specify an external key token in the *source_key_identifier* parameter. The specified key type must be compatible with the control vector in the external key token.
- Specify a valid key type in the *key_type* parameter and a null key token in the *source_key_identifier* parameter. The default control vector for the *key_type* specified will be used to process the key.

For DATA keys, this service generates a key of the same length as that contained in the input token.

Format

```
CALL CSNBKIM(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_type,
    source_key_identifier,
    importer_key_identifier,
    target_key_identifier
)
```

Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. [Appendix A, "ICSF and TSS Return and Reason Codes"] lists the return codes.

**reason_code**

Direction: Output  
Type: Integer
Key Import (CSNBKIM)

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

exit_data_length
Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

exit_data
Direction: Input/Output Type: String

The data that is passed to the installation exit.

key_type
Direction: Input Type: Character string

The type of key you want to reencipher under the master key. Specify an 8-byte keyword or the keyword TOKEN. The keyword must be left-justified and padded on the right with blanks.

If the key type is TOKEN, ICSF determines the key type from the control vector (CV) field in the external key token provided in the source_key_identifier parameter.

TOKEN is never allowed when the importer_key_identifier is NOCV.

Supported key_type values are CIPHER, DATA, DATAM, DATAMV, DATAXLAT, DECIPHER, ENCIPHER, EXPORTER, IKEYXLAT, IMPORTER, IPINENC, MAC, MACVER, OKEYXLAT, OPINENC, PINGEN and PINVER. Use key_type TOKEN for all other key types.

For information on the meaning of the key types, see Table 2 on page 21. We recommend using key type of TOKEN when importing double-length MAC and MACVER keys.

source_key_identifier
Direction: Input Type: String

The key you want to reencipher under the master key. The parameter is a 64-byte field for the enciphered key to be imported containing either an external key token or a null key token. If you specify a null token, the token is all binary zeros, except for a key in bytes 16-23 or 16-31, or in bytes 16-31 and 48-55 for triple-length DATA keys. Refer to Table 244 on page 602.

If key type is TOKEN, this field may not specify a null token.

This service supports the no-export function in the CV.

importer_key_identifier
Direction: Input/Output Type: String
Key Import (CSNBKIM)

The importer key-encrypting key that the key is currently encrypted under. The parameter is a 64-byte area containing either the key label of the key in the cryptographic key data set or the internal key token for the key. If you supply a key label that is less than 64-bytes, it must be left-justified and padded with blanks.

**Note:** If you specify a NOCV importer in the `importer_key_identifier` parameter, the key to be imported must be enciphered under the importer key itself.

**target_key_identifier**

**Direction:** Input/Output  
**Type:** String

This parameter is the generated reenciphered key. The parameter is a 64-byte area that receives the internal key token for the imported key.

If the imported key TYPE is IMPORTER or EXPORTER and the token key TYPE is the same, the `target_key_identifier` parameter changes direction to both input and output. If the application passes a valid internal key token for an IMPORTER or EXPORTER key in this parameter, the NOCV bit is propagated to the imported key token.

**Note:** Propagation of the NOCV bit is performed only if the service is processed on a Cryptographic Coprocessor Feature or on a PCIXCC, CEX2C, or CEX3C.

**Restrictions**

For existing TKE V3.1 (or higher) users, you may have to explicitly enable new access control points. Current applications will fail if they use an equal key halves importer to import a key with unequal key halves. You must have access control point 'Key Import - Unrestricted' explicitly enabled.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Use of NOCV keys are controlled by an access control point in the PCIXCC. Creation of NOCV key-encrypting keys is only available for standard IMPORTERs and EXPORTERs.

**Systems with the Cryptographic Coprocessor Feature**

The key import callable service cannot be used to import ANSI key-encrypting keys. For information on importing these types of keys, refer to "ANSI X9.17 Key Import (CSNAKIM)" on page 478. To use NOCV key-encrypting keys or to import DATAM or DATAMV keys, NOCV-enablement keys must be installed in the CKDS.

This service will marked an imported KEK as a NOCV-KEK KEK by suppling a valid IMPORTER or EXPORTER token in the target_key_identifier field with the NOCV-KEK flag enabled. The type of the token must match the key type of the imported key.
Key Import (CSNBKIM)

This service will mark DATA and key-encrypting key tokens with the system encryption algorithm if the request is processed on the CCF. The service propagates the data encryption algorithm mark on the operational KEK unless token copying overrides this:

- The imported token is marked with the DES or CDMF encryption algorithm marks of the KEK token
- The imported token is marked with the system’s default encryption algorithm when the KEK is marked SYS-ENC
- To override the encryption algorithm marks of the KEK, supply a valid token in the target_key_identifier field of the same key type being imported. The mark of the target_key_identifier token are used to mark the imported key token.

Key Import operations which specify a NOCV key-encrypting key as either the importer key or the target and also specify a source or key-encrypting key which contains a control vector not supported by the Cryptographic Coprocessor Feature will fail.

Systems with the PCI X Cryptographic Coprocessor, Crypto Express2 Coprocessor, or Crypto Express3 Coprocessor

Use of NOCV keys are controlled by an access control point in the PCIXCC, CEX2C, or CEX3C.

This service will marked an imported KEK as a NOCV-KEK KEK:

- If a token is supplied in the target token field, it must be a valid importer or exporter token. If the token fails token validation, processing continues, but the NOCV flag will not be copied
- The source token (key to be imported) must be a importer or exporter with the default control vector.
- If the target token is valid and the NOCV flag is on and the source token is valid and the control vector of the target token is exactly the same as the source token, the imported token will have the NOCV flag set on.
- If the target token is valid and the NOCV flag is on and the source token is valid and the control vector of the target token is NOT exactly the same as the source token, a return code will be given.
- All other scenarios will complete successfully, but the NOCV flag will not be copied

The software bit used to mark the imported token with export prohibited is not supported on a PCIXCC, CEX2C, or CEX3C. The internal token for an export prohibited key will have the appropriate control vector that prohibits export.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
Key Import (CSNBKIM)

Table 29. Key import required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>Propagation of token markings is only relevant when this service is processed on the Cryptographic Coprocessor Feature. If the key_type is MACD or IMP-PKA, the control vectors of supplied internal tokens must all be supported by the Cryptographic Coprocessor Feature, since processing for these key types will not be routed to a PCI Cryptographic Coprocessor. DATAC is not supported. Key_type CIPHER, DECIPHER and ENCIPHER require a PCICC.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>Key_type DATAXLAT is not supported. DES and CDMF markings are not made on DATA and key-encrypting keys and are ignored on the IMPORTER key-encrypting key. IMP-PKA keys are not supported.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

Key Part Import (CSNBKPI)

Use the key part import callable service to combine, by exclusive ORing, the clear key parts of any key type and return the combined key value either in an internal token or as an update to the CKDS.

Prior to using the key part import service for the first key part, you must use the key token build service to create the internal key token into which the key will be imported. Subsequent key parts are combined with the first part in internal token form or as a label from the CKDS.
Key Part Import (CSNBKPI)

The preferred way to specify key parts is FIRST, ADD-PART, and COMPLETE in the rule_array. Only when the combined key parts have been marked as COMPLETE can the key token be used in any other service. Key parts can also be specified as FIRST, MIDDLE, or LAST in the rule_array. ADD-PART or MIDDLE can be executed multiple times for as many key parts as necessary. Only when the LAST part has been combined can the key token be used in any other service.

New applications should employ the ADD-PART and COMPLETE keywords in lieu of the MIDDLE and LAST keywords in order to ensure a separation of responsibilities between someone who can add key-part information and someone who can declare that appropriate information has been accumulated in a key.

The key part import callable service can also be used to import a key without using key parts. Call the key part import service FIRST with key part value X'0000..' then call the key part import service LAST with the complete value.

Keys created via this service have odd parity. The FIRST key part is adjusted to odd parity. All subsequent key parts are adjusted to even parity prior to being combined.

Format

CALL CSNBKPI(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  rule_array_count,
  rule_array,
  key_part,
  key_identifier)

Parameters

return_code
Direction: Output Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

typecode

reason_code
Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

exit_data_length
Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.
Key Part Import (CSNBKPI)

exit_data
Direction: Input/Output Type: String

The data that is passed to the installation exit.

rule_array_count
Direction: Input Type: Integer

The number of keywords you supplied in the rule_array parameter. The value must be 1.

rule_array
Direction: Input Type: String

The keyword that provides control information to the callable service. The keywords must be 8 bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

Table 30. Keywords for Key Part Import Control Information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key Part (Required)</strong></td>
<td></td>
</tr>
<tr>
<td>FIRST</td>
<td>This keyword specifies that an initial key part is being entered. The callable service returns this key-part encrypted by the master key in the key token that you supplied.</td>
</tr>
<tr>
<td>ADD-PART</td>
<td>This keyword specifies that additional key-part information is provided.</td>
</tr>
<tr>
<td>COMPLETE</td>
<td>This keyword specifies that the key-part bit shall be turned off in the control vector of the key rendering the key fully operational. Note that no key-part information is added to the key with this keyword.</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>This keyword specifies that an intermediate key part, which is neither the first key part nor the last key part, is being entered. Note that the command control point for this keyword is the same as that for the LAST keyword and different from that for the ADD-PART keyword.</td>
</tr>
<tr>
<td>LAST</td>
<td>This keyword specifies that the last key part is being entered. The key-part bit is turned off in the control vector.</td>
</tr>
</tbody>
</table>

key_part
Direction: Input Type: String

A 16-byte field containing the clear key part to be entered. If the key is a single-length key, the key part must be left-justified and padded on the right with zeros. This field is ignored if COMPLETE is specified.

key_identifier
Direction: Input/Output Type: String

A 64-byte field containing an internal token or a label of an existing CKDS record. If rule_array is FIRST, this field is the skeleton of an internal token of a single- or double-length key with the KEY-PART marking. If rule_array is
Key Part Import (CSNBKPI)

MIDDLE or LAST, this is an internal token or the label of a CKDS record of a partially combined key. Depending on the input format, the accumulated partial or complete key is returned as an internal token or as an updated CKDS record. The returned key_identifier will be encrypted under the current master key.

Restrictions

If a label is specified on key_identifier, the label must be unique. If more than one record is found, the service fails.

For existing TKE V3.1 (or higher) users, you may have to explicitly enable new access control points. You must have access control point 'Key Part Import - Unrestricted' explicitly enabled. Otherwise, current applications will fail with either of these conditions:

- the first 8 bytes of key identifier is different than the second 8 bytes AND the first 8 bytes of the combined key are the same as the last second 8 bytes
- the first 8 bytes of key identifier is the same as the second 8 bytes AND the first 8 bytes of the combined key are different than the second 8 bytes.

This callable service does not support version X'10' external DES key tokens (RXX key tokens).

Usage Notes

If you are running with the Cryptographic Coprocessor Feature, this service requires that the ANSI system keys be installed on the CKDS.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 31. Key part import required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>Only key type AKEK is supported</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>PCI Cryptographic Coprocessor</td>
<td>ICSF routes all requests to the PCI Cryptographic Coprocessor except for key type of AKEK. AKEK is always processed on the Cryptographic Coprocessor Feature. Key type AKEK is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>AKEK key types are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>AKEK key types are not supported.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 31. Key part import required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>AKEK key types are not supported.</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

Related Information

This service is consistent with the Transaction Security System key part import verb.

Key Record Create (CSNBKRC)

Use the key record create callable service to add a key record to the CKDS that will be used to store AES and DES tokens. The record contains a key token set to binary zeros and is identified by the label passed in the key_label parameter. This service updates both the DASD copy of the CKDS currently in use by ICSF and the in-storage copy of the CKDS.

Format

CALL CSNBKRC(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  key_label
)

Parameters

return_code
Direction: Output Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

reason_code
Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

exit_data_length
Direction: Input/Output Type: Integer
Key Record Create (CSNBKRC)

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

exit_data
Direction: Input/Output Type: String

The data that is passed to the installation exit.

key_label
Direction: Input Type: Character string

The 64-byte label of a record in the CKDS that is the target of this service. The created record contains a key token set to binary zeros and has a key type of NULL.

Restrictions

The record must have a unique label. Therefore, there cannot be another record in the CKDS with the same label and a different key type.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

Usage Notes

The key record create callable service checks the syntax of the label provided in the key_label parameter to ensure that it follows the KGUP rules. To bypass label syntax checking, use a preprocessing exit to turn on the bypass parse bit in the Exit Parameter Control Block (EXPB). For more information about preprocessing exits and the EXPB, refer to the z/OS Cryptographic Services ICSF System Programmer's Guide.

You must use either the key record create callable service or KGUP to create an initial record in the CKDS prior to using the key record write service to update the record with a valid key token. Your applications perform better if you use KGUP to create the initial records and REFRESH the entire in-storage copy of the CKDS, rather than using key record create to create the initial NULL key entries. This is particularly true if you are creating a large number of key records. Key record create adds a record to a portion of the CKDS that is searched sequentially during key retrieval. Using KGUP followed by a REFRESH puts the null key records in the portion of the CKDS that is ordered in key-label/type sequence. A binary search of the key-label/type sequenced part of the CKDS is more efficient than searching the sequentially ordered section.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
Table 32. CKDS record create required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>None.</td>
<td></td>
</tr>
</tbody>
</table>

Key Record Delete (CSNBKRD)

Use the key record delete callable service to delete a key record containing a DES or AES token from both the DASD copy of the CKDS and the in-storage copy.

Format

```
CALL CSNBKRD(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_label)
```

Parameters

**return_code**
- Direction: Output
- Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**
- Direction: Output
- Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.
Key Record Delete (CSNBKRD)

exit_data_length
Direction: Input/Output                  Type: Integer

The length of the data that is passed to the installation exit. The length can be
from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the
exit_data parameter.

exit_data
Direction: Input/Output                  Type: String

The data that is passed to the installation exit.

rule_array_count
Direction: Input                  Type: Integer

The number of keywords supplied in the rule_array parameter. This number
must always be 1.

rule_array
Direction: Input                  Type: Character string

The 8 byte keyword that defines the action to be performed. The keyword must
be LABEL-DL.

key_label
Direction: Input                  Type: Character string

The 64-byte label of a record in the CKDS that is the target of this service. The
record can contain an AES or a DES key token. The record pointed to by this
label is deleted.

Restrictions
The record defined by the key_label must be unique. If more than one record per
label is found, the service fails.

This callable service does not support version X'10' external DES key tokens (RKX
key tokens).

Usage Notes
Secure key tokens cannot be processed when the master key is not loaded.

Clear AES and DES tokens can be processed on a system without a cryptographic
coprocessor or accelerator.

This table lists the required cryptographic hardware for each server type and
describes restrictions for this callable service.
Table 33. CKDS record delete required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>None.</td>
<td></td>
</tr>
</tbody>
</table>

Key Record Read (CSNBKRR)

Use the key record read callable service to copy an internal AES or DES key token from the in-storage CKDS to application storage. Other cryptographic services can then use the copied key token directly. The key token can also be used as input to the token copying functions of key generate, key import, or secure key import services to create additional NOCV keys.

Format

```
CALL CSNBKRR(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  key_label,
  key_token)
```

Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned.
Key Record Read (CSNBKRR)

to it indicating specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit_data_length**
- Direction: Input/Output
- Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the `exit_data` parameter.

**exit_data**
- Direction: Input/Output
- Type: String

The data that is passed to the installation exit.

**key_label**
- Direction: Input
- Type: Character string

The 64-byte label of a record containing an AES or DES token in the in-storage CKDS. The internal key token in this record is returned to the caller.

**key_token**
- Direction: Output
- Type: String

The 64-byte internal key token retrieved from the in-storage CKDS.

**Restrictions**

The record defined by the `key_label` parameter must be unique and must already exist in the CKDS.

If the internal key token is a clear key token, the token is not returned to the caller unless the caller is in supervisor state or system key.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

**Usage Notes**

Clear AES and DES tokens can be processed on a system without a cryptographic coprocessor or accelerator.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
Key Record Read (CSNBKRR)

Table 34. CKDS record read required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>None.</td>
<td></td>
</tr>
</tbody>
</table>

Key Record Write (CSNBKRW)

Use the key record write callable service to write an internal AES or DES key token to the CKDS record specified by the key_label parameter. This service supports writing a record to the CKDS which contains a key token with a control vector which is not supported by the Cryptographic Coprocessor Feature. These records will be written to the CKDS with a key type of CV, unless the key is a DES IMPORTER, EXPORTER, PINGEN, PINVER, IPINENC, or OPINENC type. These key types will be preserved in the CKDS record, even if the control vector is not supported by the Cryptographic Coprocessor Feature.

This service updates both the DASD copy of the CKDS currently in use by ICSF and the in-storage copy. The record you are updating must be unique and must already exist in both the DASD and in-storage copies of the CKDS.

This service supports writing a clear AES or DES key token with non-zero key values to the CKDS.

Format

```
CALL CSNBKRW(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_token,
    key_label)
```
Key Record Write (CSNBKRW)

Parameters

**return_code**

Direction: Output          Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

**reason_code**

Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit_data_length**

Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**

Direction: Input/Output Type: String

The data that is passed to the installation exit.

**key_token**

Direction: Input/output Type: String

The 64-byte internal AES or DES key token that is written to the CKDS.

**key_label**

Direction: Input Type: Character string

The 64-byte label of a record in the CKDS that is the target of this service. The record is updated with the AES or DES internal key token supplied in the key_token parameter.

Restrictions

The record defined by the key_label parameter must be unique and must already exist in the CKDS.

On CCF systems, writing a NOCV key-encrypting key is restricted to callers in supervisor mode or in system key.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).
Usage Notes

With a PCIXCC, CEX2C, or CEX3C, you can write NOCV keys to the CKDS without being in supervisor state.

Secure AES tokens in the CKDS can only be overwritten by a secure AES token encrypted under the same AES master keys. The same is true for secure DES tokens.

DES tokens cannot be overwritten by an AES token. AES tokens cannot be overwritten by a DES token.

Secure key tokens cannot be processed when the master key is not loaded.

Clear AES and DES tokens can be processed on a system without a cryptographic coprocessor or accelerator.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>None.</td>
<td></td>
</tr>
</tbody>
</table>

Related Information

You can use this service with the key record create callable service to write an initial record to key storage. Use it following the key import and key generate callable services to write an operational key imported or generated by these services directly to the CKDS.

Key Test (CSNBKYT)

Use the key test callable service to generate or verify a secure, cryptographic verification pattern for keys. The key to test can be in the clear or encrypted under the master key. Keywords in the rule array specify whether the callable service generates or verifies a verification pattern.
Key Test (CSNBKYT)

DES keys use the algorithm defined in "Key Test Verification Pattern Algorithm for DES Keys" on page 699 as the default algorithm (except for triple-length DATA keys). When generating a verification pattern, the service generates a random number and calculates the verification pattern. The random number and verification pattern are returned to the caller. When verifying a key, the random number and key are used to verify the verification pattern.

AES keys use the SHA-256 algorithm as the default algorithm. An 8-byte verification pattern is generated for the key specified. The random number parameter is not used.

The optional ENC-ZERO algorithm can be used with any key. A 4-byte verification pattern is generated. The random number parameter is not used.

CSNBKYT is consistent with the Transaction Security System verb of the same name. If you generate a key on the Transaction Security System, you can verify it on ICSF and vice versa.

See "Key Test Extended (CSNBKTYX)" on page 144 to verify the value of a DES key encrypted using a KEK.

Format

```call CSNBKTYT(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  rule_array_count,
  rule_array,
  key_identifier,
  random_number,
  verification_pattern)
```

Parameters

`return_code`

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

`reason_code`

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

`exit_data_length`

Direction: Input/Output

Type: Integer
Key Test (CSNBKYT)

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**rule_array_count**

Direction: Input  
Type: Integer

The number of keywords you supplied in the rule_array parameter. The value can be 2, 3 or 4.

**rule_array**

Direction: Input  
Type: String

Keywords provide control information to the callable service. Table 36 lists the keywords. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key or key part rule (one keyword required)</strong></td>
<td></td>
</tr>
<tr>
<td>CLR-A128</td>
<td>Process a 128–bit AES clear key.</td>
</tr>
<tr>
<td>CLR-A192</td>
<td>Process a 192–bit AES clear key.</td>
</tr>
<tr>
<td>CLR-A256</td>
<td>Process a 256–bit AES clear key.</td>
</tr>
<tr>
<td>KEY-CLR</td>
<td>Specifies the key supplied in key_identifier is a single-length clear key.</td>
</tr>
<tr>
<td>KEY-CLRD</td>
<td>Specifies the key supplied in key_identifier is a double-length clear key.</td>
</tr>
<tr>
<td>KEY-ENC</td>
<td>Specifies the key supplied in key_identifier is a single-length encrypted key.</td>
</tr>
<tr>
<td>KEY-ENCD</td>
<td>Specifies the key supplied in key_identifier is a double-length encrypted key.</td>
</tr>
<tr>
<td>TOKEN</td>
<td>Process an AES clear or encrypted key token.</td>
</tr>
</tbody>
</table>

**Process Rule (one keyword required)**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENERATE</td>
<td>Generate a verification pattern for the key supplied in key_identifier.</td>
</tr>
<tr>
<td>VERIFY</td>
<td>Verify a verification pattern for the key supplied in key_identifier.</td>
</tr>
</tbody>
</table>

**Parity Adjustment - can not be specified with any of the AES keywords (optional)**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADJUST</td>
<td>Adjust the parity of test key to odd prior to generating or verifying the verification pattern. The key_identifier field itself is not adjusted.</td>
</tr>
<tr>
<td>NOADJUST</td>
<td>Do not adjust the parity of test key to odd prior to generating or verifying the verification pattern. This is the default.</td>
</tr>
</tbody>
</table>
Key Test (CSNBKYT)

Table 36. Keywords for Key Test Control Information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Verification Process Rule (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>ENC-ZERO</td>
<td>ENC-ZERO can be used with any of the rules. It is not supported on systems with CCFs.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>Use the 'SHA-256' method. Use with CLR-A128, CLR-A192, CLR-A256, and TOKEN. SHA-256 is also the default for the AES rules.</td>
</tr>
</tbody>
</table>

**key_identifier**

Direction: Input/Output  
Type: String

The key for which to generate or verify the verification pattern. The parameter is a 64-byte string of an internal token, key label, or a clear key value left-justified.

**Note:** If you supply a key label for this parameter, it must be unique on the CKDS.

**random_number**

Direction: Input/Output  
Type: String

This is an 8-byte field that contains a random number supplied as input for the test pattern verification process and returned as output with the test pattern generation process. random_number is only used with the default algorithm for DES operational keys.

**verification_pattern**

Direction: Input/Output  
Type: String

This is an 8-byte field that contains a verification pattern supplied as input for the test pattern verification process and returned as output with the test pattern generation process.

**Restrictions**

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

You can generate the verification pattern for a key when you generate the key. You can distribute the pattern with the key and it can be verified at the receiving node. In this way, users can ensure using the same key at the sending and receiving locations. You can generate and verify keys of any combination of key forms, that is, clear, operational or external.

The parity of the key is not tested.
With a PCIXC, CEX2C, or CEX3C, there is support for the generation and verification of single, double and triple-length keys for the ENC-ZERO verification process. For triple-length keys, use KEY-ENC or KEY-ENCD with ENC-ZERO. Clear triple-length keys are not supported.

In the Transaction Security System, KEY-ENC and KEY-ENCD both support enciphered single-length and double-length keys. They use the key-form bits in byte 5 of CV to determine the length of the key. To be consistent, in ICSF, both KEY-ENC and KEY-ENCD handle single- and double-length keys. Both products effectively ignore the keywords, which are supplied only for compatibility reasons.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 37: Key test required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>Triple-length DATA keys are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>PCI Cryptographic Coprocessor</td>
<td>Triple-length DATA keys are not supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ICSF routes the request to a PCI Cryptographic Coprocessor if:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ANSI enablement keys are not installed in the CKDS.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Verification process rule ENC-ZERO is specified.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AES keys are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>Clear triple-length keys are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td>Encrypted triple-length keys are supported with the ENC-ZERO keyword only.</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>AES keys are not supported.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td>Secure AES keys require the Nov. 2008 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Clear triple-length keys are not supported.</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td>Encrypted triple-length keys are supported with the ENC-ZERO keyword only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Secure AES keys require the Nov. 2008 or later licensed internal code (LIC).</td>
</tr>
</tbody>
</table>
Key Test Extended (CSNBKYTX)

Use the key test extended service to generate or verify a secure, cryptographic verification pattern for keys. The key to test can be in the clear or encrypted under the master key. The callable service also supports keys encrypted under a key-encrypting key (KEK). AES keys are not supported by CSNBKYTX. Keywords in the rule array specify whether the callable service generates or verifies a verification pattern.

This algorithm is supported for encrypted single and double length keys. Single, double and triple length keys are also supported with the ENC-ZERO algorithm.

When the service generates a verification pattern, it creates and cryptographically processes a random number. The service returns the random number with the verification pattern.

When the service tests a verification pattern against a key, you must supply the random number and the verification pattern from a previous call to key test extended. The service returns the verification result in the return and reason codes.

Format

```call csnbkytx(return_code, reason_code, exit_data_length, exit_data, rule_array_count, rule_array, key_identifier, random_number, verification_pattern, KEK_key_identifier)
```

Parameters

**return_code**  
Direction: Output  
Type: Integer  

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**  
Direction: Output  
Type: Integer  

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**  
Direction: Input/Output  
Type: Integer
The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**

Direction: Input/Output   Type: String

The data that is passed to the installation exit.

**rule_array_count**

Direction: Input   Type: Integer

The number of keywords you supplied in the rule_array parameter. The value can be 2, 3 or 4.

**rule_array**

Direction: Input   Type: String

Two or three keywords that provide control information to the callable service. Table 38 lists the keywords. The keywords must be in 16 or 24 bytes of contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

**Table 38. Keywords for Key Test Extended Control Information**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key Rule (required)</strong></td>
<td></td>
</tr>
<tr>
<td>KEY-ENC</td>
<td>Specifies the key supplied in key_identifier is a single-length encrypted DES key.</td>
</tr>
<tr>
<td>KEY-ENCD</td>
<td>Specifies the key supplied in key_identifier is a double-length encrypted DES key.</td>
</tr>
<tr>
<td><strong>Process Rule (required)</strong></td>
<td></td>
</tr>
<tr>
<td>GENERATE</td>
<td>Generate a verification pattern for the key supplied in key_identifier.</td>
</tr>
<tr>
<td>VERIFY</td>
<td>Verify a verification pattern for the key supplied in key_identifier.</td>
</tr>
<tr>
<td><strong>Parity Adjustment (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>ADJUST</td>
<td>Adjust the parity of test key to odd prior to generating or verifying the verification pattern. The key_identifier field itself is not adjusted.</td>
</tr>
<tr>
<td>NOADJUST</td>
<td>Do not adjust the parity of test key to odd prior to generating or verifying the verification pattern. This is the default.</td>
</tr>
<tr>
<td><strong>Verification Process Rule (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>ENC-ZERO</td>
<td>Specifies use of the &quot;encrypted zeros&quot; method.</td>
</tr>
</tbody>
</table>

**key_identifier**

Direction: Input/Output   Type: String

The key for which to generate or verify the verification pattern. The parameter is a 64-byte string of an internal token or key label that is left-justified.
Key Test Extended (CSNBKYTE)

**Note:** If you supply a key label for this parameter, it must be unique on the CKDS.

**random_number**

Direction: Input/Output  
Type: String

This is an 8-byte field that contains a random number supplied as input for the test pattern verification process and returned as output with the test pattern generation process.

**verification_pattern**

Direction: Input/Output  
Type: String

This is an 8-byte field that contains a verification pattern supplied as input for the test pattern verification process and returned as output with the test pattern generation process.

**KEK_key_identifier**

Direction: Input/Output  
Type: String

If `key_identifier` is an external token, then this is a 64-byte string of an internal token or a key label of an IMPORTER or EXPORTER used to encrypt the test key. If `key_identifier` is an internal token, then the parameter is ignored.

**Note:** If you supply a key label for this parameter, it must be unique on the CKDS.

**Restrictions**

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

You can generate the verification pattern for a key when you generate the key. You can distribute the pattern with the key and it can be verified at the receiving node. In this way, users can ensure using the same key at the sending and receiving locations. You can generate and verify keys of any combination of key forms, that is, clear, operational or external.

The parity of the key is not tested.

With a PCIXCC, CEX2C, or CEX3C and using the ENC-ZERO verification rule, there is support for enciphered single and double-length DES keys. There is no support for systems with CCF’s.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
## Key Test Extended (CSNBKYTX)

### Table 39. Key test extended required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>Triple-length DATA keys are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td>The key test extended callable service is processed on the Cryptographic Coprocessor Feature. <strong>Rule_array keyword ENC-ZERO is not valid.</strong></td>
</tr>
<tr>
<td></td>
<td>PCI Cryptographic Coprocessor</td>
<td>Triple-length DATA keys are not supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ICSF routes the request to a PCI Cryptographic Coprocessor if:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ANSI enablement keys are not installed in the CKDS.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Verification process rule ENC-ZERO is specified.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AES keys are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>Clear triple-length keys are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td>Encrypted triple-length keys are supported with the ENC-ZERO keyword only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AES keys are not supported.</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Clear triple-length keys are not supported.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td>Encrypted triple-length keys are supported with the ENC-ZERO keyword only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AES keys are not supported.</td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Clear triple-length keys are not supported.</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td>Encrypted triple-length keys are supported with the ENC-ZERO keyword only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AES keys are not supported.</td>
</tr>
</tbody>
</table>

### Key Token Build (CSNBKTB)

Use the key token build callable service to build an external or internal key token from information which you supply. The token can be used as input for the key generate and key part import callable services. You can specify a control vector or the service can build a control vector based upon the key type you specify and the control vector-related keywords in the rule array. ICSF supports the building of an internal key token with the key encrypted under a master key other than the current master key and building internal clear AES and DES tokens.
Key Token Build (CSNBKTB)

Format

```c
CALL CSNBKTB(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_token,
    key_type,
    rule_array_count,
    rule_array,
    key_value,
    master_key_version_number,
    key_register_number,
    token_data_1,
    control_vector,
    initialization_vector,
    pad_character,
    cryptographic_period_start,
    master_key_verification_pattern)
```

Parameters

return_code
Direction: Output Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code
Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

exit_data_length
Direction: Input/Output Type: Integer

Reserved field.

exit_data
Direction: Input/Output Type: String

Reserved field.

key_token
Direction: Input/Output Type: String

If the key_type parameter is TOKEN, then this is a 64-byte internal token that is updated as specified in the rule_array. The internal token must be a DATA, IMPORTER or EXPORTER key type. Otherwise this field is an output-only field.
**Key Token Build (CSNBKTB)**

**key_type**

Direction: Input  
Type: String

An 8-byte field that specifies the type of key you want to build or the keyword TOKEN for updating a supplied token. The key types are:

*Table 40. Key type keywords for key token build*

<table>
<thead>
<tr>
<th>Key type</th>
<th>Description</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>AKEK</td>
<td>See Table 2 on page 21</td>
<td>DES</td>
</tr>
<tr>
<td>CIPHER</td>
<td>See Table 2 on page 21</td>
<td>DES</td>
</tr>
<tr>
<td>CLRAES</td>
<td>The key_token parameter is a clear AES token. The rule_array must contain the keyword INTERNAL and one of the optional keywords: KEYLN16, KEYLN24 or KEYLN32. A key value parameter must also be provided.</td>
<td>AES</td>
</tr>
<tr>
<td>CLRDES</td>
<td>The key_token parameter is a clear DES token. The rule_array must contain the keyword INTERNAL and one of the optional keywords: KEYLN8, KEYLN16 or KEYLN24. A key value parameter must also be provided.</td>
<td>DES</td>
</tr>
<tr>
<td>CVARDEC, CVARENC, CVARPINE, CVARXCVL, CVARXCVR</td>
<td>See Table 2 on page 21</td>
<td>DES</td>
</tr>
<tr>
<td>DATA</td>
<td>Valid for AES and DES keys and must be specified with the rule_array keyword AES to build an encrypted AES key token.</td>
<td>AES and DES</td>
</tr>
<tr>
<td>DATAC, DATAM, DATAMV, DATAXLAT, DECIPHER, DKYGENKY, ENCIPHER</td>
<td>See Table 2 on page 21</td>
<td>DES</td>
</tr>
<tr>
<td>EXPORTER</td>
<td>If the key_type parameter is TOKEN, then this is a 64-byte internal token that is updated as specified in the rule_array.</td>
<td>DES</td>
</tr>
<tr>
<td>IKEYXLAT</td>
<td>See Table 2 on page 21</td>
<td>DES</td>
</tr>
<tr>
<td>IMPORTER</td>
<td>If the key_type parameter is TOKEN, then this is a 64-byte internal token that is updated as specified in the rule_array.</td>
<td>DES</td>
</tr>
<tr>
<td>KEYGENKY</td>
<td>CLR8-ENC or UKPT must be coded in rule_array parameter</td>
<td>DES</td>
</tr>
<tr>
<td>IPINENC, MAC, MACVER, OKEYXLAT, OPINENC, PINGEN, PINVER, and SECMSG</td>
<td>See Table 2 on page 21</td>
<td>DES</td>
</tr>
<tr>
<td>SECMSG</td>
<td>SMKEY or SMPIN must be specified in the rule_array parameter.</td>
<td>DES</td>
</tr>
</tbody>
</table>
Key Token Build (CSNBKTB)

Unless specified otherwise, the key_type values are valid only for DES keys.

If key_type is TOKEN, then the key_token field must contain a single-length DATA key or an IMPORTER or EXPORTER key with the standard control vector. The valid keywords for TOKEN are EXTERNAL, INTERNAL, DES and SYS-ENC. The service will set the system encryption bits in the token (byte 59, bits 0 and 1) to zero and return the token.

Key type USE-CV is used when a user-supplied control vector is specified. The USE-CV key type specifies that the key type should be obtained from the control vector specified in the control_vector parameter. The CV rule array keyword should be specified if USE-CV is specified.

For information on the meaning of the key types, see Table 2 on page 21.

rule_array_count
Direction: Input Type: Integer

The number of keywords you supplied in the rule_array parameter.

rule_array
Direction: Input Type: String

The rule_array contains keywords that provide control information to the callable service. See Table 41 for a list. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks. For any key type, there are no more than four valid rule_array values.

Table 41. Keywords for Key Token Build Control Information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Token Algorithm (optional - zero or one keyword)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>Specifies that an AES key token will be built. This keyword is required when building an encrypted AES token. It is optional when using the CLRAES key type to build a clear AES token.</td>
<td>AES</td>
</tr>
<tr>
<td>DES</td>
<td>Specifies a DES token will be built.</td>
<td>DES</td>
</tr>
<tr>
<td>SYS-ENC</td>
<td>Tolerated for compatibility reasons.</td>
<td>DES</td>
</tr>
<tr>
<td><strong>Token Type (one keyword required)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXTERNAL</td>
<td>Specifies that an external key token will be built.</td>
<td>DES</td>
</tr>
<tr>
<td>INTERNAL</td>
<td>Specifies that an internal key token will be built.</td>
<td>AES or DES</td>
</tr>
<tr>
<td><strong>Key Status (optional - zero or one keyword)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEY</td>
<td>This keyword indicates that the key token to build will contain an encrypted key. The key_value parameter identifies the field that contains the key.</td>
<td>AES or DES</td>
</tr>
<tr>
<td>NO-KEY</td>
<td>This keyword indicates that the key token to build will not contain a key. This is the default key status.</td>
<td>AES or DES</td>
</tr>
<tr>
<td><strong>Key Length (one keyword required for AES keys)</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 41. Keywords for Key Token Build Control Information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYLN8</td>
<td>Single-length or 8-byte key. Default for CLRDES.</td>
<td>DES</td>
</tr>
<tr>
<td>KEYLN16</td>
<td>Specifies that the key is 16-bytes long. AES or DES</td>
<td></td>
</tr>
<tr>
<td>KEYLN24</td>
<td>Specifies that the key is 24-bytes long. AES or DES</td>
<td></td>
</tr>
<tr>
<td>KEYLN32</td>
<td>Specifies that the key is 32-bytes long. AES</td>
<td></td>
</tr>
<tr>
<td>DOUBLE</td>
<td>Double-length or 16-byte key. Synonymous with KEYLN16. Not valid for CLRDES. <strong>Note:</strong> See <a href="#">Table 43 on page 154</a> for valid key types for these key length values.</td>
<td>DES</td>
</tr>
<tr>
<td>MIXED</td>
<td>Double-length key. Indicates that the key can either be a replicated single-length key or a double-length key with two different 8-byte values. Not valid for CLRDES.</td>
<td>DES</td>
</tr>
<tr>
<td>SINGLE</td>
<td>Single-length or 8-byte key. Synonymous with KEYLN8. Not valid for CLRDES.</td>
<td>DES</td>
</tr>
</tbody>
</table>

**Key Part Indicator (optional) — not valid for CLRDES**

| KEY-PART | This token is to be used as input to the key part import service. | DES |

**Control vector (CV) source (optional - zero or one of these keywords is permitted)**

| CV       | This specifies that the key token should be built using the control_vector supplied in the control_vector parameter. | DES |
| NO-CV    | This specifies that the key token should be built using a control vector that is based on the supplied key type control vector related rule array keywords. It is the default. | DES |

**Control vector on the link specification (optional) — valid only for IMPORTER and EXPORTER.**

| CV-KEK   | This keyword indicates marking the KEK as a CV KEK. The control vector is applied to the KEK prior to using it in encrypting other keys. This is the default. | DES |
| NOCV-KEK | This keyword indicates marking the KEK as a NOCV KEK. The control vector is not applied to the KEK prior to its use in encrypting other keys. Services using NO-CV keys must be processed on the Cryptographic Coprocessor Feature. | DES |

**Control vector keywords (optional - zero or more of these keywords are permitted)**

See [Table 43 on page 154](#) for the key-usage keywords that can be specified for a given key type.

**Master Key Verification Pattern (optional) — not valid for CLRDES or CLRAES keywords**
Table 41. Keywords for Key Token Build Control Information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>MKVP</td>
<td>This keyword indicates that the <em>key_value</em> is enciphered under the master key which corresponds to the master key verification pattern specified in the <em>master_key_verification_pattern</em> parameter. If this keyword is not specified, the key contained in the <em>key_value</em> field must be enciphered under the current master key.</td>
<td>AES and DES</td>
</tr>
</tbody>
</table>

**key_value**

Direction: Input  
Type: String

If you use the KEY keyword, this parameter is a 16-byte string that contains the encrypted key value. Single-length keys must be left-justified in the field and padded on the right with X'00'. If you are building a triple-length DATA key, this parameter is a 24-byte string containing the encrypted key value. If you supply an encrypted key value and also specify INTERNAL, the service will check for the presence of the MKVP keyword. If MKVP is present, the service will assume the *key_value* is enciphered under the master key which corresponds to the master key verification pattern specified in the *master_key_verification_pattern* parameter, and will place the key into the internal token along with the verification pattern from the *master_key_verification_pattern* parameter. If MKVP is not specified, ICSF assumes the key is enciphered under the current host master key and places the key into an internal token along with the verification pattern for the current master key. In this case, the application must ensure that the master key has not changed since the key was generated or imported to this system. Otherwise, use of this parameter is not recommended.

For *key_type* CLRDES and CLRAES, this field is required to contain the clear key value. For KEYLN8, this is an 8-byte field. For KEYLN16, this is a 16-byte field. For KEYLN24, this a 24-byte field. For KEYLN32, this is a 32-byte field.

Table 42. Key types and field lengths for AES keys

<table>
<thead>
<tr>
<th>Key type</th>
<th>Field length</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-128 clear text key</td>
<td>16-bytes</td>
</tr>
<tr>
<td>AES-192 clear text key</td>
<td>24-bytes</td>
</tr>
<tr>
<td>AES-256 clear text key</td>
<td>32-bytes</td>
</tr>
<tr>
<td>AES-128, AES-192, AES-256 encrypted key</td>
<td>32-bytes</td>
</tr>
</tbody>
</table>

**master_key_version_number**

Direction: Input  
Type: Integer

This field is examined only if the KEY keyword is specified, in which case, this field must be zero.

**key_register_number**

Direction: Input  
Type: Integer
Key Token Build (CSNBKTB)

This field is ignored.

**token_data_1 (was secure_token)**

Direction: Input  
Type: String

This parameter is ignored for DES keys.

This parameter is the LRC value for AES keys. For clear AES keys it is 8-bytes of X’00’ indicating to the service that it must compute the LRC field value. For encrypted AES keys, you provide a 1-byte area containing the LRC value for the key passed in the *key_value* parameter. The service copies it into the LRC field of the key token.

**control_vector**

Direction: Input  
Type: String

A pointer to a 16 byte string variable. When the CV rule array keyword is used, this parameter must point to a control vector which is copied into the key token. This parameter is ignored for AES keys.

**initialization_vector**

Direction: Input  
Type: String

This field is ignored.

**pad_character**

Direction: Input  
Type: Integer

The only allowed value for key types MAC and MACVER is 0. This field is ignored for all other key types.

**cryptographic_period_start**

Direction: Input  
Type: String

This field is ignored.

**master_key_verification_pattern**

Direction: Input  
Type: String

A pointer to an 8-byte string variable. It is used when the KEY and INTERNAL rule_array keywords are specified. The value is inserted into the master key verification pattern field of the key token. If the KEY and INTERNAL keywords are specified in rule_array, the service will check for the existence of the MKVP rule array keyword. This parameter is ignored for any other rule_array keyword combinations.

**Restrictions**

This callable service does not support version X'10' external DES key tokens (RXX key tokens).
Usage Notes

No pre- or post-processing or security exits are enabled for this service. No RACF checking is done, and no calls to RACF are issued when this service is used.

You can use this service to create skeleton key tokens with the desired data encryption algorithm bits for use in some key management services to override the default system specifications.

- If you are running with the Cryptographic Coprocessor Feature and need to generate operational AKEKs, use `key_type` of TOKEN and provide a skeleton AKEK key token as the `generated_key_identifier_1` into the key generate service.
- If you are running with the Cryptographic Coprocessor Feature, the KEY-PART AKEK key token can also be used as input to key part import service.
- To create an internal token with a specified KEY value, ICSF needs to supply a valid master key verification pattern (MKVP).

NOCV keyword is only supported for the standard IMPORTERs and EXPORTERs with the default CVs.

This illustrates the key type and key usage keywords that can be combined in the Control Vector Generate and Key Token Build callable services to create a control vector.

### Table 43. Control Vector Generate and Key Token Build Control Vector Keyword Combinations

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Key Usage</th>
<th>XPORT-OK</th>
<th>NO-XPORT</th>
<th>KEY-PART</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>SINGLE</td>
<td>XPORT-OK</td>
<td>NO-XPORT</td>
<td>KEY-PART</td>
</tr>
<tr>
<td></td>
<td>KEYLN8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MIXED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DOUBLE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>KEYLN16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>KEYLN24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CIPHER</td>
<td>SINGLE</td>
<td>XPORT-OK</td>
<td>NO-XPORT</td>
<td>KEY-PART</td>
</tr>
<tr>
<td>ENCIPHER</td>
<td>KEYLN8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECIPHER</td>
<td>MIXED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAC</td>
<td>DOUBLE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACVER</td>
<td>KEYLN16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATAXLAT</td>
<td>SINGLE</td>
<td>XPORT-OK</td>
<td>NO-XPORT</td>
<td>KEY-PART</td>
</tr>
<tr>
<td>CVARPINE</td>
<td>KEYLN8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARENCE</td>
<td>MIXED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARDEC</td>
<td>DOUBLE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARXCVL</td>
<td>KEYLN16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVARXCVR</td>
<td>KEYLN16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATAC</td>
<td>DOUBLE</td>
<td>XPORT-OK</td>
<td>NO-XPORT</td>
<td>KEY-PART</td>
</tr>
<tr>
<td>DATAM</td>
<td>KEYLN16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATAMV</td>
<td>MIXED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEYGENKY</td>
<td>DOUBLE</td>
<td>XPORT-OK</td>
<td>NO-XPORT</td>
<td>KEY-PART</td>
</tr>
<tr>
<td>CLR8-ENC</td>
<td>KEYLN16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UKPT</td>
<td>MIXED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DKYGENKY</td>
<td>DOUBLE</td>
<td>XPORT-OK</td>
<td>NO-XPORT</td>
<td>KEY-PART</td>
</tr>
<tr>
<td>DDATA</td>
<td>KEYLN16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DMAC</td>
<td>MIXED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DMV</td>
<td>KEYLN16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIMP</td>
<td>KEYLN16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEXP</td>
<td>KEYLN16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DPKVR</td>
<td>KEYLN16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DMKEY</td>
<td>KEYLN16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DMPIN</td>
<td>KEYLN16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DALL</td>
<td>KEYLN16</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 43. Control Vector Generate and Key Token Build Control Vector Keyword Combinations (continued)

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Key Usage</th>
<th>Key Type</th>
<th>Key Usage</th>
<th>Key Type</th>
<th>Key Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECMSG</td>
<td>SMKEY</td>
<td>DOUBLE</td>
<td>XPORT-OK</td>
<td>KEY-PART</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SMPIN</td>
<td>KEYLN16</td>
<td>NO-XPORT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IKEYXLAT</td>
<td>OKEYXLAT</td>
<td>ANY</td>
<td>XPORT-OK</td>
<td>KEY-PART</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOT-KEK</td>
<td>NO-XPORT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LMTD-KEK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMPORTER</td>
<td>OPIM*</td>
<td>ANY</td>
<td>XPORT-OK</td>
<td>KEY-PART</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IMEX*</td>
<td>NOT-KEK</td>
<td>NO-XPORT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IMIM*</td>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IMPORT*</td>
<td>PIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LMTD-KEK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXPORTER</td>
<td>OPEX*</td>
<td>ANY</td>
<td>XPORT-OK</td>
<td>KEY-PART</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IMEX*</td>
<td>NOT-KEK</td>
<td>NO-XPORT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXEX*</td>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXPORT*</td>
<td>PIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LMTD-KEK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PINVER</td>
<td>NO-SPEC**</td>
<td>NOOFFSET</td>
<td>XPORT-OK</td>
<td>KEY-PART</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IBM-PIN**</td>
<td>KEYLN16</td>
<td>NO-XPORT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GBP-PIN**</td>
<td>MIXED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IBM-PINO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GBP-PINO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VISA-PVV</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INBK-PIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PINGEN</td>
<td>CPINGEN*</td>
<td>NOOFFSET</td>
<td>XPORT-OK</td>
<td>KEY-PART</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CPINGENA*</td>
<td>KEYLN16</td>
<td>NO-XPORT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EPINGENA*</td>
<td>MIXED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EPINGEN*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EPINVER*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPINENC</td>
<td>CPINGEN*</td>
<td>DOUBLE</td>
<td>XPORT-OK</td>
<td>KEY-PART</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EPINVER*</td>
<td>KEYLN16</td>
<td>NO-XPORT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>REFORMAT*</td>
<td>MIXED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TRANSLAT*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPINENC</td>
<td>CPINENC*</td>
<td>DOUBLE</td>
<td>XPORT-OK</td>
<td>KEY-PART</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EPINGEN*</td>
<td>KEYLN16</td>
<td>NO-XPORT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>REFORMAT*</td>
<td>MIXED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TRANSLAT*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes: Default keys are indicated in bold.

CLR8-ENC and/or UKPT must be specified for the KEYGENKY key type - SMKEY or SMPIN must be specified for the SECMSG key type.

* All keywords in the list are defaults unless one or more keywords in the list are specified.

A key usage keyword is required for the KEYGENKY key type.

** The NOOFFSET keyword is only valid if NO-SPEC, IBM-PIN, GBP-PIN, or the default (NO-SPEC) is specified.

See Appendix C, “Control Vectors and Changing Control Vectors with the CVT Callable Service,” on page 633.
Key Token Build (CSNBKTB)

Related Information

Attention: CDMF is no longer supported.

The ICSF key token build callable service provides a subset of the parameters and keywords available with the Transaction Security System key token build verb.

These key types are not supported: ADATA, AMAC, CIPHERXI, CIPHERXL, CIPHERXO, UKPTBASE.

These rule array keywords are not supported: ACTIVE, ADAPTER, CARD, CBC, CLEAR-IV, CUSP, INACTIVE, IPS, KEY-REF, MACLEN4, MACLEN6, MACLEN8, NO-IV, READER, X9.2, X9.9-1.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 44. Key token build required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>None.</td>
<td></td>
</tr>
</tbody>
</table>

Key Translate (CSNBKTR)

The Key Translate callable service uses one key-encrypting key to decipher an input key and then enciphers this key using another key-encrypting key within the secure environment.

Note: All key labels must be unique.
Key Translate (CSNBKTR)

Format

```
CALL CSNKTR(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    input_key_token,
    input_KEK_key_identifier,
    output_KEK_key_identifier,
    output_key_token )
```

Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. [Appendix A, "ICSF and TSS Return and Reason Codes"](#) lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. [Appendix A, "ICSF and TSS Return and Reason Codes"](#) lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**input_key_token**

Direction: Input  
Type: String

A 64-byte string variable containing an external key token. The external key token contains the key to be re-enciphered (translated).

**input_KEK_key_identifier**

Direction: Input/Output  
Type: String

A 64-byte string variable containing the internal key token or the key label of an internal key token record in the CKDS. The internal key token contains the key-encrypting key used to decipher the key. The internal key token must
Key Translate (CSNBKTR)

contain a control vector that specifies an importer or IKEYXLAT key type. The control vector for an importer key must have the XLATE bit set to 1.

**output KEK_key_identifier**

Direction: Input/Output  
Type: String

A 64-byte string variable containing the internal key token or the key label of an internal key token record in the CKDS. The internal key token contains the key-encrypting key used to encipher the key. The internal key token must contain a control vector that specifies an exporter or OKEYXLAT key type. The control vector for an exporter key must have the XLATE bit set to 1.

**output_key_token**

Direction: Output  
Type: String

A 64-byte string variable containing an external key token. The external key token contains the re-enciphered key.

**Restrictions**

Triple length DATA key tokens are not supported.

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

**Table 45. Key translate required hardware**

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>PCI Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>
Multiple Clear Key Import (CSNBCKM and CSNECKM)

The multiple clear key import callable service imports a clear AES or DES key, enciphers the key under the corresponding master key, and returns the enciphered key in an internal key token. The enciphered key's type is DATA. The enciphered key is in operational form.

This callable service supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSNECKM.

Format

```call csnbckm(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    clear_key_length,
    clear_key,
    key_identifier_length,
    key_identifier
)
```

Parameters

return_code

Direction: Output Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code

Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

exit_data_length

Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

exit_data

Direction: Input/Output Type: String

The data that is passed to the installation exit.
Multiple Clear Key Import (CSNBCKM and CSNECKM)

**rule_array_count**

Direction: Input  
Type: Integer

The number of keywords you are supplying in the `rule_array` parameter. The `rule_array_count` parameter must be 0 or 1.

**rule_array**

Direction: Input  
Type: String

Zero or one keyword that supplies control information to the callable service. The keyword must be in 8 bytes of contiguous storage, left-justified and padded on the right with blanks. Refer to Table 46 for a list of keywords.

Table 46. Keywords for Multiple Clear Key Import Rule Array Control Information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm (optional)</td>
<td></td>
</tr>
<tr>
<td>CDMF</td>
<td>The output key identifier is to be a CDMF token. For a DATA key of length 16 or 24, you may not specify CDMF. CDMF is only supported on CCF systems.</td>
</tr>
<tr>
<td>AES</td>
<td>The output key identifier is to be an AES token.</td>
</tr>
<tr>
<td>DES</td>
<td>The output key identifier is to be a DES token. This is the default.</td>
</tr>
</tbody>
</table>

**clear_key_length**

Direction: Input  
Type: Integer

The `clear_key_length` specifies the length of the clear key value to import in bytes. For DES keys, this length must be 8-, 16-, or 24-bytes. For AES keys, this length must be 16-, 24-, or 32-bytes.

**clear_key**

Direction: Input  
Type: String

The `clear_key` specifies the clear key value to import.

**key_identifier_length**

Direction: Input/Output  
Type: Integer

The byte length of the `key_identifier` parameter. This must be exactly 64 bytes.

**key_identifier**

Direction: Input/Output  
Type: String

A 64-byte string that is to receive an internal AES or DES key token.

**Usage Notes**

This service produces an internal DES DATA token with a control vector which is usable on the Cryptographic Coprocessor Feature. If a valid internal DES token is
Multiple Clear Key Import (CSNCKM and CSNECKM)

supplied as input to the service in the key_identifier field, that token's control vector will not be used in the encryption of the clear key value.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 47. Multiple clear key import required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>Tokens are not marked with the system encryption algorithm.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>CDMF keyword is not supported. Tokens are not marked with the system encryption algorithm.</td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>Crypto Express2 Coprocessor</td>
<td>CDMF keyword is not supported. Tokens are not marked with the system encryption algorithm.</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>CDMF keyword is not supported. Tokens are not marked with the system encryption algorithm. Secure AES keys require the Nov. 2008 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>CDMF keyword is not supported. Tokens are not marked with the system encryption algorithm. Secure AES keys require the Nov. 2008 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

Multiple Secure Key Import (CSNBSKM)

Use this service to encipher a single-length, double-length, or triple-length DES key under the system master key or an importer key-encrypting key. The clear DES key can then be imported as any of the possible key types.

In addition to DES keys, this service imports a clear AES key, enciphers the AES key under the AES master key, and returns the enciphered key in an internal token. The enciphered key's type is DATA. The enciphered key is in operational form.

The callable service can execute only when ICSF is in special secure mode, which is described in "Special Secure Mode" on page 10.
Multiple Secure Key Import (CSNBSKM)

Format

```call
CALL CSNBSKM(
return_code,
reason_code,
exit_data_length,
exit_data,
rule_array_count,
rule_array,
clear_key_length,
clear_key,
key_type,
key_form,
key_encrypting_key_identifier,
imported_key_identifier_length,
imported_key_identifier)
```

Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. Appendix A, ICSF and TSS Return and Reason Codes lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, ICSF and TSS Return and Reason Codes lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**rule_array_count**

Direction: Input  
Type: Integer

The number of keywords you are supplying in the rule_array parameter. The rule_array_count parameter must be 0, 1, or 2.

**rule_array**

Direction: Input  
Type: String
Zero, one or two keywords that supply control information to the callable service. Each keyword must be in 8 bytes of contiguous storage, left-justified and padded on the right with blanks. The keywords are shown in Table 48.

The first keyword is the algorithm. If no algorithm is specified, the system default algorithm is used. If no algorithm is specified on a CDMF only system and either a double- or triple-length DATA key is specified, the token is marked DES. The algorithm keyword applies only when the desired output token is of key form OP and key type IMPORTER, EXPORTER, or DATA. For key form IM or any other key type, specifying DES or CDMF causes an error.

The second keyword is optional and specifies that the output key token be marked as an NOCV-KEK.

Table 48. Keywords for Multiple Secure Key Import Rule Array Control Information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm (optional)</td>
<td></td>
</tr>
<tr>
<td>CDMF</td>
<td>The output key identifier is to be a CDMF token. For a DATA key of length 16 or 24, you may not specify CDMF. CDMF is only supported on CCF systems.</td>
</tr>
<tr>
<td>AES</td>
<td>The output key identifier is to be a AES token.</td>
</tr>
<tr>
<td>DES</td>
<td>The output key identifier is to be a DES token. This is the default.</td>
</tr>
<tr>
<td>NOCV Choice (optional)</td>
<td></td>
</tr>
<tr>
<td>NOCV-KEK</td>
<td>The output token is to be marked as an NOCV-KEK. This keyword only applies if key form is OP and key type is IMPORTER, EXPORTER or IMP-PKA. For key form IM or any other key type, specifying NOCV-KEK causes an error.</td>
</tr>
</tbody>
</table>

clear_key_length

Direction: Input
Type: Integer

The clear_key_length specifies the length of the clear key value to import in bytes. For AES keys, this length must be 16-, 24-, or 32-bytes. For DES keys, this length must be 8-, 16- or 24-bytes.

clear_key

Direction: Input
Type: String

The clear_key specifies the AES or DES clear key value to import.

key_type

Direction: Input
Type: 8 Character String

The type of key you want to encipher under the master key or an importer key. Specify an 8-byte field that must contain a keyword from this list or the keyword TOKEN. For types with fewer than 8 characters, the type should be padded on the right with blanks. If the key type is TOKEN, ICSF determines the key type from the control vector (CV) field in the internal key token provided in the imported_key_identifier parameter. When key_type is TOKEN, ICSF does not check for the length of the key but uses the clear_key_length parameter to determine the length of the key.
Multiple Secure Key Import (CSNBSKM)

Key type values for the Multiple Secure Key Import callable service are:
CIPHER, CVARDEC, CVARENC, CVARPINE, CVARXCVL, CVARXCVR, DATA, DATAM, DATAMV, DATAXLAT, DECIPHER, ENCIIPHER, EXPORTER, IKEYXLAT, IMPORTER, IMP-PKA, IPINENC, MAC, MACVER, OKEYXLAT, OPINENC, PINGEN and PINVER. For information on the meaning of the key types, see Table 2 on page 21.

**key_form**

Direction: Input
Type: 4 Character String

The key form you want to generate. Enter a 4-byte keyword specifying whether the key should be enciphered under the master key (OP) or the importer key-encrypting key (IM). The keyword must be left-justified and padded with blanks. Valid DES keyword values are OP for encryption under the master key or IM for encryption under the importer key-encrypting key. If you specify IM, you must specify an importer key-encrypting key in the **key_encrypting_key_identifier** parameter. For a **key_type** of IMP-PKA, this service supports only the OP **key_form**.

The only valid AES keyword value is OP.

**key_encrypting_key_identifier**

Direction: Input/Output
Type: String

A 64-byte string internal key token or key label of a DES importer key-encrypting key. This parameter is ignored for AES secure keys.

**imported_key_identifier_length**

Direction: Input/Output
Type: Integer

The byte length of the **imported_key_identifier** parameter. This must be at least 64.

**imported_key_identifier**

Direction: Input/Output
Type: String

A 64-byte string that is to receive the output key token. If OP is specified in the **key_form** parameter, the service returns an internal key token. If IM is specified in the **key_form** parameter, the service returns an external key token. On input, this parameter is ignored except when the **key_type** is TOKEN. If you specify a **key_type** of TOKEN, then this field contains a valid token of the **key_type** you want to encipher. See **key_type** for a list of valid key types. Appendix B, “Key Token Formats,” on page 597 describes the key tokens.

Note that for a DATA key of length 16 or 24, no reference will be made to the data encryption algorithm bits or to the system's default algorithm; the token will be marked DES.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

On CCF systems, to generate double-length DATAM and DATAMV keys in the importable form, the ANSI system keys must be installed in the CKDS.
Multiple Secure Key Import (CSNBSKM)

CDMF is only supported on CCF systems.

With a PCIXCC, CEX2C, or CEX3C, creation of a DES NOCV key-encrypting key is only available for standard IMPORTERs and EXPORTERs.

On an IBM @server zSeries 990, if `key_form` of the DES key is IM and the `key_encrypting_key_identifier` is a NOCV KEK, then the NOCV IMPORTER access control point must be enabled in the PCIXCC to use the function.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 49. Multiple secure key import required hardware*

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>Only control vectors and key types supported by the Cryptographic Coprocessor Feature will be valid when importing a triple-length key.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td>ICSF routes the request to a PCI Cryptographic Coprocessor if the control vector of a supplied internal token cannot be processed on the Cryptographic Coprocessor Feature, or if the key type is not valid for the Cryptographic Coprocessor Feature. DATAC is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td><code>Key_type</code> DATAXLAT is not supported. CDMF keyword is not supported. DATA and KEK tokens are not marked with the system encryption algorithm.</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td><code>Key_type</code> DATAXLAT is not supported. CDMF keyword is not supported. DATA and KEK tokens are not marked with the system encryption algorithm. Secure AES keys require the Nov. 2008 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td><code>Key_type</code> DATAXLAT is not supported. CDMF keyword is not supported. DATA and KEK tokens are not marked with the system encryption algorithm. Secure AES keys require the Nov. 2008 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PKA Decrypt (CSNDPKD and CSNFPKD)

Use this service to decrypt (unwrap) a formatted key value. The service unwraps the key, deformats it, and returns the deformatted value to the application in the clear. PKCS 1.2 and ZERO-PAD formatting is supported. For PKCS 1.2, the decrypted data is examined to ensure it meets RSA DSI PKCS #1 block type 2 format specifications. ZERO-PAD is only supported for external RSA clear private keys.

This service allows the use of clear or encrypted RSA private keys. If an external clear key token is used, the master keys are not required to be installed in any cryptographic coprocessor and PKA callable services does not have to be enabled. Requests are routed to a PCICA if available when a clear key token is used.

This callable service supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSNFPKD.

Format

```call
CALL CSNDPKD(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    PKA_enciphered_keyvalue_length,
    PKA_enciphered_keyvalue,
    data_structure_length,
    data_structure,
    PKA_key_identifier_length,
    PKA_key_identifier,
    target_keyvalue_length,
    target_keyvalue
)
```

Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. [Appendix A, "ICSF and TSS Return and Reason Codes"] lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. [Appendix A, "ICSF and TSS Return and Reason Codes"] lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer
PKA Decrypt (CSNDPKD and CSNFPKD)

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

exit_data
Direction: Input/Output  Type: String

The data that is passed to the installation exit.

rule_array_count
Direction: Input  Type: Integer

The number of keywords you supplied in the rule_array parameter. This value must be 1.

rule_array
Direction: Input  Type: String

The keyword that provides control information to the callable service. The keyword is left-justified in an 8-byte field and padded on the right with blanks.

Table 50. Keywords for PKA Decrypt

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recovery Method (required)</td>
<td>specifies the method to use to recover the key value.</td>
</tr>
<tr>
<td>PKCS-1.2</td>
<td>RSA DSI PKCS #1 block type 02 will be used to recover the key value.</td>
</tr>
<tr>
<td>ZERO-PAD</td>
<td>The input PKA_enciphered_keyvalue is decrypted using the RSA private key. The entire result (including leading zeros) will be returned in the target_keyvalue field. The PKA_key_identifier must be an external RSA token or the labelname of a external token. This keyword requires requires May 2004 or later version of Licensed Internal Code (LIC) or a z890. This support on the PCICA does not require LIC code updates.</td>
</tr>
</tbody>
</table>

PKA_enciphered_keyvalue_length
Direction: Input  Type: integer

The length of the PKA_enciphered_keyvalue parameter in bytes. The maximum size that you can specify is 512 bytes. The length should be the same as the modulus length of the PKA_key_identifier.

PKA_enciphered_keyvalue
Direction: Input  Type: String

This field contains the key value protected under an RSA public key. This byte-length string is left-justified within the PKA_enciphered_keyvalue parameter.
PKA Decrypt (CSNDPKD and CSNFPKD)

**data_structure_length**
Direction: Input  
Type: Integer

The value must be 0.

**data_structure**
Direction: Input  
Type: String

This field is currently ignored.

**PKA_key_identifier_length**
Direction: Input  
Type: Integer

The length of the **PKA_key_identifier** parameter. When the **PKA_key_identifier** is a key label, this field specifies the length of the label. The maximum size that you can specify is 3500 bytes.

**PKA_key_identifier**
Direction: Input  
Type: String

An internal RSA private key token, the label of an internal RSA private key token, or an external RSA private key token containing a clear RSA private key in modulus-exponent or Chinese Remainder format. The corresponding public key was used to wrap the key value.

**target_keyvalue_length**
Direction: Input/Output  
Type: Integer

The length of the **target_keyvalue** parameter. The maximum size that you can specify is 512 bytes. On return, this field is updated with the actual length of **target_keyvalue**.

If ZERO-PAD is specified, this length will be the same as the **PKA_enciphered_keyvalue_length** which is equal to the RSA modulus byte length.

**target_keyvalue**
Direction: Output  
Type: String

This field will contain the decrypted, deformatted key value. If ZERO-PAD is specified, the decrypted key value, including leading zeros, will be returned.

**Restrictions**

The exponent of the RSA public key must be odd.

Access control checking will not be performed in the PCI Cryptographic Coprocessor when a clear external key token is supplied.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.
PKA Decrypt (CSNDPKD and CSNFPKD)

The RSA private key must be enabled for key management functions.

The hardware configuration sets the limit on the modulus size of keys for key management; thus, this service will fail if the RSA key modulus bit length exceeds this limit.

Routing of requests to coprocessors for systems with CCFs

This service examines the RSA key specified in the PKA_key_identifier parameter to determine how to route the request.

- If the modulus bit length is less than 512 bits, or if the key is a X'02' form modulus-exponent private key, ICSF routes the request to the Cryptographic Coprocessor Feature.
- If the key is a X'08' form CRT private key or a retained private key, the service routes the request to a PCI Cryptographic Coprocessor.
- In the case of a retained key, the service routes the request to the specific PCI Cryptographic Coprocessor in which the key is retained.
- If the key is a modulus-exponent form private key with a private section ID of X'06', then the service routes the request as follows:
  - Since the key must be a key-management key, if the KMMK is equal to the SMK on the Cryptographic Coprocessor Feature, the PKA decrypt service uses load balancing to route the request to either a Cryptographic Coprocessor Feature or to an available PCI Cryptographic Coprocessor.
  - If the KMMK is not equal to the SMK on the Cryptographic Coprocessor Feature, the request must be processed on a PCI Cryptographic Coprocessor. If there is no PCI Cryptographic Coprocessor online, the request will fail.
- If the key is an external clear key, the request is routed in this order of preference.
  - PCICA
  - PCICC
  - CCF

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
### Table 51. PKA decrypt required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>ICSF routes the request to the Cryptographic Coprocessor Feature if the modulus bit length is less than 512 bits, or if the key is a X'02' form modulus-exponent private key.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td>The ZERO-PAD keyword is not supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA keys with moduli greater than 1024-bit length are not supported.</td>
</tr>
<tr>
<td></td>
<td>PCI Cryptographic Coprocessor</td>
<td>This service routes the request to the PCI Cryptographic Coprocessor in which the key is retained if the key is a X'08' form CRT private key or a retained private key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The ZERO-PAD keyword is not supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA keys with moduli greater than 2048-bit length are not supported.</td>
</tr>
<tr>
<td></td>
<td>PCI Cryptographic Accelerator</td>
<td>Only clear RSA private keys are supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The ZERO-PAD keyword is not supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA keys with moduli greater than 2048-bit length are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>Old RSA private tokens encrypted under the CCF KMMK are not usable on the PCIXCC/CEX2C if the KMMK was not same as the ASYM-MK.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td>RSA keys with moduli greater than 2048-bit length are not supported.</td>
</tr>
<tr>
<td></td>
<td>PCI Cryptographic Accelerator</td>
<td>Only clear RSA private keys are supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA keys with moduli greater than 2048-bit length are not supported.</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Old RSA private tokens encrypted under the CCF KMMK are not usable on the CEX2C if the KMMK was not same as the ASYM-MK.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td></td>
<td>Crypto Express2 Accelerator</td>
<td>Only clear RSA private keys are supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA keys with moduli greater than 2048-bit length are not supported.</td>
</tr>
</tbody>
</table>
Table 51. PKA decrypt required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Old RSA private tokens encrypted under the CCF KMMK are not usable on the CEX2C or CEX3C if the KMMK was not same as the ASYM-MK.</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td></td>
<td>Crypto Express2 Accelerator</td>
<td>Only clear RSA private keys are supported.</td>
</tr>
<tr>
<td></td>
<td>Crypto Express3 Accelerator</td>
<td>RSA keys with moduli greater than 2048-bit length are not supported.</td>
</tr>
</tbody>
</table>

PKA Decrypt (CSNDPKD and CSNFPKD)

This callable service encrypts a supplied clear key value under an RSA public key. The rule array keyword specifies the format of the key prior to encryption.

On the z990 and if the ZERO-PAD or MRP keyword is specified, this service is routed to a PCI Cryptographic Accelerator.

This callable service supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSNFPKE.

Format

```
call csndpke(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    keyvalue_length,
    keyvalue,
    data_structure_length,
    data_structure,
    pka_key_identifier_length,
    pka_key_identifier,
    pka_enciphered_keyvalue_length,
    pka_enciphered_keyvalue)
```

Parameters

return_code

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.
PKA Encrypt (CSNDPKE and CSNFPKE)

**reason_code**
Direction: Output  Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit_data_length**
Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**
Direction: Input/Output  Type: String

The data that is passed to the installation exit.

**rule_array_count**
Direction: Input  Type: Integer

The number of keywords you supplied in the rule_array parameter. This value can be 1 or 2.

**rule_array**
Direction: Input  Type: String

A keyword that provides control information to the callable service. The keyword is left-justified in an 8-byte field and padded on the right with blanks.

Table 52. Keywords for PKA Encrypt

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formatting Method (required)</td>
<td>specifies the method to use to format the key value prior to encryption.</td>
</tr>
<tr>
<td>PKCS-1.2</td>
<td>RSA DSI PKCS #1 block type 02 format will be used to format the supplied key value.</td>
</tr>
<tr>
<td>ZERO-PAD</td>
<td>The key value will be padded on the left with binary zeros to the length of the PKA key modulus. The exponent of the public key must be odd.</td>
</tr>
<tr>
<td>MRP</td>
<td>The key value will be padded on the left with binary zeros to the length of the PKA key modulus. The RSA public key may have an even or odd exponent. This keyword requires May 2004 or later version of Licensed Internal Code (LIC) or a z890. For PCICAs, the LIC code update is not required.</td>
</tr>
<tr>
<td>Key Rule (Optional)</td>
<td></td>
</tr>
<tr>
<td>KEYIDENT</td>
<td>This indicates that the value in the keyvalue field is the label of clear tokens in the CKDS. The keyvalue_length must be 64.</td>
</tr>
</tbody>
</table>
PKA Encrypt (CSNDPKE and CSNFPKE)

**keyvalue_length**

Direction: Input  
Type: Integer

- The length of the `keyvalue` parameter. The maximum field size is 512 bytes.
- The actual maximum size depends on the modulus length of `PKA_key_identifier` and the formatting method you specify in the `rule_array` parameter. When key rule KEYIDENT is specified, then the `keyvalue_length` parameter is required to be 64 bytes.

**keyvalue**

Direction: Input  
Type: String

- This field contains the supplied clear key value to be encrypted under the `PKA_key_identifier`. When key rule KEYIDENT is specified, the `keyvalue` parameter is assumed to contain a label for a valid CKDS clear key token.

**data_structure_length**

Direction: Input  
Type: Integer

- This value must be 0.

**data_structure**

Direction: Input  
Type: String

- This field is currently ignored.

**PKA_key_identifier_length**

Direction: Input  
Type: Integer

- The length of the `PKA_key_identifier` parameter. When the `PKA_key_identifier` is a key label, this field specifies the length of the label. The maximum size that you can specify is 3500 bytes.

**PKA_key_identifier**

Direction: Input  
Type: String

- The RSA public or private key token or the label of the RSA public or private key to be used to encrypt the supplied key value.

**PKA_enciphered_keyvalue_length**

Direction: Input/Output  
Type: integer

- The length of the `PKA_enciphered_keyvalue` parameter in bytes. The maximum size that you can specify is 512 bytes. On return, this field is updated with the actual length of `PKA_enciphered_keyvalue`.
- This length should be the same as the modulus length of the `PKA_key_identifier`.

**PKA_enciphered_keyvalue**

Direction: Output  
Type: String
PKA Encrypt (CSNDPKE and CSNFPKE)

This field contains the key value protected under an RSA public key. This byte-length string is left-justified within the PKA_enciphered_keyvalue parameter.

Restrictions

The exponent for RSA public keys must be odd. When the modulus is greater than 2048, the public key exponent must be 3 or 65537.

Usage Notes

- SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.
- For RSA DSI PKCS #1 formatting, the key value length must be at least 11 bytes less than the modulus length of the RSA key.
- The hardware configuration sets the limit on the modulus size of keys for key management; thus, this service will fail if the RSA key modulus bit length exceeds this limit.
- The key value to be encrypted must be smaller than the modulus in the PKA_key_identifier.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 53. PKA encrypt required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>The MRP keyword is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>PCI Cryptographic Coprocessor</td>
<td>If the modulus bit length of the key specified in the PKA_key_identifier parameter is greater than 1024, the request is routed to the PCICC.</td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>Routed to a PCICA if one is available (ZERO-PAD and MRP only).</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td>RSA keys with moduli greater than 2048-bit length are not supported.</td>
</tr>
<tr>
<td></td>
<td>PCI Cryptographic Accelerator</td>
<td>PKCS-1.2 keyword not supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA keys with moduli greater than 2048-bit length are not supported.</td>
</tr>
</tbody>
</table>
Table 53. PKA encrypt required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Routed to a CEX2A if one is available (ZERO-PAD and MRP only).</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td></td>
<td>Crypto Express2 Accelerator</td>
<td>PKCS-1.2 keyword not supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA keys with moduli greater than 2048-bit length are not supported.</td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Routed to a CEX2A or CEX3A if one is available (ZERO-PAD and MRP only).</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td></td>
<td>Crypto Express2 Accelerator</td>
<td>PKCS-1.2 keyword not supported.</td>
</tr>
<tr>
<td></td>
<td>Crypto Express3 Accelerator</td>
<td>RSA keys with moduli greater than 2048-bit length are not supported.</td>
</tr>
</tbody>
</table>

Prohibit Export (CSNBPEX)

Use this service to modify an exportable internal DES key token so that it cannot be exported.

Format

```
CALL CSNBPEX(
   return_code,
   reason_code,
   exit_data_length,
   exit_data,
   key_identifier)
```

Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

Direction: Output  
Type: Integer
Prohibit Export (CSNBPEX)

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

exit_data_length

Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

exit_data

Direction: Input/Output  Type: String

The data that is passed to the installation exit.

key_identifier

Direction: Input/Output  Type: String

A 64-byte string variable containing the internal key token to be modified. The returned key_identifier will be encrypted under the current master key.

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 54. Prohibit export required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>PCI Cryptographic Coprocessor</td>
<td>On a PCI Cryptographic Coprocessor, the Prohibit Export service does not support NOCV key-encrypting keys, or DATA, DATAM, DATAMV, MAC, or MACVER keys with standard control vectors (for example, control vectors supported by the Cryptographic Coprocessor Feature).</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>DATA keys are not supported. Old, internal DATAM and DATAMV keys are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td>DATA keys are not supported. Old, internal DATAM and DATAMV keys are not supported.</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>DATA keys are not supported. Old, internal DATAM and DATAMV keys are not supported.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td>DATA keys are not supported. Old, internal DATAM and DATAMV keys are not supported.</td>
</tr>
</tbody>
</table>
Prohibit Export (CSNBPEXX)

Use the prohibit export extended callable service to change the external token of a cryptographic key in exportable DES key token form so that it can be imported at the receiver node and is non-exportable from that node. You cannot prohibit export of DATA keys.

The inputs are an external token of the key to change in the source_key_token parameter and the label or internal token of the exporter key-encrypting key in the KEK_key_identifier parameter.

CSNBPEXX is a variation of the prohibit export service CSNBPEX, which supports changing an internal token.

Format

```
CALL CSNBPEXX(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    source_key_token,
    KEK_key_identifier)
```

Parameters

- **return_code**
  
  Direction: Output  
  Type: Integer  
  
  The return code specifies the general result of the callable service. [Appendix A, “ICSF and TSS Return and Reason Codes”](#) lists the return codes.

- **reason_code**
  
  Direction: Output  
  Type: Integer  
  
  The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. [Appendix A, “ICSF and TSS Return and Reason Codes”](#) lists the reason codes.

- **exit_data_length**
  
  Direction: Input/Output  
  Type: Integer  
  
  The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

- **exit_data**
  
  Direction: Input/Output  
  Type: String  
  
  The data that is passed to the installation exit.
Prohibit Export Extended (CSNBPEXX)

**source_key_token**

Direction: Input/Output  
Type: String

A 64-byte string of an external token of a key to change. It is in exportable form.

**KEK_key_identifier**

Direction: Input/Output  
Type: String

A 64-byte string of an internal token or label of the exporter KEK used to encrypt the key contained in the external token specified in the previous parameter.

**Restrictions**

This callable service does not support version X'10' external DES key tokens (RKX key tokens).

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 55. Prohibit export extended required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>External MACD keys are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>External MACD keys are not supported.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>External MACD keys are not supported.</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

Random Number Generate (CSNBRNG, CSNERNG, CSNBRNGL and CSNERNGL)

The callable service uses the cryptographic feature to generate a random number. The foundation for the random number generator is a time variant input with a very low probability of recycling.
Random Number Generate (CSNBRNG)

There are two forms of the Random Number Generate callable service. One version returns an 8-byte random number. The second version allows the caller to specify the length of the random number.

Note: Random Number Generate on a z800 or z900 server requires the symmetric-keys master key to be set prior to using the service.

This callable service supports invocation in AMODE(64). The callable service names for AMODE(64) invocation are CSNERNG and CSNERNGL.

Format

CALL CSNBRNG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    form,
    random_number )

CALL CSNBRNGL(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    reserved_length,
    reserved,
    random_number_length,
    random_number )

Parameters

return_code
Direction: Output Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code
Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

exit_data_length
Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.
**Random Number Generate (CSNBRNG)**

**exit_data**
Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**form**
Direction: Input  
Type: Character string

The 8-byte keyword that defines the characteristics of the random number should be left-justify and pad on the right with blanks. The keywords are listed in Table 56.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVEN</td>
<td>Generate a 64-bit random number with even parity in each byte.</td>
</tr>
<tr>
<td>ODD</td>
<td>Generate a 64-bit random number with odd parity in each byte.</td>
</tr>
<tr>
<td>RANDOM</td>
<td>Generate a 64-bit random number.</td>
</tr>
</tbody>
</table>

Parity is calculated on the 7 high-order bits in each byte and is presented in the low-order bit in the byte.

**rule_array_count**
Direction: Input  
Type: Integer

The number of keywords you are supplying in the *rule_array* parameter. The value must be one.

**rule_array**
Direction: Input  
Type: String

The keyword that provides control information to the callable service. The recovery method is the method to use to recover the symmetric key. The keyword is left-justified in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parity of the random number bytes (required)</td>
<td></td>
</tr>
<tr>
<td>EVEN</td>
<td>Generate a random number with even parity in each byte. Its length is the random_number_length.</td>
</tr>
<tr>
<td>ODD</td>
<td>Generate a random number with odd parity in each byte. Its length is the random_number_length.</td>
</tr>
<tr>
<td>RANDOM</td>
<td>Generate a random number. Its length is the random_number_length.</td>
</tr>
</tbody>
</table>

**reserved_length**
Direction: Input  
Type: Integer
Random Number Generate (CSNBRNG)

This parameter must be zero.

reserved

Direction: Input  Type: Integer

This parameter is ignored.

random_number_length

Direction: Input/Output  Type: Integer

This parameter contains the desired length of the random_number that is returned by the CSNBRNGL callable service. The minimum value is 1 byte; the maximum value is 8192 bytes.

random_number

Direction: Output  Type: String

The generated number returned by the CSNBRNG callable service is stored in an 8-byte variable.

The generated number returned by the CSNBRNGL callable service is stored in a variable that is at least random_number_length bytes long.

Usage Notes

The CSNBRNGL callable service returns a value under the following conditions:

- The server has the cryptographic coprocessor that supports CSNBRNGL and the coprocessor creates the random number with the desired length. This requires a CEX2C or CEX3C with a version of the licensed internal code (LIC) that supports the RNGL verb.
- The server has the cryptographic coprocessor that processes CSNBRNG requests. In this case, the CSNBRNGL callable service calls the processor to create the random number with the desired length, 8 bytes at a time.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>PCI X Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Random Number Generate (CSNBRNG)

Table 58. Random number generate required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

Remote Key Export (CSNDRKX)

This callable service uses the trusted block to generate or export DES keys for local use and for distribution to an ATM or other remote device. RKX uses a special structure to hold encrypted symmetric keys in a way that binds them to the trusted block and allows sequences of RKX calls to be bound together as if they were an atomic operation.

Format

```
CALL CSNDRKX(
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    trusted_block_length,  
    trusted_block_identifier,  
    certificate_length,  
    certificate,  
    certificate_parms_length,  
    certificate_parms,  
    transport_key_length,  
    transport_key_identifier,  
    rule_id_length,  
    rule_id,  
    importer_key_length,  
    importer_key_identifier,  
    source_key_length,  
    source_key_identifier,  
    asym_encrypted_key_length,  
    asym_encrypted_key,  
    sym_encrypted_key_length,  
    sym_encrypted_key,  
    extra_data_length,  
    extra_data,  
    key_check_parameters_length,  
    key_check_parameters,  
    key_check_length,  
    key_check_value)
```

Parameters

- **return_code**
  - Direction: Output
  - Type: Integer
Remote Key Export (CSNDRKX)

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code
Direction: Output  Type: Integer

The reason code specifies the specific results of processing. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

exit_data_length
Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

exit_data
Direction: Input/Output  Type: String

The data that is passed to the installation exit.

rule_array_count
Direction: Input  Type: Integer

The number of keywords you are supplying in the rule_array parameter. This number must be 0.

rule_array
Direction: Input  Type: String

Specifies a string variable containing an array of keywords. Currently no rule_array keywords are defined for this service, but you must still specify this parameter.

trusted_block_length
Direction: Input  Type: Integer

Specifies the number of bytes in the trusted_block_identifier parameter. The maximum length is 3500 bytes.

trusted_block_identifier
Direction: Input  Type: String

Specifies a trusted block label or trusted block token of an internal/complete trusted block constructed by the service, which is used to validate the public key certificate (certificate) and to define the rules for key generation and export.

certificate_length
Direction: Input  Type: Integer
Remote Key Export (CSNDRKX)

Specifies the number of bytes in the certificate parameter. The maximum is 5000 bytes.

If the certificate_length is zero and the trusted block's Asymmetric Encrypted Output Key Format indicates no asymmetric key output, this service will not attempt to use or validate the certificate in any way. Consequently, the output parameter asym_encrypted_key_length will contain zero and output parameter asym_encrypted_key will not be changed from its input content.

If the certificate_length is zero and the trusted block's Asymmetric Encrypted Output Key Format indicates PKCS1.2 output format or RSAOAEP output format, this service will exit with an error.

If the certificate_length is non-zero and the trusted block's Asymmetric Encrypted Output Key Format indicates no asymmetric key output, this service will fail.

certificate
Direction: Input
Type: String

Contains a public-key certificate. The certificate must contain the public key modulus and exponent in binary_form, as well as a digital signature. The signature in the certificate will be verified using the root public key that is in the trusted block supplied in trusted_block_identifier parameter.

certificate_parms_length
Direction: Input
Type: Integer

Contains the number of bytes in the certificate_parms parameter. The length must be 36 bytes.

certificate_parms
Direction: Input
Type: String

Contains a structure provided by the caller used for identifying the location and length of values within the certificate in parameter certificate. For each of the values used by RKX, the structure contains offsets from the start of the certificate and length in bytes. It is the responsibility of the calling application program to provide these values. This method gives the greatest flexibility to support different certificate formats. The structure has this layout:

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>Offset of modulus</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Length of modulus</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>Offset of public exponent</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>Length of public exponent</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>Offset of digital signature</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>Length of digital signature</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>Identifier for the hash algorithm used</td>
</tr>
</tbody>
</table>
Remote Key Export (CSNDRKX)

Table 59. Structure of values used by RKX (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>1</td>
<td>Identifier for the digital hash formatting method</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 01 - PKCS-1.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 02 - PKCS-1.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 03 - X9.31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 04 - ISO-9796</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 05 - ZERO-PAD</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>Reserved - must be filled with 0x00 bytes</td>
</tr>
<tr>
<td>28</td>
<td>4</td>
<td>Offset of first byte of certificate data hashed to compute the digital signature</td>
</tr>
<tr>
<td>32</td>
<td>4</td>
<td>Length of the certificate data hashed to compute the digital signature</td>
</tr>
</tbody>
</table>

The modulus, exponent, and signature values are right-justified and padded on the left with binary zeros if necessary.

These values are defined for the hash algorithm identifier at offset 24 in the structure.

Table 60. Values defined for hash algorithm identifier at offset 24 in the structure for remote key export

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0X01</td>
<td>SHA-1</td>
</tr>
<tr>
<td>0X02</td>
<td>MD5</td>
</tr>
<tr>
<td>0X03</td>
<td>RIPEMD-160</td>
</tr>
</tbody>
</table>

transport_key_length
Direction: Input
Type: Integer
Contains the number of bytes in the transport_key_identifier parameter.

transport_key_identifier
Direction: Input
Type: String
Contains a label of an internal key token, or an RKX token for a Key Encrypting Key (KEK) that is used to encrypt a key exported by the RKX service. A transport key will not be used to encrypt a generated key.

For flag bit0=1 (export existing key) within Rule section and parameter rule_id = Rule section ruleID, the transport_key_identifier encrypts the exported version of the key received in parameter source_key_identifier. The service can distinguish between the internal key token or RKX key token by virtue of the version number at offset 0x04 contained in the key token and the flag value at offset 0x00 as follows:
Remote Key Export (CSNDRKX)

Table 61. Transport_key_identifier used by RKX

<table>
<thead>
<tr>
<th>Flag Byte Offset 00</th>
<th>Version Number Offset 04</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>0x00</td>
<td>Internal DES key token version 0</td>
</tr>
<tr>
<td>0x02</td>
<td>0x10</td>
<td>RKX Key token (Flag byte 0x02 indicates external key token)</td>
</tr>
</tbody>
</table>

**rule_id_length**

Direction: Input  
Type: Integer

Contains the number of bytes in the rule_id parameter. The value must be 8.

**rule_id**

Direction: Input  
Type: String

Specifies the rule in the trusted block that will be used to control key generation or export. The trusted block can contain multiple rules, each of which is identified by a rule ID value.

**importer_key_length**

Direction: Input  
Type: Integer

Contains the number of bytes in the importer_key_identifier parameter. It must be zero if the Generate/Export flag in the rule section (section 0x12) of the Trusted Block is a zero, indicating a new key is to be generated.

**importer_key_identifier**

Direction: Input  
Type: String

Contains a key token or key label for the IMPORTER key-encrypting key that is used to decipher the key passed in parameter source_key_identifier. It is unused if either RKX is being used to generate a key, or if the source_key_identifier is an RKX key token.

**source_key_length**

Direction: Input  
Type: Integer

Contains the number of bytes in the source_key_identifier parameter. The parameter must be 0 if the trusted block Rule section ruleID = rule_id parameter and the flag bit0 = 0 (Generate new key).

The parameter must be 64 if the trusted block Rule section has a flag bit0 = 1 (Export existing key).

**source_key_identifier**

Direction: Input  
Type: String

Contains a label of a single or double length external or internal key token or an RKX key token for a key to be exported. It must be empty (source_key_length=0) if RKX is used to generate a new key. The service
Remote Key Export (CSNDRKX)

examines the key token to determine which form has been provided. This parameter is known as the source_key_identifier in other callable services.

<table>
<thead>
<tr>
<th>Flag Byte Offset 00</th>
<th>Version Number Offset 04</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0X01</td>
<td>0X00</td>
<td>Internal DES key token version 0</td>
</tr>
<tr>
<td>0X02</td>
<td>0X00</td>
<td>External DES key token version 0</td>
</tr>
<tr>
<td>0X02</td>
<td>0X01</td>
<td>External DES key token version 1</td>
</tr>
<tr>
<td>0X02</td>
<td>0X10</td>
<td>RKX Key token (Flag byte 0x02 indicates external key token)</td>
</tr>
</tbody>
</table>

asym_encrypted_key_length
Direction: Input/Output Type: Integer

The length of the asym_encrypted_key parameter. On input, it is the length of the storage to receive the output. On output, it is the length of the data returned in the asym_encrypted_key parameter. The maximum length is 512 bytes.

asym_encrypted_key
Direction: Output Type: String

The contents of this field is ignored on input. A string buffer RKX will use to return a generated or exported key that is encrypted under the public (asymmetric) key passed in parameter certificate. An error will be returned if the caller's buffer is too small to hold the value that would be returned.

sym_encrypted_key_length
Direction: Input/Output Type: Integer

On input, the sym_encrypted_key_length parameter is an integer variable containing the number of bytes in the sym_encrypted_key field. On output, that value in sym_encrypted_key_length is replaced with the length of the key returned in sym_encrypted_key field.

sym_encrypted_key
Direction: Output Type: String

Sym_encrypted_key is the string buffer RKX uses to return a generated or exported key that is encrypted under the key-encrypting key passed in the transport_key_identifier parameter. The value returned will be 64 bytes. An error will be returned if the caller's buffer is smaller than 64 bytes, and so too small to hold the value that would be returned. The sym_encrypted_key may be an RKX key token or a key token depending upon the value of the Symmetric Encrypted Output Key Format value of the Rule section within the trusted_block_identifier parameter.

extra_data_length
Direction: Input Type: Integer
Remote Key Export (CSNDRKX)

Contains the number of bytes of data in the extra_data parameter. It must be zero if the output format for the RSA-encrypted key in asym_encrypted_key is anything but RSAOEAP. The maximum size is 1024 bytes.

extra_data

Direction: Input Type: String

Can be used in the OAEP key wrapping process. Extra_data is optional and is only applicable when the output format for the RSA-encrypted key returned in asym_encrypted_key is RSAOEAP.

Note: RSAOAEP format is specified in the rule in the trusted block.

key_check_parameters_length

Direction: Input Type: Integer

Contains the number of bytes in the key_check_parameters parameter. Currently, none of the defined key check algorithms require any key check parameters, so this field must specify 0.

key_check_parameters

Direction: Input Type: String

Contains parameters that are required to calculate a key check value parameter, which will be returned in key_check_value. Currently, none of the defined key check algorithms require any key check parameters, but you must still specify this parameter.

key_check_length

Direction: Input/Output Type: Integer

On input this parameter contains the number of bytes in the key_check_value parameter. On output, the value is replaced with the length of the key check value returned in the key_check_value parameter. The length depends on the key-check algorithm identifier. See Table 260 on page 621.

key_check_value

Direction: Output Type: String

Used by RKX to return a key check value that calculates on the generated or exported key. Values in the rule specified with rule_id can specify a key check algorithm that should be used to calculate this output value.

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
Remote Key Export (CSNDRKX)

Table 63. Remote key export required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 990</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 800</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM @server z9 EC</td>
<td>Crypto Express 2 Coprocessor</td>
<td>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

Secure Key Import (CSNBSKI)

Use the secure key import callable service to encipher a single-length or double-length clear key under the system master key (DES or SYM-MK) or under an importer key-encrypting key. The clear key can then be imported as any of the possible key types. This service does not adjust key parity.

The callable service can execute only when ICSF is in special secure mode, which is described in "Special Secure Mode" on page 10.

To import double-length and triple-length DATA keys, or double-length MAC, MACVER, CIPHER, DECIPHER and ENCIPHER keys, use the multiple secure key import (CSNBSKM) callable service. See "Multiple Secure Key Import (CSNBSKM)" on page 161.

To import AES DATA keys, use the multiple secure key import service "Multiple Secure Key Import (CSNBSKM)" on page 161.

Format

```
CALL CSNBSKI(  
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    clear_key,  
    key_type,  
    key_form,  
    importer_key_identifier,  
    key_identifier)
```
Secure Key Import (CSNBSKI)

Parameters

return_code
Direction: Output
Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

reason_code
Direction: Output
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

exit_data_length
Direction: Input/Output
Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

exit_data
Direction: Input/Output
Type: String

The data that is passed to the installation exit.

clear_key
Direction: Input
Type: String

The clear key to be enciphered. Specify a 16-byte string (clear key value). For single-length keys, the value must be left-justified and padded with zeros. For effective single-length keys, the value of the right half must equal the value of the left half. For double-length keys, specify the left and right key values.

Note: For key types that can be single or double-length, a single length encrypted key will be generated if a clear_key value of zeros is supplied.

key_type
Direction: Input
Type: Character string

The type of key you want to encipher under the master key or an importer key. Specify an 8-byte field that must contain a keyword from this list or the keyword TOKEN. If the key type is TOKEN, ICSF determines the key type from the CV in the key_identifier parameter.

Key type values for the Secure Key Import callable service are: CIPHER, CVARDEC, CVARENC, CVARPINE, CVARXCVL, CVARXCVR, DATA, DATAXLAT, DECIPHER, ENCIIPHER, EXPORTER, IKEYXLAT, IMPORTER, IMP-PKA, IPINENC, MAC, MACVER, OKEYXLAT, OPINENC, PINGEN and PINVER. For information on the meaning of the key types, see Table 2 on page 21.
key_form

Direction: Input
Type: Character string

The key form you want to generate. Enter a 4-byte keyword specifying whether the key should be enciphered under the master key (OP) or the importer key-encrypting key (IM). The keyword must be left-justified and padded with blanks. Valid keyword values are OP for encryption under the master key or IM for encryption under the importer key-encrypting key. If you specify IM, you must specify an importer key-encrypting key in the importer_key_identifier parameter. For a key_type of IMP-PKA, this service supports only the OP key_form.

importer_key_identifier

Direction: Input/Output
Type: String

The importer key-encrypting key under which you want to encrypt the clear key. Specify either a 64-byte string of the internal key format or a key label. If you specify IM for the key_form parameter, the importer_key_identifier parameter is required.

key_identifier

Direction: Input/Output
Type: String

The generated encrypted key. The parameter is a 64-byte string. The callable service returns either an internal key token if you encrypted the clear key under the master key (key_form was OP); or an external key token if you encrypted the clear key under the importer key-encrypting key (key_form was IM).

If the imported key_type is IMPORTER or EXPORTER and the key_form is OP, the key_identifier parameter changes direction to both input and output. If the application passes a valid internal key token for an IMPORTER or EXPORTER key in this parameter, the NOCV bit is propagated to the imported key token.

Note: Propagation of the NOCV bit is not performed if the service is processed on the PCI Cryptographic Coprocessor.

The secure key import service does not adjust key parity.

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Systems with the Cryptographic Coprocessor Feature

To generate double-length MAC and MACVER keys in the importable form, the ANSI system keys must be installed in the CKDS.

This service will mark DATA, IMPORTER and EXPORTER key tokens with the system encryption algorithm.
- This service marks the imported DATA key token according to the system’s default encryption algorithm, unless token copying overrides this.
- KEKs are marked SYS-ENC unless token copying overrides this.
Secure Key Import (CSNBSKI)

- To override the default mark, supply a valid internal token of the same key type in the key_identifier field. The service will copy the marks of the supplied token to the imported token.

Systems with the PCI X Cryptographic Coprocessor, Crypto Express2 Coprocessor, or Crypto Express3 Coprocessor

If key_form is IM and the importer_key_identifier is NOCV KEK, the NOCV IMPORTER access control point must be enabled.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 64. Secure key import required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>Marking of data encryption algorithm bits and token copying are performed only if the service is processed on the Cryptographic Coprocessor Feature.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>PCI Cryptographic Coprocessor</td>
<td>ICSF routes the request to a PCI Cryptographic Coprocessor if:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The control vector of a supplied internal token cannot be processed on the Cryptographic Coprocessor Feature, or if the key type is not valid for the Cryptographic Coprocessor Feature.</td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>Key_type DATAXLAT is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td>Key_type DATAXLAT is not supported.</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Key_type DATAXLAT is not supported.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Key_type DATAXLAT is not supported.</td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Key_type DATAXLAT is not supported.</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td>Key_type DATAXLAT is not supported.</td>
</tr>
</tbody>
</table>

Symmetric Key Export (CSNDSYX and CSNFSYX)

Use the symmetric key export callable service to transfer an application-supplied AES or DES key (a DATA key) from encryption under a master key to encryption under an application-supplied RSA public key. The application-supplied DATA key must be an ICSF AES or DES internal key token or the label of such a token in the CKDS. The symmetric key import callable service can import the RSA public key encrypted DATA key at the receiving node.

This callable service supports invocation in AMODE(64). The callable service name for AMODE(64) is CSNFSYX.
Symmetric Key Export (CSNDSYX and CSNFSYX)

Format

CALL CSNDSYX(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    DATA_key_identifier_length,
    DATA_key_identifier,
    RSA_public_key_identifier_length,
    RSA_public_key_identifier,
    RSA_enciphered_key_length,
    RSA_enciphered_key)

Parameters

return_code
Direction: Output  Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

reason_code
Direction: Output  Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

exit_data_length
Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

exit_data
Direction: Input/Output  Type: String

The data that is passed to the installation exit.

rule_array_count
Direction: Input  Type: Integer

The number of keywords you are supplying in the rule_array parameter. Value may be 1 or 2.

rule_array
Direction: Input  Type: String
Symmetric Key Export (CSNDSYX and CSNFSYX)

Keywords that provide control information to the callable service. Table 65 lists the keywords. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 65. Keywords for Symmetric Key Export Control Information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm (one keyword, optional)</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>The key being exported is an AES key.</td>
</tr>
<tr>
<td>DES</td>
<td>The key being exported is a DES key. This is the default.</td>
</tr>
<tr>
<td>Recovery Method (required)</td>
<td></td>
</tr>
<tr>
<td>PKCSOAEP</td>
<td>Specifies to use the method found in RSA DSI PKCS #1V2 OAEP.</td>
</tr>
<tr>
<td>PKCS–1.2</td>
<td>Specifies to use the method found in RSA DSI PKCS #1 block type 02 to recover the symmetric key.</td>
</tr>
<tr>
<td>ZERO-PAD</td>
<td>The clear key is right-justified in the field provided, and the field is padded to the left with zeros up to the size of the RSA encryption block (which is the modulus length).</td>
</tr>
</tbody>
</table>

**DATA_key_identifier_length**

Direction: Input  Type: Integer

The length of the *DATA_key_identifier* parameter. The minimum size is 64 bytes. The maximum size is 128 bytes.

**DATA_key_identifier**

Direction: Input/Output  Type: String

The label or internal token of a secure AES or DES DATA key to encrypt under the supplied RSA public key. The key in the key identifier must match the algorithm in the *rule_array*. DES is the default algorithm.

**RSA_public_key_identifier_length**

Direction: Input  Type: Integer

The length of the *RSA_public_key_identifier* parameter. The maximum size is 3500 bytes.

**RSA_public_key_identifier**

Direction: Input  Type: String

A PKA public key token or label of the key to protect the exported symmetric key.

**RSA_enciphered_key_length**

Direction: Input/Output  Type: Integer

The length of the *RSA_enciphered_key* parameter. This is updated with the actual length of the *RSA_enciphered_key* generated. The maximum size you
Symmetric Key Export (CSNDSYX and CSNFSYX)

can specify in this parameter is 512 bytes, although the actual key length may be further restricted by your hardware configuration (as shown in Table 66).

**RSA_enciphered_key**

Direction: Output  
Type: String

This field contains the RSA_enciphered key, protected by the public key specified in the *RSA_public_key_identifier* field.

**Restrictions**

If you are running with the Cryptographic Coprocessor Feature, the enhanced system keys must be present in the CKDS.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

The hardware configuration sets the limit on the modulus size of keys for key management; thus, this service will fail if the RSA key modulus bit length exceeds this limit.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 66. Symmetric key export required hardware*

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>RSA keys with moduli greater than 1024-bit length are not supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Secure AES keys are not supported.</td>
</tr>
<tr>
<td>IBM zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PCI Cryptographic Coprocessor</td>
<td>ICSF routes this service to a PCI Cryptographic Coprocessor if one is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>available on your server. This service will not be routed to a PCI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cryptographic Coprocessor if the modulus bit length of the RSA public key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is less than 512 bits. Use of keyword PKCSOAEP requires the PCI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cryptographic Coprocessor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA keys with moduli greater than 2048-bit length are not supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Secure AES keys are not supported.</td>
</tr>
<tr>
<td>IBM zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>RSA keys with moduli greater than 2048-bit length are not supported.</td>
</tr>
<tr>
<td>IBM zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td>Secure AES keys are not supported.</td>
</tr>
</tbody>
</table>
Symmetric Key Export (CSNDSYX and CSNFSYX)

Table 66. Symmetric key export required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC). Secure AES keys require the Nov. 2008 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC). Secure AES keys require the Nov. 2008 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

Symmetric Key Generate (CSNDSYG)

Use the symmetric key generate callable service to generate an AES or DES DATA key and return the key in two forms: enciphered under the master key and encrypted under an RSA public key.

You can import the RSA public key encrypted form by using the symmetric key import service at the receiving node.

Also use the symmetric key generate callable service to generate any DES importer or exporter key-encrypting key encrypted under a RSA public key according to the PKA92 formatting structure. See "PKA92 Key Format and Encryption Process" on page 691 for more details about PKA92 formatting.

Format

```call csndsysg(
return_code,
reason_code,
exit_data_length,
exit_data,
rule_array_count,
rule_array,
key_encrypting_key_identifier,
RSA_public_key_identifier_length,
RSA_public_key_identifier,
local_enciphered_key_token_length,
local_enciphered_key_token,
RSA_enciphered_key_length,
RSA_enciphered_key)
```

Parameters

**return_code**

Direction: Output  Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.
Symmetric Key Generate (CSNDSYG)

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. [Appendix A, “ICSF and TSS Return and Reason Codes”](#) lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the `exit_data` parameter.

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**rule_array_count**

Direction: Input  
Type: Integer

The number of keywords you supplied in the `rule_array` parameter. The value must be 1, 2, 3 or 4.

**rule_array**

Direction: Input  
Type: String

Keywords that provide control information to the callable service. [Table 67](#) lists the keywords. The recovery method is the method to use to recover the symmetric key. Each keyword is left-justified in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage.

**Table 67. Keywords for Symmetric Key Generate Control Information**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithm (one keyword, optional)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>The key being generated is a secure AES key.</td>
<td>AES</td>
</tr>
<tr>
<td>DES</td>
<td>The key being generated is a DES key. This is the default.</td>
<td>DES</td>
</tr>
<tr>
<td><strong>Key formatting method (one keyword required)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PKA92</td>
<td>Specifies the key-encrypting key is to be encrypted under a PKA96 RSA public key according to the PKA92 formatting structure.</td>
<td>DES</td>
</tr>
<tr>
<td>PKCSOAEP</td>
<td>Specifies using the method found in RSA DSI PKCS #1V2 OAEP.</td>
<td>AES or DES</td>
</tr>
<tr>
<td>PKCS-1.2</td>
<td>Specifies the method found in RSA DSI PKCS #1 block type 02.</td>
<td>AES or DES</td>
</tr>
</tbody>
</table>
**Table 67. Keywords for Symmetric Key Generate Control Information (continued)**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZERO-PAD</td>
<td>The clear key is right-justified in the field provided, and the field is padded to the left with zeros up to the size of the RSA encryption block (which is the modulus length).</td>
<td>AES or DES</td>
</tr>
</tbody>
</table>

**Key Length (optional - for use with PKA92)**

| SINGLE-R  | For key-encrypting keys, this specifies that the left half and right half of the generated key will have identical values. This makes the key operate identically to a single-length key with the same value. Without this keyword, the left and right halves of the key-encrypting key will each be generated randomly and independently. | DES        |

**Key Length (optional - for use with PKCSOAEP, PKCS-1.2, or ZERO-PAD)**

| SINGLE, KEYLN8 | Specifies that the generated key should be 8 bytes in length. | DES        |
| DOUBLE        | Specifies that the generated key should be 16 bytes in length. | DES        |
| KEYLN16       | Specifies that the generated key should be 16 bytes in length. | AES or DES |
| KEYLN24       | Specifies that the generated key should be 24 bytes in length. | AES or DES |
| KEYLN32       | Specifies that the generated key should be 32 bytes in length. | AES        |

**Encipherment method for the local enciphered copy of the key (optional - for use with PKCSOAEP, PKCS-1.2, or ZERO-PAD)**

| OP          | Enciphers the key with the master key. The DES master key is used with DES keys and the AES master key is used with AES keys. | AES or DES |
| EX          | Enciphers the key with the EXPORTER key that is provided through the `key_encrypting_key_identifier` parameter. | DES        |
| IM          | Enciphers the key with the IMPORTER key-encrypting key specified with the `key_encrypting_key_identifier` parameter. | DES        |

**key_encrypting_key_identifier**

Direction: Input/Output  
Type: String

The label or internal token of a key-encrypting key. If the `rule_array` specifies IM, this DES key must be an IMPORTER. If the `rule_array` specifies EX, this DES key must be an EXPORTER. Otherwise, the parameter is ignored.
Symmetric Key Generate (CSNDSYG)

**RSA_public_key_identifier_length**
Direction: Input  
Type: Integer

The length of the `RSA_public_key_identifier` parameter. If the `RSA_public_key_identifier` parameter is a label, this parameter specifies the length of the label. The maximum size is 3500 bytes.

**RSA_public_key_identifier**
Direction: Input  
Type: String

The token, or label, of the RSA public key to be used for protecting the generated symmetric key.

**local_enciphered_key_token_length (was DES_enciphered_key_token_length)**
Direction: Input/Output  
Type: Integer

The length in bytes of the `local_enciphered_key_token`. This field is updated with the actual length of the token that is generated. The minimum length is 64-bytes and the maximum length is 128 bytes.

**local_enciphered_key_token (was DES_enciphered_key_token)**
Direction: Input/Output  
Type: String

This parameter contains the generated DATA key in the form of an internal or external token, depending on `rule_array` specification. If you specify PKA92, on input specify an internal (operational) key token of an Importer or Exporter Key.

**RSA_enciphered_key_length**
Direction: Input/Output  
Type: Integer

The length of the `RSA_enciphered_key` parameter. This service updates this with the actual length of the `RSA_enciphered_key` it generates. The maximum size is 512 bytes.

**RSA_enciphered_key**
Direction: Input/Output  
Type: String

This field contains the RSA enciphered key, which the public key specified in the `RSA_public_key_identifier` field protects.

**Restrictions**

If the service is executed on the Cryptographic Coprocessor Feature, and you specify IM in the `rule_array`, you must enable Special Secure Mode.

Use of PKA92 or PKCSOAEP requires a PCICC, PCIXCC, CEX2C, or CEX3C.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.
Symmetric Key Generate (CSNDSYG)

If the service is executed on the Cryptographic Coprocessor Feature, the generated internal DATA key token is marked according to the system default algorithm.

The hardware configuration sets the limit on the modulus size of keys for key management; thus, this service will fail if the RSA key modulus bit length exceeds this limit.

Specification of PKA92 with an input NOCV key-encrypting key token is not supported.

Use the PKA92 key-formating method to generate a key-encrypting key. The service enciphers one key copy using the key encipherment technique employed in the IBM Transaction Security System (TSS) 4753, 4755, and AS/400 cryptographic product PKA92 implementations (see "PKA92 Key Format and Encryption Process" on page 691). The control vector for the RSA-enciphered copy of the key is taken from an internal (operational) DES key token that must be present on input in the RSA_enciphered_key variable. Only key-encrypting keys that conform to the rules for an OPEX case under the key generate service are permitted. The control vector for the local key is taken from a DES key token that must be present on input in the DES_enciphered_key_token variable. The control vector for one key copy must be from the EXPORTER class while the control vector for the other key copy must be from the IMPORTER class.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 68. Symmetric key generate required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>ICSF routes this service to a PCI Cryptographic Coprocessor if one is available on your server. This service will not be routed to a PCI Cryptographic Coprocessor if the modulus bit length of the RSA public key is less than 512 bits. RSA keys with moduli greater than 1024-bit length are not supported. Secure AES keys are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>PCI Cryptographic Coprocessor</td>
<td>Use of keyword PKA92 or PKCSOAEP requires the PCI Cryptographic Coprocessor. RSA keys with moduli greater than 2048-bit length are not supported. Secure AES keys are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>The generated internal DATA key will not have any system encryption algorithm markings. RSA keys with moduli greater than 2048-bit length are not supported. Secure AES keys are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>
Symmetric Key Generate (CSNDSYG)

Table 68. Symmetric key generate required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM Systems z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>The generated internal DATA key will not have any system encryption algorithm markings.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Secure AES keys require the Nov. 2008 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>The generated internal DATA key will not have any system encryption algorithm markings.</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Secure AES keys require the Nov. 2008 or later licensed internal code (LIC).</td>
</tr>
</tbody>
</table>

Symmetric Key Import (CSNDSYI and CSNFSYI)

Use the symmetric key import callable service to import a symmetric (DES or AES) DATA key enciphered under an RSA public key. It returns the key in operational form, enciphered under the master key.

This service also supports import of a PKA92-formatted DES key-encrypting key under a PKA96 RSA public key.

This callable service supports invocation in AMODE(64). The callable service name for AMODE(64) is CSNFSYI.

Format

```call csndsyi(  return_code,  reason_code,  exit_data_length,  exit_data,  rule_array_count,  rule_array,  RSA_enciphered_key_length,  RSA_enciphered_key,  RSA_private_key_identifier_length,  RSA_private_key_identifier,  target_key_identifier_length,  target_key_identifier)```
Symmetric Key Import (CSNDSYI and CSNFSYI)

Parameters

**return_code**
- Direction: Output
- Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**
- Direction: Output
- Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**
- Direction: Input/Output
- Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**
- Direction: Input/Output
- Type: String

The data that is passed to the installation exit.

**rule_array_count**
- Direction: Input
- Type: Integer

The number of keywords you supplied in the rule_array parameter. The value may be 1 or 2.

**rule_array**
- Direction: Input
- Type: String

The keyword that provides control information to the callable service. Table 69 provides a list. The recovery method is the method to use to recover the symmetric key. The keyword is left-justified in an 8-byte field and padded on the right with blanks.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm (one keyword, optional)</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>The key being imported is an AES key.</td>
</tr>
<tr>
<td>DES</td>
<td>The key being imported is a DES key. This is the default.</td>
</tr>
<tr>
<td>Recovery Method (required)</td>
<td></td>
</tr>
</tbody>
</table>
Symmetric Key Import (CSNDSYI and CSNFSYI)

Table 69. Keywords for Symmetric Key Import Control Information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKA92</td>
<td>Supported by the DES algorithm. Specifies the key-encrypting key is encrypted under a PKA96 RSA public key according to the PKA92 formatting structure.</td>
</tr>
<tr>
<td>PKCSOAEP</td>
<td>Specifies to use the method found in RSA DSI PKCS #1V2 OAEP. Supported by the DES and AES algorithms.</td>
</tr>
<tr>
<td>PKCS-1.2</td>
<td>Specifies to use the method found in RSA DSI PKCS #1 block type 02. Supported by the DES and AES algorithms.</td>
</tr>
<tr>
<td>ZERO-PAD</td>
<td>The clear key is right-justified in the field provided, and the field is padded to the left with zeros up to the size of the RSA encryption block (which is the modulus length). Supported by the DES and AES algorithms.</td>
</tr>
</tbody>
</table>

**RSA_enciphered_key_length**

Direction: Input  Type: integer

The length of the `RSA_enciphered_key` parameter. The maximum size is 512 bytes.

**RSA_enciphered_key**

Direction: Input  Type: String

The key to import, protected under an RSA public key. The encrypted key is in the low-order bits (right-justified) of a string whose length is the minimum number of bytes that can contain the encrypted key. This string is left-justified within the `RSA_enciphered_key` parameter.

**RSA_private_key_identifier_length**

Direction: Input  Type: Integer

The length of the `RSA_private_key_identifier` parameter. When the `RSA_private_key_identifier` parameter is a key label, this field specifies the length of the label. The maximum size is 3500 bytes.

**RSA_private_key_identifier**

Direction: Input  Type: String

An internal RSA private key token or label whose corresponding public key protects the symmetric key.

**target_key_identifier_length**

Direction: Input/Output  Type: Integer

The length of the `target_key_identifier` parameter. This field is updated with the actual length of the `target_key_identifier` that is generated. The size must be 64 bytes.
Symmetric Key Import (CSNDSYI and CSNFSYI)

**target_key_identifier**

Direction: Input/Output  
Type: String

This field contains the internal token of the imported symmetric key. Except for PKA92 processing, this service produces a DATA key token with a key of the same length as that contained in the imported token.

**Restrictions**

The exponent of the RSA public key must be odd.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

If the service is executed on the Cryptographic Coprocessor Feature, the generated internal DATA key token is marked according to the default system encryption algorithm unless token copying overrides this. Token copying is accomplished by supplying a valid DATA token with the desired algorithm marks in the target_key_identifier field.

The hardware configuration sets the limit on the modulus size of keys for key management; thus, this service will fail if the RSA key modulus bit length exceeds this limit. The service will fail with return code 12 and reason code 11020.

Specification of PKA92 with an input NOCV key-encrypting key token is not supported.

During initialization of a PCICC, PCIXCC, CEX2C, or CEX3C, an Environment Identification, or EID, of zero will be set in the coprocessor. This will be interpreted by the PKA Symmetric Key Import service to mean that environment identification checking is to be bypassed. Thus it is possible on a OS/390 system for a key-encrypting key RSA-enciphered at a node (EID) to be imported at the same node.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
### Symmetric Key Import (CSNDSYI and CSNFSYI)

**Table 70. Symmetric key import required hardware**

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
</table>
| IBM @server zSeries 800 | Cryptographic Coprocessor Feature | Request routed to the CCF when -  
|                         |                                 | • The RSA_private_key_identifier is a modulus-exponent form private key with a private section ID of X'02'  
|                         |                                 | • The key modulus bit length is less than 512  
|                         |                                 | RSA keys with moduli greater than 1024-bit length are not supported.  
|                         |                                 | Secure AES keys are not supported.  |
| IBM @server zSeries 900 | PCI Cryptographic Coprocessor   | Request routed to PCICC when -  
|                         |                                 | • The RSA_private_key_identifier is a modulus-exponent form private key with a private section ID of X'06'  
|                         |                                 | • The RSA_private_key_identifier is a CRT form private key with a private section ID of X'08'  
|                         |                                 | • The RSA_private_key_identifier is a retained key  
|                         |                                 | • PKA92 recovery method specified  
|                         |                                 | • PKCSOAEP recovery method specified  
|                         |                                 | RSA keys with moduli greater than 2048-bit length are not supported.  
|                         |                                 | Secure AES keys are not supported.  |
| IBM @server zSeries 990 | PCI X Cryptographic Coprocessor | The imported internal DATA key will not have any system encryption markings. Old RSA private keys encrypted under the CCF KMMK is not usable if the KMMK is not the same as the PCIICC/CEX2C ASYM-MK.  
| IBM @server zSeries 890 | Crypto Express2 Coprocessor     | RSA keys with moduli greater than 2048-bit length are not supported.  
|                         |                                 | Secure AES keys are not supported.  |
| IBM System z9 EC        | Crypto Express2 Coprocessor     | The imported internal DATA key will not have any system encryption markings. Old RSA private keys encrypted under the CCF KMMK is not usable if the KMMK is not the same as the CEX2C ASYM-MK.  
| IBM System z9 BC        |                                 | RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).  
|                         |                                 | Secure AES keys are not supported.  |
Symmetric Key Import (CSNDSYI and CSNFSYI)

Table 70. Symmetric key import required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>The imported internal DATA key will not have any system encryption markings. Old RSA private keys encrypted under the CCF KMMK is not usable if the KMMK is not the same as the CEX2C or CEX3C ASYM-MK. RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC). Secure AES keys require the Nov. 2008 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

Transform CDMF Key (CSNBTKCK)

This callable service is only supported on an IBM @server zSeries 800 and IBM @server zSeries 900.

Use the transform CDMF key callable service to change a CDMF DATA key in an internal or external token to a transformed shortened DES key. You can also use the key label of a CKDS record as input.

The Cryptographic Coprocessor Feature on IBM @server zSeries 900, S/390 Enterprise Servers and S/390 Multiprise is configured as either CDMF or DES-CDMF. This callable service ignores the input internal DATA token markings, and it marks the output internal token for use in the DES.

If the input DATA key is in an external token, the operational KEK must be marked as DES or SYS-ENC. The service fails for an external DATA key encrypted under a KEK that is marked as CDMF.

Format

```
CALL CSNBTKCK(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    source_key_identifier,
    kek_key_identifier,
    target_key_identifier )
```

Parameters

```
return_code
```

Direction: Output Type: Integer

The return code specifies the general result of the callable service. [Appendix A, "ICSF and TSS Return and Reason Codes"] lists the return codes.
Transform CDMF Key (CSNBTCK)

**reason_code**

Direction: Output  Type: Integer

The reason code specifies the result of the callable service that is returned to
the application program. Each return code has different reason codes that
indicate specific processing problems. Appendix A, "ICSF and TSS Return and
Reason Codes" lists the reason codes.

**exit_data_length**

Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be
from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the
exit_data parameter.

**exit_data**

Direction: Input/Output  Type: String

The data that is passed to the installation exit.

**rule_array_count**

Direction: Input  Type: Integer

The number of keywords you are supplying in the rule_array parameter. This
number must be 0.

**rule_array**

Direction: Input  Type: String

Currently no rule_array keywords are defined for this service, but you still must
specify this parameter.

**source_key_identifier**

Direction: Input/Output  Type: String

A 64-byte string of the internal token, external token or key label that contains
the DATA key to transform. Token markings on this key token are ignored.

**kek_key_identifier**

Direction: Input/Output  Type: String

A 64-byte string of the internal token or a key label of a key encrypting key
under which the source_key_identifier is encrypted.

**target_key_identifier**

Direction: Output  Type: String

**Note:** If you supply a label for this parameter, the label must be unique in the
CKDS.
Transform CDMF Key (CSNBTCK)

A 64-byte string where the internal token or external token of the transformed shortened DES key is returned. The internal token is marked as DES.

Restrictions

This service is available on S/390 Enterprise Servers and S/390 Multiprise with Cryptographic Coprocessor Features. These systems may be configured as either CDMF or DES-CDMF.

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

This service transforms a CDMF DATA key to a transformed shortened DES DATA key to allow interoperability to a DES-only capable system. The algorithm is described in Transform CDMF Key Algorithm.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 71. Transform CDMF key required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Trusted Block Create (CSNDTBC)

This callable service is used to create a trusted block in a two step process. The block will be in external form, encrypted under an IMP-PKA transport key. This means that the MAC key contained within the trusted block will be encrypted under the IMP-PKA key.
Trusted Block Create (CSNDTBC)

Format

```
CALL CSNDTBC(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    input_block_length
    input_block_identifier
    trusted_block_length,
    trusted_block_identifier
)
```

Parameters

**return_code**

Direction: Output Type: Integer

The return code specifies the general result of the callable service. Appendix A, ICSF and TSS Return and Reason Codes lists the return codes.

**reason_code**

Direction: Output Type: Integer

The reason code specifies the specific results of processing. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit_data_length**

Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the `exit_data` parameter.

**exit_data**

Direction: Input/Output Type: String

The data that is passed to the installation exit.

**rule_array_count**

Direction: Input Type: Integer

The number of keywords you are supplying in the `rule_array` parameter. This number must be 1.

**rule_array**

Direction: Input Type: String

Specifies a string variable containing an array of keywords. The keywords are 8 bytes long and must be left-justified and right padded with blanks.
This table lists the rule_array keywords for this callable service.

Table 72. Rule_array keywords for Trusted Block Create (CSNDTBC)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operational Keywords - One Required</strong></td>
<td></td>
</tr>
<tr>
<td>INACTIVE</td>
<td>Create the trusted block, but in inactive form. The MAC key is randomly generated, encrypted with the transport key, and inserted into the block. The ACTIVE flag is set to False (0), and the MAC is calculated over the block and inserted in the appropriate field. The resulting block is fully formed and protected, but it is not usable in any other CCA services. Use of the INACTIVE keyword is authorized by the 0x030F access control point.</td>
</tr>
<tr>
<td>ACTIVATE</td>
<td>This makes the trusted block usable in CCA services. Use of the ACTIVATE keyword is authorized by the 0x0310 access control point.</td>
</tr>
</tbody>
</table>

**input_block_length**

Direction: Input/Output  
Type: String

Specifies the number of bytes of data in the input_block_identifier parameter. The maximum length is 3500 bytes.

**input_block_identifier**

Direction: Input  
Type: String

Specifies a trusted block label or complete trusted block token, which will be updated by the service and returned in trusted_block_identifier. The length is indicated by input_block_length. Its content depends on the rule array keywords supplied to the service.

When rule_array is INACTIVE the block is complete but typically does not have MAC protection. If MAC protection is present due to recycling an existing trusted block, then the MAC key and MAC value will be overlaid by the new MAC key and MAC value. The input_block_identifier includes all fields of the trusted block token, but the MAC key and MAC will be filled in by the service. The Active flag will be set to False (0) in the block returned in trusted_block_identifier.

When the rule_array is ACTIVATE the block is complete, including the MAC protection which is validated during execution of the service. The Active flag must be False (0) on input. On output, the block will be returned in trusted_block_identifier provided the identifier is a token, with the Active flag changed to True (1), and the MAC value recalculated using the same MAC key. If the trusted_block_identifier is a label, the block will be written to the PKDS.

**transport_key_identifier**

Direction: Input  
Type: String

Specifies a key label or key token for an IMP-PKA key that is used to protect the trusted block.

**trusted_block_length**

Direction: Input/Output  
Type: Integer
Trusted Block Create (CSNDTBC)

Specifies the number of bytes of data in trusted_block_identifier parameter. The maximum length is 3500 bytes.

**trusted_block_identifier**

| Direction: Output | Type: String |

Specifies a trusted block label or trusted block token for the trusted block constructed by the service. On input, the trusted_block_length contains the size of this buffer. On output, the trusted_block_length is updated with the actual byte length of the trusted block written to the buffer if the trusted_block_identifier is a token. The trusted block consists of the data supplied in input_block_identifier, but with the MAC protection and Active flag updated according to the rule array keyword that is provided. See Table 72 on page 210 for details on the actions. If the trusted_block_identifier is a label identifying a key record in key storage, the returned trusted block token will be written to the PKDS.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 73. Trusted Block Create required hardware*

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 990</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 800</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Cryptographic Express2 Coprocessor</td>
<td>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>
User Derived Key (CSFUDK)

Use the user derived key callable service to generate a single-length or double-length MAC key or to update an existing user derived key. A single-length MAC key can be used to compute a MAC following the ANSI X9.9, ANSI X9.19, or the Europay, MasterCard and VISA (EMV) Specification MAC processing rules. A double-length MAC key can be used to compute a MAC following either the ANSI X9.19 optional double MAC processing rule or the EMV Specification MAC processing rule.

This service updates an existing user derived key by XORing it with data you supply in the data_array parameter. This is called SESSION MAC key generation by VISA.

This service adjusts the user derived key or SESSION MAC key to odd parity. The parity of the supplied derivation key is not tested.

Format

CALL CSFUDK(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  key_type,
  rule_array_count,
  rule_array,
  derivation_key_identifier,
  source_key_identifier,
  data_array,
  generated_key_identifier)

Parameters

return_code
Direction: Output
Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code
Direction: Output
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

exit_data_length
Direction: Input/Output
Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.
User Derived Key (CSFUDK)

**exit_data**
Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**key_type**
Direction: Input  
Type: String

The 8-byte keyword of 'MAC ' or 'MACD ' that specifies the key type to be generated. The keyword must be left-justified and padded on the right with blanks. MAC specifies an 8-byte, single-length MAC key which is used in the ANSI X9.9-1 or the ANSI X9.19 basic MAC processing rules. MACD specifies a 16-byte, double-length internal MAC key that uses the single-length control vector for both the left and right half of the key (MAC || MAC). The double-length MAC key is used in the ANSI X9.19 optional double-key MAC processing rules. The keyword 'TOKEN ' is also accepted. If you specify TOKEN with a rule_array of VISA or NOFORMAT, the key type is determined by the valid internal token of the single-length or double-length MAC key in the generated_key_identifier parameter. If you specify TOKEN with a rule_array of SESS-MAC, the key type is determined by the valid internal token of the single-length or double-length MAC key in the source_key_identifier.

**rule_array_count**
Direction: Input  
Type: Integer

The number of keywords specified in the rule_array parameter. The value must be 1.

**rule_array**
Direction: Input  
Type: Character string

The process rule for the user derived key in an 8-byte field. The keywords must be in 8 bytes of contiguous storage, left-justified and padded on the right with blanks.

The keywords are shown in Table 74.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User Derived Key Process Rules (required)</strong></td>
<td></td>
</tr>
<tr>
<td>NOFORMAT</td>
<td>For generating a user derived key with no formatting done on the array before encryption under the derivation_key_identifier.</td>
</tr>
<tr>
<td>SESS-MAC</td>
<td>To update an existing user derived key supplied in the source_key_identifier parameter with data provided in the data_array parameter.</td>
</tr>
<tr>
<td>VISA</td>
<td>For generating a user derived key using the VISA algorithm to format the data array input before encryption under the derivation_key_identifier. For guidance information refer to the VISA Integrated Circuit Card Specification, V1.3 Aug 31, 1996.</td>
</tr>
</tbody>
</table>
User Derived Key (CSFUDK)

**derivation_key_identifier**

Direction: Input/Output  
Type: String

For a `rule_array` value of VISA or NOFORMAT, this is a 64-byte key label or internal key token of the derivation key used to generate the user derived key. The key must be an EXPORTER key type. For any other keyword, this field must be a null token.

**source_key_identifier**

Direction: Input/Output  
Type: String

For a `rule_array` value of SESS-MAC, this is a 64-byte internal token of a single-length or double-length MAC key. For any other keyword, this field must be a null token.

**data_array**

Direction: Input  
Type: String

Two 16-byte data elements required by the corresponding `rule_array` and `key_type` parameters. The data array consists of two 16-byte hexadecimal character fields whose specification depends on the process rule and key type. VISA requires only one 16-byte hexadecimal character input. Both NOFORMAT and SESS-MAC require one 16-byte input for a key type of MAC and two 16-byte inputs for a key type of MACD. If only one 16-byte field is required, then the rest of the data array is ignored by the callable service.

**generated_key_identifier**

Direction: Input/Output  
Type: String

The 64-byte internal token of the generated single-length or double-length MAC key. This is an input field only if TOKEN is specified for `key_type`.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

This service requires that the ANSI system keys be stored in the CKDS.

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### User Derived Key (CSFUDK)

#### Table 75. User derived key required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 990</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM System z9 EC and z9 BC</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM System z10 EC and z10 BC</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
</tbody>
</table>
User Derived Key (CSFUDK)
Chapter 6. Protecting Data

Use ICSF to protect sensitive data stored on your system, sent between systems, or stored off your system on magnetic tape. To protect data, encipher it under a key. When you want to read the data, decipher it from ciphertext to plaintext form.

ICSF provides encipher and decipher callable services to perform these functions. If you use a key to encipher data, you must use the same key to decipher the data. To use clear keys directly, ICSF provides symmetric key decipher, symmetric key encipher, encode and decode callable services. These services encipher and decipher with clear keys. You can use clear keys indirectly by first using the clear key import callable service, and then using the encipher and decipher callable services.

This topic describes these services:
- “Ciphertext Translate (CSNBCTT and CSNBCTT1)” on page 219
- “Decipher (CSNBDEC and CSNBDEC1)” on page 223
- “Decode (CSNBDCO)” on page 229
- “Encipher (CSNBNENC and CSNBNENC1)” on page 231
- “Encode (CSNBECO)” on page 239
- “Symmetric Algorithm Decipher (CSNBSAD, CSNBSAD1, CSNESAD and CSNESAD1)” on page 241
- “Symmetric Algorithm Encipher (CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1)” on page 247
- “Symmetric Key Decipher (CSNBSYD, CSNBSYD1, CSNESYD and CSNESYD1)” on page 253
- “Symmetric Key Encipher (CSNBSYE, CSNBSYE1, CSNESYE and CSNESYE1)” on page 260

Modes of Operation

To encipher or decipher data or keys, ICSF uses either the U.S. National Institute of Standards and Technology (NIST) Data Encryption Standard (DES) algorithm or the Commercial Data Masking Facility (CDMF). The DES algorithm is documented in Federal Information Processing Standard #46. CDMF provides DES cryptography using an effectively shortened DATA key. See “System Encryption Algorithm” on page 45 for more information.

To encipher or decipher data, ICSF also uses the U.S. National Institute of Standards and Technology (NIST) Advanced Encryption Standard (AES) algorithm. The AES algorithm is documented in Federal Information Processing Standard 197.

ICSF enciphers and deciphers using these modes of operation:
- Cipher Feedback (CFB)
- Cipher block chaining (CBC)
- Electronic code book (ECB)

Cipher Feedback (CFB) Mode

The CFB mode uses an initial chaining vector (ICV) in its processing. The CFB mode performs cipher feedback encryption. The ICV is first encrypted, then exclusive ORed with the first block of ciphertext, and thereafter, the block of
exclusive ORed data is encrypted then exclusive ORed with the next block of ciphertext, and so on. The ciphertext can be of any length. The plaintext will have the same length as the ciphertext.

**Cipher Block Chaining (CBC) Mode**

The CBC mode uses an initial chaining vector (ICV) in its processing. The CBC mode only processes blocks of data in exact multiples of eight. The ICV is exclusive ORed with the first 8 bytes of plaintext prior to the encryption step; the 8-byte block of ciphertext just produced is exclusive ORed with the next 8-byte block of plaintext, and so on. You must use the same ICV to decipher the data. This disguises any pattern that may exist in the plaintext. ICSF uses the CBC encipherment mode for encrypting and decrypting data using the encipher and decipher callable services.

**Electronic Code Book (ECB) Mode**

In the ECB mode, each 64-bit block of plaintext is separately enciphered and each block of the ciphertext is separately deciphered. In other words, the encipherment or decipherment of a block is totally independent of other blocks. ICSF uses the ECB encipherment mode for enciphering and deciphering data with clear keys using the encode and decode callable services.

ICSF does not support ECB encipherment mode on CDMF-only systems.

**Triple DES Encryption**

Triple-DES encryption uses a triple-length DATA key comprised of three 8-byte DES keys to encipher 8 bytes of data using this method:

- Encipher the data using the first key
- Decipher the result using the second key
- Encipher the second result using the third key

The procedure is reversed to decipher data that has been triple-DES enciphered:

- Decipher the data using the third key
- Encipher the result using the second key
- Decipher the second result using the first key

ICSF uses the triple-DES encryption in the CBC encipherment mode.

A variation of the triple DES algorithm supports the use of a double-length DATA key comprised of two 8-byte DATA keys. In this method, the first 8-byte key is reused in the last encipherment step.

Triple-DES encryption is available only on the S/390 G4 Enterprise Server (with LIC driver 98), or higher. Due to export regulations, triple-DES encryption may not be available on your processor.

**Processing Rules**

ICSF handles this chaining for each 8-byte block of data, from the first block until the last complete 8-byte block of data in each encipher call. There are different types of processing rules you can choose for cipher block chaining. You choose the type of processing rule that the callable service should use for CBC mode:

- **Cipher block chaining (CBC).** In exact multiples of 8 bytes.
• **Cryptographic Unit Support Program (CUSP).** Not necessarily in exact multiples of 8 bytes. The ciphertext is the same length of the plaintext.

• **Information Protection System (IPS).** Not necessarily in exact multiples of 8 bytes. The ciphertext is the same length of the plaintext.

• **ANSI X9.23.** Not necessarily in exact multiples of 8 bytes. This processing rule pads the plaintext so that the ciphertext produced is in exact multiples of 8 bytes.

• **IBM 4700.** Not necessarily in exact multiples of 8 bytes. This processing rule pads the plaintext so that the ciphertext produced is in exact multiples of 8 bytes.

**Cipher Processing Rules** describes the cipher processing rules in detail.

The resulting chaining value, after an encipher call, is known as an *output chaining vector (OCV)*. When there are multiple cipher requests, the application can pass the output chaining vector from the previous encipher call as the ICV in the next encipher call. This produces chaining between successive calls, which is known as *record chaining*. ICSF provides the ICV selection keyword **CONTINUE** in the *rule_array* parameter that an application can use to select record chaining with the CFB, CBC, IPS, and CUSP processing rules.

A chaining vector allows you to simulate CUSP or IPS record chaining by calculating the correct OCV. To do either the CUSP or IPS method of record chaining in the encipher and decipher callable services, the OCV from one service invocation is passed as the initialization vector to the next invocation. An OCV is produced for all processing rules. The OCV is the leftmost 8 bytes of the chaining_vector parameter.

### Ciphertext Translate (CSNBCTT and CSNBCTT1)

This callable service is only supported on the IBM @server zSeries 800 and the IBM @server zSeries 900.

ICSF provides a ciphertext translate callable service on DES-capable systems. The callable service deciphers encrypted data (ciphertext) under one data translation key and reenciphers it under another data translation key without having the data appear in the clear outside the Cryptographic Coprocessor Feature. ICSF uses the data translation key as either the input or the output data transport key. Such a function is useful in a multiple node network, where sensitive data is passed through multiple nodes prior to it reaching its final destination.

"Using the Ciphertext Translate Callable Service" on page 60 provides some tips on using the callable service.

Use the ciphertext translate callable service to decipher text under an “input” key and then to encipher the text under an “output” key. The callable service uses the cipher block chaining (CBC) mode of the DES. This service is available only on a DES-capable system.

### Choosing Between CSNBCTT and CSNBCTT1

CSNBCTT and CSNBCTT1 provide identical functions. When choosing the service to use, consider this:

• **CSNBCTT** requires the input text and output text to reside in the caller’s primary address space. Also, a program using CSNBCTT adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.
Ciphertext Translate (CSNBCTT and CSNBCTT1)

- **CSNBCTT1** allows the input text and output text to reside either in the caller's primary address space or in a data space. This allows you to translate more data with one call. However, a program using CSNBCTT1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

For CSNBCTT1, `text_id_in` and `text_id_out` are access list entry token (ALET) parameters of the data spaces containing the input text and output text.

**Format**

```call
CALL CSNBCTT(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier_in,
    key_identifier_out,
    text_length,
    text_in,
    initialization_vector_in,
    initialization_vector_out,
    text_out )
```

```call
CALL CSNBCTT1(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier_in,
    key_identifier_out,
    text_length,
    text_in,
    initialization_vector_in,
    initialization_vector_out,
    text_out,
    text_id_in,
    text_id_out )
```

**Parameters**

**return_code**

Direction: Output Type: Integer

The `return_code` specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.
exit_data_length
Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

exit_data
Direction: Input/Output Type: String

The data that is passed to the installation exit.

key_identifier_in
Direction: Input/Output Type: String

The 64-byte string of the internal key token containing the data translation (DATAXLAT) key, or the label of the CKDS record containing the DATAXLAT key used to encipher the input string.

key_identifier_out
Direction: Input/Output Type: String

The 64-byte string of an internal key token containing the DATAXLAT key, or the label of the CKDS record containing the DATAXLAT key, used to reencipher the encrypted text.

text_length
Direction: Input Type: Integer

The length of the ciphertext that is to be processed. The text length must be a multiple of 8 bytes. The maximum length of text is 2,147,836,647 bytes.

Note: Beginning in z/OS V1 R2, the MAXLEN value may still be specified in the options data set, but only the maximum value limit will be enforced.

text_in
Direction: Input Type: String

The text that is to be translated. The text is enciphered under the data translation key specified in the key_identifier_in parameter.

initialization_vector_in
Direction: Input Type: String

The 8-byte initialization vector that is used to decipher the input data. This parameter is the initialization vector used at the previous cryptographic node.

initialization_vector_out
Direction: Input Type: String
Ciphertext Translate (CSNBCTT and CSNBCTT1)

The 8-byte initialization vector that is used to encipher the input data. This is the new initialization vector used when the callable service enciphers the plaintext.

text_out
Direction: Output Type: String

The field where the callable service returns the translated text.

text_id_in
Direction: Input Type: Integer

For CSNBCTT1 only, the ALET of the text to be translated.

text_id_out
Direction: Input Type: Integer

For CSNBCTT1 only, the ALET of the text_out field that the application supplies.

Restrictions

The input ciphertext length must be an exact multiple of 8 bytes. The minimum length of the ciphertext that can be translated is 8 bytes.

You cannot use this service on a CDMF-only system.

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

The initialization vectors must have already been established between the communicating applications or must be passed with the data.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 76. Ciphertext translate required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>Cryptographic Coprocessor Feature</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
</tbody>
</table>
Ciphertext Translate (CSNBCTT and CSNBCTT1)

Table 76. Ciphertext translate required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM System z10 EC</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Decipher (CSNBDEC and CSNBDEC1)

Use the decipher callable service to decipher data in an address space or a data space using the cipher block chaining mode. ICSF supports these processing rules to decipher data. You choose the type of processing rule that the decipher callable service should use for block chaining.

<table>
<thead>
<tr>
<th>Processing Rule</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI X9.23</td>
<td>For cipher block chaining. The ciphertext must be an exact multiple of 8 bytes, but the plaintext will be 1 to 8 bytes shorter than the ciphertext. The text_length will also be reduced to show the original length of the plaintext.</td>
</tr>
<tr>
<td>CBC</td>
<td>For cipher block chaining. The ciphertext must be an exact multiple of 8 bytes, and the plaintext will have the same length.</td>
</tr>
<tr>
<td>CUSP</td>
<td>For cipher block chaining, but the ciphertext can be of any length. The plaintext will be the same length as the ciphertext.</td>
</tr>
<tr>
<td>IBM 4700</td>
<td>For cipher block chaining. The ciphertext must be an exact multiple of 8 bytes, but the plaintext will be 1 to 8 bytes shorter than the ciphertext. The text_length will also be reduced to show the original length of the plaintext.</td>
</tr>
<tr>
<td>IPS</td>
<td>For cipher block chaining, but the ciphertext can be of any length. The plaintext will be the same length as the ciphertext.</td>
</tr>
</tbody>
</table>

The cipher block chaining (CBC) mode uses an initial chaining value (ICV) in its processing. The first 8 bytes of ciphertext is deciphered and then the ICV is exclusive ORed with the resulting 8 bytes of data to form the first 8-byte block of plaintext. Thereafter, the 8-byte block of ciphertext is deciphered and exclusive ORed with the previous 8-byte block of ciphertext until all the ciphertext is deciphered.

The selection between single-DES decryption mode and triple-DES decryption mode is controlled by the length of the key supplied in the key_identifier parameter. If a single-length key is supplied, single-DES decryption is performed. If a double-length or triple-length key is supplied, triple-DES decryption is performed.

A different ICV may be passed on each call to the decipher callable service. However, the same ICV that was used in the corresponding encipher callable service must be passed.
Decipher (CSNBDEC and CSNBDEC1)

Short blocks are text lengths of 1 to 7 bytes. A short block can be the only block. Trailing short blocks are blocks of 1 to 7 bytes that follow an exact multiple of 8 bytes. For example, if the text length is 21, there are two 8-byte blocks and a trailing short block of 5 bytes. Because the DES and CDMF process only text in exact multiples of 8 bytes, some special processing is required to decipher such short blocks. Short blocks and trailing short blocks of 1 to 7 bytes of data are processed according to the Cryptographic Unit Support Program (CUSP) rules, or by the record chaining scheme devised by and used in the Information Protection System (IPS) in the IPS/CMS product.

These methods of treating short blocks and trailing short blocks do not increase the length of the ciphertext over the plaintext. If the plaintext was padded during encipherment, the length of the ciphertext will always be an exact multiple of 8 bytes.

ICSF supports these padding schemes:
- ANSI X9.23
- 4700-PAD

The callable service Decipher (without ALET) supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSNEDEC.

Choosing Between CSNBDEC and CSNBDEC1

CSNBDEC and CSNBDEC1 provide identical functions. When choosing which service to use, consider this:
- **CSNBDEC** requires the ciphertext and plaintext to reside in the caller’s primary address space. Also, a program using CSNBDEC adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.
- **CSNBDEC1** allows the ciphertext and plaintext to reside either in the caller’s primary address space or in a data space. This can allow you to decipher more data with one call. However, a program using CSNBDEC1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

For CSNBDEC1, `cipher_text_id` and `clear_text_id` are access list entry token (ALET) parameters of the data spaces containing the ciphertext and plaintext.

Format

```call
CALL CSNBDEC(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier,
    text_length,
    cipher_text,
    initialization_vector,
    rule_array_count,
    rule_array,
    chaining_vector,
    clear_text )
```
Decipher (CSNBDEC and CSNBDEC1)

CALL CSNBDEC1(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier,
    text_length,
    cipher_text,
    initialization_vector,
    rule_array_count,
    rule_array,
    chaining_vector,
    clear_text,
    cipher_text_id,
    clear_text_id )

Parameters

return_code
Direction: Output Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code
Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

exit_data_length
Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

exit_data
Direction: Input/Output Type: String

The data that is passed to the installation exit.

key_identifier
Direction: Input/Output Type: String

A 64-byte string that is the internal key token containing the data-encrypting key, or the label of a CKDS record containing a data-encrypting key, to be used for deciphering the data. If the key token or key label contains a single-length key, single-DES decryption is performed. If the key token or key label contains a double-length or triple-length key, triple-DES decryption is performed.
Decipher (CSNBDEC and CSNBDEC1)

On the IBM zSeries 990, IBM zSeries 890, z9 EC and z9 BC single and double length CIPHER and DECIPHER keys are also supported.

text_length
Direction: Input/Output  Type: Integer

On entry, you supply the length of the ciphertext. The maximum length of text is 214783647 bytes. A zero value for the text_length parameter is not valid. If the returned deciphered text (clear_text parameter) is a different length because of the removal of padding bytes, the value is updated to the length of the plaintext.

Note: Beginning in z/OS V1 R2, the MAXLEN value may still be specified in the options data set, but only the maximum value limit will be enforced.

The application program passes the length of the ciphertext to the callable service. The callable service returns the length of the plaintext to your application program.

cipher_text
Direction: Input  Type: String

The text to be deciphered.

initialization_vector
Direction: Input  Type: String

The 8-byte supplied string for the cipher block chaining. The first block of the ciphertext is deciphered and exclusive ORed with the initial chaining vector (ICV) to get the first block of cleartext. The input block is the next ICV. To decipher the data, you must use the same ICV used when you enciphered the data.

rule_array_count
Direction: Input  Type: Integer

The number of keywords you supply in the rule_array parameter. The value must be 1, 2, or 3.

rule_array
Direction: Input  Type: Character string

An array of 8-byte keywords providing the processing control information. The array is positional. See the keywords in Table 77 on page 227. The first keyword in the array is the processing rule. You choose the processing rule you want the callable service to use for deciphering the data. The second keyword is the ICV selection keyword. The third keyword (or the second if the ICV selection keyword is allowed to default) is the encryption algorithm to use.

The service will fail if keyword DES is specified in the rule_array in a CDMF-only system. The service will likewise fail if keyword CDMF is specified in the rule_array in a DES-only system.
Table 77. Keywords for the Decipher Rule Array Control Information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processing Rule (required)</strong></td>
<td></td>
</tr>
<tr>
<td>CBC</td>
<td>Performs ANSI X3.102 cipher block chaining. The data must be a multiple of 8 bytes. An OCV is produced and placed in the <code>chaining_vector</code> parameter. If the ICV selection keyword CONTINUE is specified, the CBC OCV from the previous call is used as the ICV for this call.</td>
</tr>
<tr>
<td>CUSP</td>
<td>Performs deciphering that is compatible with IBM’s CUSP and PCF products. The data can be of any length and does not need to be in multiples of 8 bytes. The ciphertext will be the same length as the plaintext. The CUSP/PCF OCV is placed in the <code>chaining_vector</code> parameter. If the ICV selection keyword CONTINUE is specified, the CUSP/PCF OCV from the previous call is used as the ICV for this call.</td>
</tr>
<tr>
<td>IPS</td>
<td>Performs deciphering that is compatible with IBM’s IPS product. The data can be of any length and does not need to be in multiples of 8 bytes. The ciphertext will be the same length as the plaintext. The IPS OCV is placed in the <code>chaining_vector</code> parameter. If the ICV selection keyword CONTINUE is specified, the IPS OCV from the previous call is used as the ICV for this call.</td>
</tr>
<tr>
<td>X9.23</td>
<td>Deciphers with cipher block chaining and text length reduced to the original value. This is compatible with the requirements in ANSI standard X9.23. The ciphertext length must be an exact multiple of 8 bytes. Padding is removed from the plaintext.</td>
</tr>
<tr>
<td>4700-PAD</td>
<td>Deciphers with cipher block chaining and text length reduced to the original value. The ciphertext length must be an exact multiple of 8 bytes. Padding is removed from the plaintext.</td>
</tr>
<tr>
<td><strong>ICV Selection (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>CONTINUE</td>
<td>This specifies taking the initialization vector from the output chaining vector (OCV) contained in the work area to which the <code>chaining_vector</code> parameter points. CONTINUE is valid only for processing rules CBC, IPS, and CUSP.</td>
</tr>
<tr>
<td>INITIAL</td>
<td>This specifies taking the initialization vector from the <code>initialization_vector</code> parameter. INITIAL is the default value.</td>
</tr>
<tr>
<td><strong>Encryption Algorithm (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>CDMF</td>
<td>This specifies using the Commercial Data Masking Facility and ignoring the token marking. You cannot use double- or triple-length keys with CDMF. The CDMF keyword, or tokens marked as CDMF, are only supported on an IBM @server zSeries 800 or IBM @server zSeries 900.</td>
</tr>
<tr>
<td>DES</td>
<td>This specifies using the data encryption standard and ignoring the token marking.</td>
</tr>
<tr>
<td>TOKEN</td>
<td>This specifies using the data encryption algorithm in the DATA key token. This is the default.</td>
</tr>
</tbody>
</table>
Decipher (CSNBDEC and CSNBDEC1)

Cipher Processing Rules on page 680 describes the cipher processing rules in detail.

**chaining_vector**

| Direction: Input/Output | Type: String |

An 18-byte field that ICSF uses as a system work area. Your application program must not change the data in this string. The chaining vector holds the output chaining vector (OCV) from the caller. The OCV is the first 8 bytes in the 18-byte string.

The direction is output if the ICV selection keyword of the rule_array parameter is INITIAL. The direction is input/output if the ICV selection keyword of the rule_array parameter is CONTINUE.

**clear_text**

| Direction: Output | Type: String |

The field where the callable service returns the deciphered text.

**cipher_text_id**

| Direction: Input | Type: Integer |

For CSNBDEC1 only, the ALET of the ciphertext to be deciphered.

**clear_text_id**

| Direction: Input | Type: Integer |

For CSNBDEC1 only, the ALET of the clear text supplied by the application.

**Restrictions**

The service will fail under these conditions:

- If the keyword DES is specified in the rule_array parameter in a CDMF-only system
- If the keyword CDMF is specified in the rule_array parameter in a DES-only system
- If the key token contains double or triple-length keys and triple-DES is not enabled.
- If the keyword CDMF is specified on a PCIXCC, CEX2C, or CEX3C.
- If a token is marked CDMF on a PCIXCC, CEX2C, or CEX3C.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

On a CCF system, only a DATA key token or DATA key label can be used in this service.

Single and double length CIPHER and DECIPHER keys are supported on a PCIXCC, CEX2C, or CEX3C.
Related Information

You **cannot** overlap the plaintext and ciphertext fields. For example:

```plaintext
pppppp
cccccc  is not supported.
```

```plaintext
ccccc
pppppp  is not supported.
```

```plaintext
ppppppcccccc  is supported.
```

P represents the plaintext and c represents the ciphertext.

On z990, z890, z9 EC or z9 BC system, the PCIXCC/CEX2C will support non destructive overlap. For example:

```plaintext
pppppp
cccccc  is supported.
```

“Cipher Processing Rules” on page 680 discusses the cipher processing rules.

The encipher callable services (CSNBENC and CSNBENC1) are described under “Encipher (CSNBENC and CSNBENC1)” on page 231.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

**Table 78. Decipher required hardware**

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td>If keyword CDMF is specified or if the token is marked as CDMF, the service fails.</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>If keyword CDMF is specified or if the token is marked as CDMF, the service fails.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

**Decode (CSNBDCO)**

Use the decode callable service (CSNBDCO) to decipher an 8-byte string using a clear key. The callable service uses the electronic code book (ECB) mode of the DES. (This service is available only on a DES-capable system.)
Considerations

If you have only a clear key, you are not limited to using only the encode and decode callable services.

- You can pass your clear key to the clear key import service, and get back a token that will allow you to use the encipher and decipher callable services.
- On an IBM iSeries zSeries 990 and subsequent releases, consider using the Symmetric Key Decipher service. (Symmetric Key Decipher (CSNBSYD, CSNBSYD1, CSNELSYD and CSNELSYD1) on page 253.

Format

```
CALL CSNBDCO(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    clear_key,
    cipher_text,
    clear_text)
```

Parameters

**return_code**

Direction: Output Type: Integer

The return code specifies the general result of the callable service. [Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. [Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**

Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**

Direction: Input/Output Type: String

The data that is passed to the installation exit.

**clear_key**

Direction: Input Type: String

The 8-byte clear key value that is used to decode the data.
Decode (CSNBDCO)

cipher_text
Direction: Input Type: String

The ciphertext that is to be decoded. Specify 8 bytes of text.

clear_text
Direction: Output Type: String

The 8-byte field where the plaintext is returned by the callable service.

Restrictions

You cannot use this service on a CDMF-only system.

Usage Notes

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Encipher (CSNBENC and CSNBENC1)

Use the encipher callable service to encipher data in an address space or a data space using the cipher block chaining mode. ICSF supports these processing rules to encipher data. You choose the type of processing rule that the encipher callable service should use for the block chaining.

<table>
<thead>
<tr>
<th>Processing Rule</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI X9.23</td>
<td>For block chaining not necessarily in exact multiples of 8 bytes. This process rule pads the plaintext so that ciphertext produced is an exact multiple of 8 bytes.</td>
</tr>
<tr>
<td>CBC</td>
<td>For block chaining in exact multiples of 8 bytes.</td>
</tr>
</tbody>
</table>
Encipher (CSNBENC and CSNBENC1)

CUSP
For block chaining not necessarily in exact multiples of 8 bytes. The ciphertext will be the same length as the plaintext.

IBM 4700
For block chaining not necessarily in exact multiples of 8 bytes. This process rule pads the plaintext so that the ciphertext produced is an exact multiple of 8 bytes.

IPS
For block chaining not necessarily in exact multiples of 8 bytes. The ciphertext will be the same length as the plaintext.

For more information about the processing rules, see Table 80 on page 236 and “Cipher Processing Rules” on page 680.

The cipher block chaining (CBC) mode of operation uses an initial chaining vector (ICV) in its processing. The ICV is exclusive ORed with the first 8 bytes of plaintext prior to the encryption step, and thereafter, the 8-byte block of ciphertext just produced is exclusive ORed with the next 8-byte block of plaintext, and so on. This disguises any pattern that may exist in the plaintext.

The selection between single-DES encryption mode and triple-DES encryption mode is controlled by the length of the key supplied in the key_identifier parameter. If a single-length key is supplied, single-DES encryption is performed. If a double-length or triple-length key is supplied, triple-DES encryption is performed.

To nullify the CBC effect on the first 8-byte block, supply 8 bytes of zero. However, the ICV may require zeros.

Cipher block chaining also produces a resulting chaining value called the output chaining vector (OCV). The application can pass the OCV as the ICV in the next encipher call. This results in record chaining.

Note that the OCV that results is the same, whether an encipher or a decipher callable service was invoked, assuming the same text, ICV, and key were used.

Short blocks are text lengths of 1 to 7 bytes. A short block can be the only block. Trailing short blocks are blocks of 1 to 7 bytes that follow an exact multiple of 8 bytes. For example, if the text length is 21, there are two 8-byte blocks, and a trailing short block of 5 bytes. Short blocks and trailing short blocks of 1 to 7 bytes of data are processed according to the Cryptographic Unit Support Program (CUSP) rules, or by the record chaining scheme devised by and used by the Information Protection System (IPS) in the IPS/CMS program product. These methods of treating short blocks and trailing short blocks do not increase the length of the ciphertext over the plaintext.

An alternative method is to pad the plaintext and produce a ciphertext that is longer than the plaintext. The plaintext can be padded with up to 8 bytes using one of several padding schemes. This padding produces a ciphertext that is an exact multiple of 8 bytes long.

If the ciphertext is to be transmitted over a network, where one or more intermediate nodes will use the ciphertext translate callable service, the ciphertext must be produced using one of these methods of padding:

- ANSI X9.23
Encipher (CSNBENC and CSNBENC1)

- 4700

If the cleartext is already a multiple of 8, the ciphertext can be created using any processing rule.

Because of padding, the returned ciphertext length is longer than the provided plaintext; the `text_length` parameter will have been modified. The returned ciphertext field should be 8 bytes longer than the length of the plaintext to accommodate the maximum amount of padding. You should provide this extension in your installation's storage because ICSF cannot detect whether the extension was done.

The minimum length of data that can be enciphered is one byte. Beginning in z/OS V1 R2, the MAXLEN value may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647).

**Attention:** If you lose the data-encrypting key under which the data (plaintext) is enciphered, the data enciphered under that key (ciphertext) cannot be recovered.

The callable service Encipher (without ALET) supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSNEENC.

Choosing between CSNBENC and CSNBENC1

CSNBENC and CSNBENC1 provide identical functions. When choosing which service to use, consider this:

- **CSNBENC** requires the cleartext and ciphertext to reside in the caller's primary address space. Also, a program using CSNBENC adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.

- **CSNBENC1** allows the cleartext and ciphertext to reside either in the caller's primary address space or in a data space. This can allow you to encipher more data with one call. However, a program using CSNBENC1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

For CSNBENC1, `clear_text_id` and `cipher_text_id` are access list entry token (ALET) parameters of the data spaces containing the cleartext and ciphertext.

Format

```call
CALL CSNBENC(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier,
    text_length,
    clear_text,
    initialization_vector,
    rule_array_count,
    rule_array,
    pad_character,
    chaining_vector,
    cipher_text )
```
Encipher (CSNBENC and CSNBENC1)

```
CALL CSNBENC1(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier,
    text_length,
    clear_text,
    initialization_vector,
    rule_array_count,
    rule_array,
    pad_character,
    chaining_vector,
    cipher_text,
    clear_text_id,
    cipher_text_id );
```

Parameters

**return_code**

Direction: Output Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**

Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**

Direction: Input/Output Type: String

The data that is passed to the installation exit.

**key_identifier**

Direction: Input/Output Type: String

A 64-byte string that is the internal key token containing the data-encrypting key, or the label of a CKDS record containing the data-encrypting key, to be used for encrypting the data. If the key token or key label contains a single-length key, single-DES encryption is performed. If the key token or key label contains a double-length or triple-length key, triple-DES encryption is performed.
Encipher (CSNBENC and CSNBENC1)

On an IBM @server zSeries 990 and subsequent releases, single and double length CIPHER and ENCIPHER keys are also supported.

text_length

Direction: Input/Output  
Type: Integer

On entry, the length of the plaintext (clear_text parameter) you supply. The maximum length of text is 2,14783647 bytes. A zero value for the text_length parameter is not valid. If the returned enciphered text (cipher_text parameter) is a different length because of the addition of padding bytes, the value is updated to the length of the ciphertext.

Note: Beginning in z/OS V1 R2, the MAXLEN value may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647).

The application program passes the length of the plaintext to the callable service. The callable service returns the length of the ciphertext to the application program.

clear_text

Direction: Input  
Type: String

The text that is to be enciphered.

initialization_vector

Direction: Input  
Type: String

The 8-byte supplied string for the cipher block chaining. The first 8 bytes (or less) block of the data is exclusive ORed with the ICV and then enciphered. The input block is enciphered and the next ICV is created. You must use the same ICV to decipher the data.

rule_array_count

Direction: Input  
Type: Integer

The number of keywords you supply in the rule_array parameter. The value must be 1, 2, or 3.

rule_array

Direction: Input  
Type: Character string

An array of 8-byte keywords providing the processing control information. The array is positional. See the keywords in Table 80 on page 236. The first keyword in the array is the processing rule. You choose the processing rule you want the callable service to use for enciphering the data. The second keyword is the ICV selection keyword. The third keyword (or the second if the ICV selection keyword is allowed to default to INITIAL) is the encryption algorithm to use.

The service will fail if keyword DES is specified in the rule_array in a CDMF-only system. The service will likewise fail if the keyword CDMF is specified in the rule_array in a DES-only system.
## Table 80. Keywords for the Encipher Rule Array Control Information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processing Rule (required)</strong></td>
<td></td>
</tr>
<tr>
<td>CBC</td>
<td>Performs ANSI X3.102 cipher block chaining. The data must be a multiple of 8 bytes. An OCV is produced and placed in the <code>chaining_vector</code> parameter. If the ICV selection keyword CONTINUE is specified, the CBC OCV from the previous call is used as the ICV for this call.</td>
</tr>
<tr>
<td>CUSP</td>
<td>Performs ciphering that is compatible with IBM's CUSP and PCF products. The data can be of any length and does not need to be in multiples of 8 bytes. The ciphertext will be the same length as the plaintext. The CUSP/PCF OCV is placed in the <code>chaining_vector</code> parameter. If the ICV selection keyword CONTINUE is specified, the CUSP/PCF OCV from the previous call is used as the ICV for this call.</td>
</tr>
<tr>
<td>IPS</td>
<td>Performs ciphering that is compatible with IBM's IPS product. The data may be of any length and does not need to be in multiples of 8 bytes. The ciphertext will be the same length as the plaintext. The IPS OCV is placed in the <code>chaining_vector</code> parameter. If the ICV selection keyword CONTINUE is specified, the IPS OCV from the previous call is used as the ICV for this call.</td>
</tr>
<tr>
<td>X9.23</td>
<td>Performs cipher block chaining with 1 to 8 bytes of padding. This is compatible with the requirements in ANSI standard X9.23. If the data is not in exact multiples of 8 bytes, X9.23 pads the plaintext so that the ciphertext produced is an exact multiple of 8 bytes. The plaintext is padded to the next multiple 8 bytes, even if this adds 8 bytes. An OCV is produced.</td>
</tr>
<tr>
<td>4700-PAD</td>
<td>Performs padding by extending the user's plaintext with the caller's specified pad character, followed by a one-byte binary count field that contains the total number of bytes added to the message. 4700-PAD pads the plaintext so that the ciphertext produced is an exact multiple of 8 bytes. An OCV is produced.</td>
</tr>
<tr>
<td><strong>ICV Selection (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>CONTINUE</td>
<td>This specifies taking the initialization vector from the output chaining vector (OCV) contained in the work area to which the <code>chaining_vector</code> parameter points. CONTINUE is valid only for processing rules CBC, IPS, and CUSP.</td>
</tr>
<tr>
<td>INITIAL</td>
<td>This specifies taking the initialization vector from the <code>initialization_vector</code> parameter. INITIAL is the default value.</td>
</tr>
<tr>
<td><strong>Encryption Algorithm (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>CDMF</td>
<td>This specifies using the Commercial Data Masking Facility and ignoring the token marking. You cannot use double-length or triple-length keys with CDMF. The CDMF keyword, or tokens marked as CDMF, are only supported on an IBM @server zSeries 800 or IBM @server zSeries 900.</td>
</tr>
<tr>
<td>DES</td>
<td>This specifies using the data encryption standard and ignoring the token marking.</td>
</tr>
<tr>
<td>TOKEN</td>
<td>This specifies using the data encryption algorithm in the DATA key token. TOKEN is the default.</td>
</tr>
</tbody>
</table>
Encipher (CSNBENC and CSNBENC1)

These recommendations help the caller determine which encipher processing rule to use:

- If you are exchanging enciphered data with a specific implementation, for example, CUSP or ANSI X9.23, use that processing rule.
- If the ciphertext translate callable service is to be invoked on the enciphered data at an intermediate node, ensure that the ciphertext is a multiple of 8 bytes. Use CBC, X9.23, or 4700-PAD to prevent the creation of ciphertext that is not a multiple of 8 bytes and that cannot be processed by the ciphertext translate callable service.
- If the ciphertext length must be equal to the plaintext length and the plaintext length cannot be a multiple of 8 bytes, use either the IPS or CUSP processing rule.

"Cipher Processing Rules" on page 680 describes the cipher processing rules in detail.

**pad_character**

Direction: Input  
Type: Integer

An integer, 0 to 255, that is used as a padding character for the 4700-PAD process rule (rule_array parameter).

**chaining_vector**

Direction: Input/Output  
Type: String

An 18-byte field that ICSF uses as a system work area. Your application program must not change the data in this string. The chaining vector holds the output chaining vector (OCV) from the caller. The OCV is the first 8 bytes in the 18-byte string.

The direction is output if the ICV selection keyword of the rule_array parameter is INITIAL.

The direction is input/output if the ICV selection keyword of the rule_array parameter is CONTINUE.

**cipher_text**

Direction: Output  
Type: String

The enciphered text the callable service returns. The length of the ciphertext is returned in the text_length parameter. The cipher_text may be 8 bytes longer than the length of the clear_text field because of the padding that is required for some processing rules.

**clear_text_id**

Direction: Input  
Type: Integer

For CSNBENC1 only, the ALET of the clear text to be enciphered.

**cipher_text_id**

Direction: Input  
Type: Integer

For CSNBENC1 only, the ALET of the ciphertext that the application supplied.
Encipher (CSNBENC and CSNBENC1)

Restrictions

The service will fail under these conditions:

- If the keyword DES is specified in the rule_array parameter in a CDMF-only system
- If the keyword CDMF is specified in the rule_array parameter in a DES-only system
- If the key token contains double- or triple-length keys and triple-DES is not enabled.
- If the keyword CDMF is specified on a PCIXCC, CEX2C, or CEX3C.
- If a token is marked CDMF on a PCIXCC, CEX2C, or CEX3C.

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

On a CCF system, only a DATA key token or DATA key label can be used in this service.

Single and double length CIPHER and ENCIPHER keys are supported on a PCIXCC, CEX2C, or CEX3C.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>If keyword CDMF is specified or if the token is marked as CDMF, the service fails.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>If keyword CDMF is specified or if the token is marked as CDMF, the service fails.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>If keyword CDMF is specified or if the token is marked as CDMF, the service fails.</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

Related Information

You cannot overlap the plaintext and ciphertext fields. For example:

```
  pppppp
  cccccc  is not supported.

  cccccc
  pppppp  is not supported.
```
### Encipher (CSNBENC and CSNBENC1)

pppppcccccc is supported.

P represents the plaintext and c represents the ciphertext.

On z990, z890, z9 EC and z9 BC systems, the PCIXCC/CEX2C will support non-destructive overlap. For example:

cccccc  
ppppp  is supported.

The method used to produce the OCV is the same with the CBC, 4700-PAD, and X9.23 processing rules. However, that method is different from the method used by the CUSP and IPS processing rules.

"Cipher Processing Rules" on page 680 discusses the cipher processing rules.

The decipher callable services (CSNBDEC and CSNBDEC1) are described under "Decipher (CSNBDEC and CSNBDEC1)" on page 223.

---

### Encode (CSNBECO)

Use the encode callable service (CSNBECO) to encipher an 8-byte string using a clear key. The callable service uses the electronic code book (ECB) mode of the DES. (This service is available only on a DES-capable system.)

#### Considerations

If you have only a clear key, you are not limited to using just the encode and decode callable services.

- You can pass your clear key to the clear key import service, and get back a token that will allow you to use the encipher and decipher callable services.
- On an IBM @server zSeries 990 and subsequent releases, consider using the Symmetric Key Encipher service ("Symmetric Key Encipher (CSNBSYE, CSNBSYE1, CSNESYE and CSNESYE1)" on page 260).

#### Format

```plaintext
CALL CSNBECO(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    clear_key,
    clear_text,
    cipher_text)
```

#### Parameters

- **return_code**
  
  Direction: Output  
  Type: Integer  
  
  The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.
Encode (CSNBECO)

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the **exit_data** parameter.

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**clear_key**

Direction: Input  
Type: String

The 8-byte clear key value that is used to encode the data.

**clear_text**

Direction: Input  
Type: String

The plaintext that is to be encoded. Specify 8 bytes of text.

**cipher_text**

Direction: Output  
Type: String

The 8-byte field where the ciphertext is returned by the callable service.

**Restrictions**

You cannot use this service on a CDMF-only system.

**Usage Notes**

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 82. Encode required hardware*

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>Cryptographic Coprocessor Feature</td>
<td></td>
</tr>
</tbody>
</table>
Table 82. Encode required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 990</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
</tbody>
</table>

**Symmetric Algorithm Decipher (CSNBSAD, CSNBSAD1, CSNESAD and CSNESAD1)**

The symmetric algorithm decipher callable service decipheres data with the AES algorithm. Data is deciphered that has been enciphered in either CBC mode or ECB mode.

You can specify that the clear text data was padded before encryption using the method described in the PKCS standards. In this case, the callable service will remove the padding bytes and return the unpadded clear text data. PKCS padding is described in [“PKCS Padding Method” on page 683](#).

The callable service Symmetric Algorithm Decipher supports invocation in AMODE (64). The callable service names for AMODE (64) invocation are CSNESAD and CSNESAD1.

**Choosing Between CSNBSAD and CSNBSAD1 or CSNESAD and CSNESAD1**

CSNBSAD, CSNBSAD1, CSNESAD, and CSNESAD1 provide identical functions. When choosing which service to use, consider this:

- CSNBSAD and CSNESAD require the cipher text and plaintext to reside in the caller’s primary address space. Also, a program using CSNBSAD adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.

- CSNBSAD1 and CSNESAD1 allow the cipher text and plaintext to reside either in the caller’s primary address space or in a data space. This can allow you to decipher more data with one call. However, a program using CSNBSAD1 and CSNESAD1 does not adhere to the IBM CCA: Cryptographic API and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

For CSNBSAD1 and CSNESAD1, `cipher_text_id` and `clear_text_id` are access list entry token (ALET) parameters of the data spaces containing the cipher text and plaintext.
Symmetric Algorithm Decipher (CSNBSAD, CSNBSAD1, CSNESAD and CSNESAD1)

Format

CALL CSNBSAD(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  rule_array_count,
  rule_array,
  key_identifier_length,
  key_identifier,
  key_parms_length,
  key_parms,
  block_size,
  initialization_vector_length,
  initialization_vector,
  chain_data_length,
  chain_data,
  cipher_text_length,
  cipher_text,
  clear_text_length,
  clear_text,
  optional_data_length,
  optional_data)

CALL CSNBSAD1(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  rule_array_count,
  rule_array,
  key_length,
  key_identifier,
  key_parms_length,
  key_parms,
  block_size,
  initialization_vector_length,
  initialization_vector,
  chain_data_length,
  chain_data,
  cipher_text_length,
  cipher_text,
  clear_text_length,
  clear_text,
  optional_data_length,
  optional_data,
  cipher_text_id
  clear_text_id)

Parameters

return_code
Direction: Output Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code
Direction: Output Type: Integer
Symmetric Algorithm Decipher (CSNBSAD, CSNBSAD1, CSNESAD and CSNESAD1)

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit_data_length**

Direction: Input  Type: Integer

The length of the `exit_data` parameter. It is recommended that this parameter be 0.

**exit_data**

Direction: Ignored  Type: String

The data that is passed to the installation exit. Reserved for future use.

**rule_array_count**

Direction: Input  Type: Integer

The number of keywords you supplied in the `rule_array` parameter. The value may be 2, 3 or 4.

**rule_array**

Direction: Input  Type: String

An array of 8-byte keywords providing the processing control information. The keywords must be in contiguous storage, left-justified and padded on the right with blanks.

**Table 83. Symmetric Algorithm Decipher Rule Array Keywords**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithm (one keyword, required)</strong></td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>Specifies that the Advanced Encryption Standard (AES) algorithm is to be used. The block size is 16 bytes. The key length may be 16, 24, or 32 bytes.</td>
</tr>
<tr>
<td><strong>Processing Rule (optional - zero or one keyword)</strong></td>
<td></td>
</tr>
<tr>
<td>CBC</td>
<td>Performs encryption in cipher block chaining (CBC) mode. The text length must be a multiple of the AES block size (16-bytes). This is the default value.</td>
</tr>
<tr>
<td>ECB</td>
<td>Performs encryption in electronic code book (ECB) mode. The text length must be a multiple of the AES block size (16-bytes).</td>
</tr>
<tr>
<td>PKCS-PAD</td>
<td>Deciphers with cipher block chaining and text length reduced to the original value. The ciphertext length must be an exact multiple of 16 bytes. Padding is removed from the plaintext.</td>
</tr>
<tr>
<td><strong>Key Rule (required)</strong></td>
<td></td>
</tr>
<tr>
<td>KEYIDENT</td>
<td>This indicates that the value in the <code>key_identifier</code> parameter is either an internal key token or the label of a key token in the CKDS. The key must be a secure AES key, that is, enciphered under the current master key.</td>
</tr>
<tr>
<td><strong>ICV Selection (optional - zero or one keyword)</strong></td>
<td></td>
</tr>
</tbody>
</table>
Symmetric Algorithm Decipher (CSNBSAD, CSNBSAD1, CSNESAD and CSNESAD1)

Table 83. Symmetric Algorithm Decipher Rule Array Keywords (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIAL</td>
<td>This specifies that this is the first request of a sequence of chained requests, and indicates that the initialization vector should be taken from the initialization_vector parameter. This is the default value.</td>
</tr>
<tr>
<td>CONTINUE</td>
<td>This specifies that this request is part of a sequence of chained requests, and is not the first request in that sequence. The initialization vector will be taken from the work area identified in the chain_data parameter. This keyword is only valid for processing rule CBC.</td>
</tr>
</tbody>
</table>

key_identifier_length

Direction: Input  Type: Integer

The length of the key_identifier parameter. The length must be 64 bytes.

key_identifier

Direction: Input  Type: String

This specifies an internal secure AES token or the labelname of a secure AES token in the CKDS. Normal CKDS labelname syntax is required.

The AES key identifier must be an encrypted key contained in an internal key token, where the key is enciphered under the AES master key. The key can be 128-, 192-, or 256-bits in length.

key_parms_length

Direction: Input  Type: Integer

The length of the key_parms parameter. This must be 0.

key_parms

Direction: Ignored  Type: String

This parameter is ignored. It is reserved for future use.

block_size

Direction: Input  Type: Integer

The block size for the cryptographic algorithm. AES requires the block size to be 16.

initialization_vector_length

Direction: Input  Type: Integer

The length of the initialization_vector parameter. The length should be equal to the block length for the algorithm specified. This parameter is ignored if the process rule is ECB.
Symmetric Algorithm Decipher (CSNBSAD, CSNBSAD1, CSNESAD and CSNESAD1)

**initialization_vector**
Direction: Input  
Type: String

This parameter contains the initialization vector (IV) for CBC mode decryption, including CBC mode invoked using the PKCS-PAD keyword. This parameter is ignored if the process rule is ECB. For AES CBC mode decryption, the initialization vector length must be 16 bytes, the length of an AES block. The IV must be the same value used when the data was encrypted.

**chain_data_length**
Direction: Input/Output  
Type: Integer

The length of the chain_data parameter. On input it contains the length of the buffer provided with parameter chain_data. On output, it is updated with the length of the data returned in the chain_data parameter.

**chain_data**
Direction: Input/Output  
Type: String

A buffer that is used as a work area for sequences of chained symmetric algorithm decipher requests. When the keyword INITIAL is used, this is an output parameter and receives data that is needed when deciphering the next part of the input data. When the keyword CONTINUE is used, this is an input/output parameter; the value received as output from the previous call in the sequence is provided as input to this call, and in turn this call will return new chain_data that will be used as input on the next call. When CONTINUE is used, both the data (chain_data parameter) and the length (chain_data_length parameter) must be the same values that were received in these parameters as output on the preceding call to the service in the chained sequence.

The exact content and layout of chain_data is not described. For AES CBC encryption, the field must be at least 32-bytes in length. For AES ECB encryption the field is not used and any length is acceptable including zero.

**cipher_text_length**
Direction: Input  
Type: Integer

The length of the cipher text. The length must be a multiple of the algorithm block size.

**cipher_text**
Direction: Input  
Type: String

The text to be deciphered.

**clear_text_length**
Direction: Input/Output  
Type: Integer

On input, this parameter specifies the size of the storage pointed to by the clear_text parameter. On output, this parameter has the actual length of the text stored in the clear_text parameter.
Symmetric Algorithm Decipher (CSNBSAD, CSNBSAD1, CSNESAD and CSNESAD1)

If process rule PKCS-PAD is used, the clear text length will be less than the cipher text length since padding bytes are removed.

clear_text
Direction: Output Type: String

The deciphered text the service returns.

optional_data_length
Direction: Input Type: Integer

The length of the optional_data parameter. This parameter must be 0.

optional_data
Direction: Ignored Type: String

Optional data required by a specified algorithm.

cipher_text_id
Direction: Input Type: Integer

For CSNBSAD1 and CSNESAD1 only, the ALET of the dataspace in which the cipher_text parameter resides.

clear_text_id
Direction: Input Type: Integer

For CSNBSAD1 and CSNESAD1 only, the ALET of the dataspace in which the clear_text parameter resides.

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Table 84. Symmetric Algorithm Decipher required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td></td>
<td>This service is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td>This service is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td></td>
<td>This service is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td></td>
<td>This service is not supported.</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Secure AES keys requires the Nov. 2008 or later licensed internal code (LIC)</td>
</tr>
</tbody>
</table>
Symmetric Algorithm Decipher (CSNBSAD, CSNBSAD1, CSNESAD and CSNESAD1)

Table 84. Symmetric Algorithm Decipher required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Secure AES keys requires the Nov. 2008 or later licensed internal code (LIC)</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

Symmetric Algorithm Encipher (CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1)

The symmetric algorithm encipher callable service enciphers data with the AES algorithm. Data is enciphered that has been deciphered in either CBC mode or ECB mode.

The callable service Symmetric Algorithm Encipher supports invocation in AMODE (64). The callable service names for AMODE (64) invocation are CSNBSAE and CSNESAE1.

Choosing between CSNBSAE and CSNBSAE1 or CSNESAE and CSNESAE1

CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1 provide identical functions. When choosing which service to use, consider this:

- CSNBSAE and CSNESAE require the cipher text and plaintext to reside in the caller's primary address space. Also, a program using CSNBSAE adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.

- CSNBSAE1 and CSNESAE1 allow the cipher text and plaintext to reside either in the caller's primary address space or in a data space. This can allow you to encipher more data with one call. However, a program using CSNBSAE1 and CSNESAE1 does not adhere to the IBM CCA: Cryptographic API and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

For CSNBSAE1 and CSNESAE1, cipher_text_id and clear_text_id are access list entry token (ALET) parameters of the data spaces containing the cipher text and plaintext.
Symmetric Algorithm Encipher (CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1)

Format

CALL CSNBSAE(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier_length,
    key_identifier,
    key_parms_length,
    key_parms,
    block_size,
    initialization_vector_length,
    initialization_vector,
    chain_data_length,
    chain_data,
    clear_text_length,
    clear_text,
    cipher_text_length,
    cipher_text,
    optional_data_length,
    optional_data)

CALL CSNBSAE1(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier_length,
    key_identifier,
    key_parms_length,
    key_parms,
    block_size,
    initialization_vector_length,
    initialization_vector,
    chain_data_length,
    chain_data,
    clear_text_length,
    clear_text,
    cipher_text_length,
    cipher_text,
    optional_data_length,
    optional_data
    clear_text_id
    cipher_text_id)

Parameters

return_code
Direction: Output  Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code
Direction: Output  Type: Integer
Symmetric Algorithm Encipher (CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1)

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

exit_data_length
Direction: Input Type: Integer

The length of the exit_data parameter. It is recommended that this parameter be 0.

exit_data
Direction: Ignored Type: String

The data that is passed to the installation exit. Reserved for future use.

rule_array_count
Direction: Input Type: Integer

The number of keywords you supplied in the rule_array parameter. The value may be 2, 3 or 4.

rule_array
Direction: Input Type: String

This keyword provides control information to the callable service. The keywords must be eight bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

Table 85. Symmetric Algorithm Encipher Rule Array Keywords

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm (one keyword, required)</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>Specifies that the Advanced Encryption Standard (AES) algorithm will be used. The block size is 16-bytes, and the key length may be 16-, 24-, or 32-bytes (128-, 192-, 256-bits).</td>
</tr>
<tr>
<td>Processing Rule (optional - zero or one keyword)</td>
<td></td>
</tr>
<tr>
<td>CBC</td>
<td>Performs encryption in cipher block chaining (CBC) mode. The text length must be a multiple of the AES block size (16-bytes). This is the default value.</td>
</tr>
<tr>
<td>ECB</td>
<td>Performs encryption in electronic code book (ECB) mode. The text length must be a multiple of the AES block size (16-bytes).</td>
</tr>
<tr>
<td>PKCS-PAD</td>
<td>Performs encryption in cipher block chaining (CBC) mode, but the data is padded using PKCS padding rules. The length of the clear text data does not have to be a multiple of the cipher block length. The cipher text will be longer than the clear text by at least one byte, and up to 16-bytes. The PKCS padding method is described in &quot;PKCS Padding Method&quot; on page 683.</td>
</tr>
</tbody>
</table>
Symmetric Algorithm Encipher (CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1)

Table 85. Symmetric Algorithm Encipher Rule Array Keywords (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYIDENT</td>
<td>This indicates that the value in the key_identifier parameter is either an internal key token or the label of a key token in the CKDS. The key must be a secure AES key, that is, enciphered under the current master key.</td>
</tr>
</tbody>
</table>

ICV Selection (optional - zero or one keyword)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIAL</td>
<td>This specifies that this is the first request of a sequence of chained requests, and indicates that the initialization vector should be taken from the initialization_vector parameter. This is the default value.</td>
</tr>
<tr>
<td>CONTINUE</td>
<td>This specifies that this request is part of a sequence of chained requests, and is not the first request in that sequence. The initialization vector will be taken from the work area identified in the chain_data parameter. This keyword is only valid for processing rule CBC.</td>
</tr>
</tbody>
</table>

key_identifier_length

Direction: Input  
Type: Integer

The length of the key_identifier parameter. The length must be 64 bytes.

key_identifier

Direction: Input  
Type: String

This specifies an internal secure AES token or the labelname of a secure AES token in the CKDS. Normal CKDS labelname syntax is required.

The AES key identifier must be an encrypted key contained in an internal key token, where the key is enciphered under the AES master key. The key can be 128-, 192-, or 256-bits in length.

key_parms_length

Direction: Input  
Type: Integer

The length of the key_parms parameter in bytes. It must be set to 0.

key_parms

Direction: Ignored  
Type: String

This parameter is ignored. It is reserved for future use.

block_size

Direction: Input  
Type: Integer

The block size for the cryptographic algorithm. AES requires the block size to be 16.

initialization_vector_length

Direction: Input  
Type: Integer
Symmetric Algorithm Encipher (CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1)

The length of the initialization_vector parameter in bytes. This parameter is ignored if the process rule is ECB.

**initialization_vector**

Direction: Input
Type: String

This parameter contains the initialization vector (IV) for CBC mode encryption, including the CBC mode invoked using the PKCS-PAD keyword. This parameter is ignored if the process rule is ECB. For AES CBC mode encryption, the initialization vector length must be 16 bytes, the length of an AES block. The same IV must be used when decrypting the data.

**chain_data_length**

Direction: Input/Output
Type: Integer

The length in bytes of the chain_data parameter. On input it contains the length of the buffer provided with parameter chain_data. On output, it is updated with the length of the data returned in the chain_data parameter.

**chain_data**

Direction: Input/Output
Type: String

A buffer that is used as a work area for sequences of chained symmetric algorithm encipher requests. When the keyword INITIAL is used, this is an output parameter and receives data that is needed when enciphering the next part of the input data. When the keyword CONTINUE is used, this is an input/output parameter; the value received as output from the previous call in the sequence is provided as input to this call, and in turn this call will return new chain_data that will be used as input on the next call. When CONTINUE is used, both the data (chain_data parameter) and the length (chain_data_length parameter) must be the same values that were received in these parameters as output on the preceding call to the service in the chained sequence.

The exact content and layout of chain_data is not described. For AES CBC encryption, the field must be at least 32-bytes in length. For AES ECB encryption the field is not used and any length is acceptable including zero.

**clear_text_length**

Direction: Input
Type: Integer

The length of the clear text data in the clear_text parameter. Unless process rule PKCS-PAD is used, the length must be a multiple of the algorithm block size. The length must be 1 or greater.

**clear_text**

Direction: Input
Type: String

The text to be enciphered.

**cipher_text_length**

Direction: Input/Output
Type: Integer
Symmetric Algorithm Encipher (CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1)

On input, this parameter specifies the size of the storage pointed to by the cipher_text parameter. On output, this parameter has the actual length of the text stored in the buffer addressed by the cipher_text parameter.

If process rule PKCS-PAD is used, the cipher text length will exceed the clear text length by at least one byte, and up to 16-bytes. For other process rules, the cipher text length will be equal to the clear text length.

cipher_text
Direction: Output Type: String

The enciphered text the service returns.

optional_data_length
Direction: Input Type: Integer

The length of the optional_data parameter. This parameter is reserved for future use. It must be set to 0.

optional_data
Direction: Ignored Type: String

The optional data used in processing the request. This parameter is ignored.

cipher_text_id
Direction: Input Type: Integer

For CSNBSAE1 and CSNESAE1 only, the ALET of the dataspace in which the cipher_text parameter resides.

clear_text_id
Direction: Input Type: Integer

For CSNBSAE1 and CSNESAE1 only, the ALET of the dataspace in which the clear_text parameter resides.

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

Table 86. Symmetric Algorithm Encipher required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td></td>
<td>This service is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Symmetric Algorithm Encipher (CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1)

Table 86. Symmetric Algorithm Encipher required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 990</td>
<td></td>
<td>This service is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Secure AES keys require the Nov. 2008 or</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td>later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Secure AES keys require the Nov. 2008 or</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td>later licensed internal code (LIC).</td>
</tr>
</tbody>
</table>

Symmetric Key Decipher (CSNBSYD, CSNBSYD1, CSNESYD and CSNESYD1)

Use the symmetric key decipher callable service to decipher data using the cipher block chaining, electronic code book or cipher feedback modes. ICSF supports these processing rules to decipher data. You choose the type of processing rule that the decipher callable service should use for block chaining.

<table>
<thead>
<tr>
<th>Processing Rule</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI X9.23</td>
<td>For cipher block chaining. The ciphertext must be an exact multiple of 8 bytes, but the plaintext will be 1 to 8 bytes shorter than the ciphertext.</td>
</tr>
<tr>
<td>CBC</td>
<td>For cipher block chaining. The ciphertext must be an exact multiple of 8 bytes, and the plaintext will have the same length.</td>
</tr>
<tr>
<td>CFB</td>
<td>Performs cipher feedback encryption. The ciphertext can be of any length. The plaintext will have the same length as the ciphertext.</td>
</tr>
<tr>
<td>CUSP</td>
<td>For cipher block chaining, but the ciphertext can be of any length. The plaintext will be the same length as the ciphertext.</td>
</tr>
<tr>
<td>ECB</td>
<td>Performs electronic code book encryption. The text length must be a multiple of the block size for the specified algorithm.</td>
</tr>
<tr>
<td>IPS</td>
<td>For cipher block chaining, but the ciphertext can be of any length. The plaintext will be the same length as the ciphertext.</td>
</tr>
<tr>
<td>PKCS-PAD</td>
<td>For cipher block chaining, but the ciphertext must be an exact multiple of the block length (8 bytes for DES and 16 bytes for AES). The plaintext will be 1 to 8 bytes shorter for DES and 1 to 16 bytes shorter for AES than the ciphertext.</td>
</tr>
</tbody>
</table>
Symmetric Key Decipher (CSNBSYD, CSNBSYD1, CSNESYD and CSNESYD1)

The Advanced Encryption Standard (AES) and Data Encryption Standard (DES) are supported. AES encryption uses a 128-, 192-, or 256-bit key. The CBC, ECB, and CFB modes are supported.

This service supports electronic code book (ECB), cipher block chaining (CBC), and cipher feedback (CFB) modes. The CBC and CFB modes of operation use an initial chaining vector (ICV) in their processing. During CBC mode processing, the ICV is exclusive ORed with the first block of plaintext after the decryption step, and thereafter, each block of ciphertext is exclusive ORed with the next block of plaintext after decryption, and so on. For CFB mode processing, the ICV is first encrypted, then exclusive ORed with the first block of ciphertext, and thereafter, the block of exclusive ORed data is encrypted then exclusive ORed with the next block of ciphertext, and so on.

Both Cipher block chaining and Cipher feedback mode also produce a resulting chaining value called the output chaining vector (OCV). The application can pass the OCV as the ICV in the next encipher call. This results in record chaining.

The electronic code book mode does not use the initial chaining vector.

The selection between single-DES decryption mode and triple-DES decryption mode is controlled by the length of the key supplied in the key_identifier parameter. If a single-length key is supplied, single-DES decryption is performed. If a double-length or triple-length key is supplied, triple-DES decryption is performed.

The AES or DES key is specified as a clear key value, an internal clear key token, or the label name of a clear key or an encrypted key in the CKDS.

The callable service Symmetric Key Decipher (without ALET) supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSNESYD.

Choosing Between CSNBSYD and CSNBSYD1

CSNBSYD and CSNBSYD1 provide identical functions. When choosing which service to use, consider this:

- **CSNBSYD** requires the ciphertext and plaintext to reside in the caller’s primary address space. Also, a program using CSNBSYD adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.
- **CSNBSYD1** allows the ciphertext and plaintext to reside either in the caller’s primary address space or in a data space. This can allow you to decipher more data with one call. However, a program using CSNBSYD1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

For CSNBSYD1, cipher_text_id and clear_text_id are access list entry token (ALET) parameters of the data spaces containing the ciphertext and plaintext.
Symmetric Key Decipher (CSNBSYD, CSNBSYD1, CSNESYD and CSNESYD1)

Format

CALL CSNBSYD(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier_length,
    key_identifier,
    key_parms_length,
    key_parms,
    block_size,
    initialization_vector_length,
    initialization_vector,
    chain_data_length,
    chain_data,
    cipher_text_length,
    cipher_text,
    clear_text_length,
    clear_text,
    optional_data_length,
    optional_data)

CALL CSNBSYD1(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier_length,
    key_identifier,
    key_parms_length,
    key_parms,
    block_size,
    initialization_vector_length,
    initialization_vector,
    chain_data_length,
    chain_data,
    cipher_text_length,
    cipher_text,
    clear_text_length,
    clear_text,
    optional_data_length,
    optional_data
    cipher_text_id,
    clear_text_id)

Parameters

return_code
Direction: Output
Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code
Direction: Output
Type: Integer
Symmetric Key Decipher (CSNBSYD, CSNBSYD1, CSNESYD and CSNESYD1)

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

exit_data_length
Direction: Ignored Type: Integer

Reserved field.

exit_data
Direction: Ignored Type: String

Reserved field.

rule_array_count
Direction: Input Type: Integer

The number of keywords you supplied in the rule_array parameter. The value may be 1, 2, 3 or 4.

rule_array
Direction: Input Type: String

An array of 8-byte keywords providing the processing control information. The keywords must be in contiguous storage, left-justified and padded on the right with blanks.

Table 87. Symmetric Key Decipher Rule Array Keywords

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithm (required)</strong></td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>Specifies that the Advanced Encryption Standard (AES) algorithm is to be used. The block size is 16 bytes. The key length may be 16, 24, or 32 bytes. The chain_data field must be at least 32 bytes in length. The OCV is the first 16 bytes in the chain_data. The supported processing rules for AES are CBC, ECB, CFB and PKCS-PAD.</td>
</tr>
<tr>
<td>DES</td>
<td>Specifies that the Data Encryption Standard (DES) algorithm is to be used. The algorithm, DES or TDES, will be determined from the length of the key supplied. The key length may be 8, 16, or 24. The block size is 8 bytes. The chain_data field must be at least 16 bytes in length. The OCV is the first eight bytes in the chain_data. The processing rules supported for DES are CBC, ECB, X9.23, CUSP and IPS, CFB and PKCS-PAD.</td>
</tr>
<tr>
<td><strong>Processing Rule (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>CBC</td>
<td>Performs cipher block chaining. The text length must be a multiple of the block size for the specified algorithm. CBC is the default value.</td>
</tr>
<tr>
<td>CFB</td>
<td>CFB mode (cipher feedback) that is compatible with IBM's Encryption Facility product. Input text may be any length.</td>
</tr>
</tbody>
</table>
Table 87. Symmetric Key Decipher Rule Array Keywords (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSP</td>
<td>CBC mode (cipher block chaining) that is compatible with IBM's CUSP and PCF products. Input text may be any length.</td>
</tr>
<tr>
<td>ECB</td>
<td>Performs electronic code book encryption. The text length must be a multiple of the block size for the specified algorithm.</td>
</tr>
<tr>
<td>IPS</td>
<td>CBC mode (cipher block chaining) that is compatible with IBM's IPS product. Input text may be any length.</td>
</tr>
<tr>
<td>PKCS-PAD</td>
<td>CBC mode (cipher block chaining) but the ciphertext must be an exact multiple of the block length (8 bytes for DES and 16 bytes for AES). The plaintext will be 1 to 8 bytes shorter for DES and 1 to 16 bytes shorter for AES than the ciphertext.</td>
</tr>
<tr>
<td>X9.23</td>
<td>CBC mode (cipher block chaining) for 1 to 8 bytes of padding dropped from the output clear text.</td>
</tr>
</tbody>
</table>

**Key Rule (optional)**

- **KEY-CLR**
  - This specifies that the key parameter contains a clear key value. KEY-CLR is the default value.

- **KEYIDENT**
  - This specifies that the `key_identifier` field will be an internal clear token, or the label name of a clear key or encrypted key in the CKDS. Normal CKDS labelname syntax is required. Valid with DES and AES.

**ICV Selection (optional)**

- **INITIAL**
  - This specifies taking the initialization vector from the `initialization_vector` parameter. INITIAL is the default value.

- **CONTINUE**
  - This specifies taking the initialization vector from the output chaining vector contained in the work area to which the `chain_data` parameter points. CONTINUE is valid for processing rules CBC, CFB, PKCS-PAD, IPS, and CUSP only.

---

**key_identifier_length**

Direction: Input  
Type: Integer

The length of the `key_identifier` parameter. For clear keys, the length is in bytes and includes only the value of the key. The maximum size is 256 bytes.

For the KEYIDENT keyword, this parameter value must be 64.

**key_identifier**

Direction: Input  
Type: String

For the KEY-CLR keyword, this specifies the cipher key. The parameter must be left justified.

For the KEYIDENT keyword, this specifies an internal clear token, or the label name of a clear key or an encrypted key in the CKDS. Normal CKDS labelname syntax is required. KEYIDENT is valid with DES and AES.

**key_parms_length**

Direction: Ignored  
Type: Integer
Symmetric Key Decipher (CSNBSYD, CSNBSYD1, CSNESYD and CSNESYD1)

The length of the key_parms parameter. The maximum size is 256 bytes.

key_parms
Direction: Ignored Type: String

This parameter contains key-related parameters specific to the encryption algorithm.

block_size
Direction: Input Type: Integer

This parameter contains the processing size of the text block in bytes. This value will be algorithm specific. Be sure to specify the same block size as used to encipher the text.

initialization_vector_length
Direction: Input Type: Integer

The length of the initialization_vector parameter. The length should be equal to the block length for the algorithm specified.

initialization_vector
Direction: Input Type: String

This initialization chaining value for CBC encryption. You must use the same ICV that was used to encipher the data.

chain_data_length
Direction: Input/Output Type: Integer

The length of the chain_data parameter. On output, the actual length of the chaining vector will be stored in the parameter.

chain_data
Direction: Input/Output Type: String

This field is used as a system work area for the chaining vector. Your application program must not change the data in this string. The chaining vector holds the output chaining vector from the caller.

The direction is output if the ICV selection keyword is INITIAL.

The mapping of the chain_data depends on the algorithm specified. For AES, the chain_data field must be at least 32 bytes in length. The OCV is in the first 16 bytes in the chain_data. For DES, chain_data field must be at least 16 bytes in length.

cipher_text_length
Direction: Input Type: Integer
Symmetric Key Decipher (CSNBSYD, CSNBSYD1, CSNESYD and CSNESYD1)

The length of the ciphertext. A zero value in the cipher_text_length parameter is not valid. The length must be a multiple of the algorithm block size.

Ciphertext may be any length with the CFB, CUSP, or IPS keywords.

cipher_text
Direction: Input Type: String

The text to be deciphered.

clear_text_length
Direction: Input/Output Type: Integer

On input, this parameter specifies the size of the storage pointed to by the clear_text parameter. On output, this parameter has the actual length of the text stored in the clear_text parameter.

Input text may be any length with the CUSP keyword.

clear_text
Direction: Output Type: String

The deciphered text the service returns.

optional_data_length
Direction: Ignored Type: Integer

The length of the optional_data parameter.

optional_data
Direction: Ignored Type: String

Optional data required by a specified algorithm.

cipher_text_id
Direction: Input Type: Integer

For CSNBSYD1 only, the ALET of the ciphertext to be deciphered.

clear_text_id
Direction: Input Type: Integer

For CSNBSYD1 only, the ALET of the clear text supplied by the application.

Usage Notes

- SAF may be invoked to verify the caller is authorized to use the specified key label stored in the CKDS.
- To use a CKDS encrypted key, the ICSF segment of the CSFKEYS class general resource profile associated with the specified key label must contain SYMCPACFWRAP(YES).
- No pre- or post-processing exits are enabled for this service.
Symmetric Key Decipher (CSNBSYD, CSNBSYD1, CSNESYD and CSNESYD1)

- The master keys need to be loaded only when using this service with encrypted key labels.
- The AES algorithm will use hardware if it is available. Otherwise, clear key operations will be performed in software.
- AES has the same availability restrictions as triple-DES.
- This service will fail if execution would cause destructive overlay of the cipher_text field.

Table 88. Symmetric Key Decipher required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>DES keyword is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>CP Assist for Cryptographic Functions</td>
<td>Encrypted keys require CEX3C with the November 2009 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

Related Information

You cannot overlap the plaintext and ciphertext fields. For example:

```
pppppp
    ccccc   is not supported.
```

```
cccccc
    ppppp   is not supported.
```

ppppppcccccc is supported.

P represents the plaintext and c represents the ciphertext.

"Cipher Processing Rules" on page 680 discusses the cipher processing rules.

Symmetric Key Encipher (CSNBSYE, CSNBSYE1, CSNESYE and CSNESYE1)

Use the symmetric key encipher callable service to encipher data using the cipher block chaining, electronic code book, or cipher feedback modes. ICSF supports these processing rules to encipher data. You choose the type of processing rule that the encipher callable service should use for the block chaining.

<table>
<thead>
<tr>
<th>Processing Rule</th>
<th>Purpose</th>
</tr>
</thead>
</table>

## Symmetric Key Encipher (CSNBSYE, CSNBSYE1, CSNESYE and CSNESYE1)

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI X9.23</td>
<td>For block chaining not necessarily in exact multiples of 8 bytes. This process rule pads the plaintext so that ciphertext produced is an exact multiple of 8 bytes.</td>
</tr>
<tr>
<td>CBC</td>
<td>For block chaining in exact multiples of 8 bytes.</td>
</tr>
<tr>
<td>CFB</td>
<td>Performs cipher feedback encryption. The plaintext can be of any length. The ciphertext will have the same length as the plaintext.</td>
</tr>
<tr>
<td>CUSP</td>
<td>For block chaining not necessarily in exact multiples of 8 bytes. The ciphertext will be the same length as the plaintext.</td>
</tr>
<tr>
<td>ECB</td>
<td>Performs electronic code book encryption. The text length must be a multiple of the block size for the specified algorithm.</td>
</tr>
<tr>
<td>IPS</td>
<td>For block chaining not necessarily in exact multiples of 8 bytes. The ciphertext will be the same length as the plaintext.</td>
</tr>
<tr>
<td>PKCS-PAD</td>
<td>For block chaining not necessarily in exact multiples of the block length (8 bytes for DES and 16 bytes for AES). The ciphertext will be longer than the plaintext.</td>
</tr>
</tbody>
</table>

The Advanced Encryption Standard (AES) and Data Encryption Standard (DES) are supported. AES encryption uses a 128-, 192-, or 256-bit key. The CBC, CFB, and ECB modes are supported.

This service supports electronic code book (ECB), cipher block chaining (CBC), and cipher feedback (CFB) modes. The CBC and CFB modes of operation use an initial chaining vector (ICV) in their processing. During CBC mode processing, the ICV is exclusive ORed with the first block of plaintext prior to the encryption step, and thereafter, the block of ciphertext just produced is exclusive ORed with the next block of plaintext, and so on. This disguises any pattern that may exist in the plaintext. For CFB mode processing, the ICV is first encrypted, then exclusive ORed with the first block of ciphertext, and thereafter, the block of exclusive ORed data is encrypted then exclusive ORed with the next block of ciphertext, and so on.

Both Cipher block chaining and Cipher Feedback mode also produce a resulting chaining value called the output chaining vector (OCV). The application can pass the OCV as the ICV in the next encipher call. This results in record chaining.

The electronic code book mode does not use the initial chaining vector.

The selection between single-DES decryption mode and triple-DES decryption mode is controlled by the length of the key supplied in the key_identifier parameter. If a single-length key is supplied, single-DES decryption is performed. If a double-length or triple-length key is supplied, triple-DES decryption is performed.

The key may be specified as a clear key value, an internal clear key token, or the label name of a clear key or an encrypted key in the CKDS.

The callable service Symmetric Key Decipher (without ALET) supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSNESYE.
Choosing between CSNBSYE and CSNBSYE1

CSNBSYE and CSNBSYE1 provide identical functions. When choosing which service to use, consider this:

- **CSNBSYE** requires the cleartext and ciphertext to reside in the caller’s primary address space. Also, a program using CSNBSYE adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.

- **CSNBSYE1** allows the cleartext and ciphertext to reside either in the caller’s primary address space or in a data space. This can allow you to encipher more data with one call. However, a program using CSNBSYE1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

For CSNBSYE1, `clear_text_id` and `cipher_text_id` are access list entry token (ALET) parameters of the data spaces containing the cleartext and ciphertext.

**Format**

```c
CALL CSNBSYE(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier_length,
    key_identifier,
    key_parms_length,
    key_parms,
    block_size,
    initialization_vector_length,
    initialization_vector,
    chain_data_length,
    chain_data,
    clear_text_length,
    clear_text,
    cipher_text_length,
    cipher_text,
    optional_data_length,
    optional_data)
```
CALL CSNBSYE1(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  rule_array_count,
  rule_array,
  key_identifier_length,
  key_identifier,
  key_parms_length,
  key_parms,
  block_size,
  initialization_vector_length,
  initialization_vector,
  chain_data_length,
  chain_data,
  clear_text_length,
  clear_text,
  cipher_text_length,
  cipher_text,
  optional_data_length,
  optional_data,
  clear_text_id,
  cipher_text_id)

Parameters

return_code
Direction: Output Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code
Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

exit_data_length
Direction: Ignored Type: Integer

Reserved field.

exit_data
Direction: Ignored Type: String

Reserved field.

rule_array_count
Direction: Input Type: Integer
Symmetric Key Encipher (CSNBSYE, CSNBSYE1, CSNESYE and CSNESYE1)

The number of keywords you supplied in the `rule_array` parameter. The value may be 1, 2, 3 or 4.

**rule_array**

**Direction:** Input  
**Type:** String

An array of 8-byte keywords providing the processing control information. The keywords must be in contiguous storage, left-justified and padded on the right with blanks.

**Table 89. Symmetric Key Encipher Rule Array Keywords**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithm (required)</strong></td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>Specifies that the Advanced Encryption Standard (AES) algorithm is to be used. On systems that contain a Cryptographic Coprocessor Feature, AES is the only algorithm that is supported. The block size is 16 bytes. The key length may be 16, 24, or 32 bytes. The <code>chain_data</code> field must be at least 32 bytes in length. The OCV is the first 16 bytes in the <code>chain_data</code>. The supported processing rules for AES are CBC, ECB, CFB and PKCS-PAD.</td>
</tr>
<tr>
<td>DES</td>
<td>Specifies that the Data Encryption Standard (DES) algorithm is to be used. The algorithm, DES or TDES, will be determined from the length of the key supplied. The key length may be 8, 16, or 24. The block size is 8 bytes. The <code>chain_data</code> field must be at least 16 bytes in length. The OCV is the first eight bytes in the <code>chain_data</code>. The processing rules supported for DES are CBC, ECB, X9.23, CUSP, IPS, CFB and PKCS-PAD.</td>
</tr>
<tr>
<td><strong>Processing Rule (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>CBC</td>
<td>Performs cipher block chaining. The text length must be a multiple of the block size for the specified algorithm. CBC is the default value.</td>
</tr>
<tr>
<td>CFB</td>
<td>CFB mode (cipher feedback) that is compatible with IBM's Encryption Facility product. Input text may be any length.</td>
</tr>
<tr>
<td>CUSP</td>
<td>CBC mode (cipher block chaining) that is compatible with IBM's CUSP and PCF products. Input text may be any length.</td>
</tr>
<tr>
<td>ECB</td>
<td>Performs electronic code book encryption. The text length must be a multiple of the block size for the specified algorithm.</td>
</tr>
<tr>
<td>IPS</td>
<td>CBC mode (cipher block chaining) that is compatible with IBM's IPS product. Input text may be any length.</td>
</tr>
<tr>
<td>PKCS-PAD</td>
<td>CBC mode (cipher block chaining) not necessarily in exact multiples of the block length (8 bytes for DES and 16 bytes for AES). PKCS-PAD always pads the plaintext so that the ciphertext produced is an exact multiple of the block length and longer than the plaintext.</td>
</tr>
<tr>
<td>X9.23</td>
<td>CBC mode (cipher block chaining) for 1 to 8 bytes of padding added according to ANSI X9.23. Input text may be any length.</td>
</tr>
<tr>
<td><strong>Key Rule (optional)</strong></td>
<td></td>
</tr>
</tbody>
</table>
Symmetric Key Encipher (CSNBSYE, CSNBSYE1, CSNESYE and CSNESYE1)

Table 89. Symmetric Key Encipher Rule Array Keywords (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY-CLR</td>
<td>This specifies that the key parameter contains a clear key value. KEY-CLR is the default.</td>
</tr>
<tr>
<td>KEYIDENT</td>
<td>This specifies that the key_identifier field will be an internal clear token, or the label name of a clear key or an encrypted key in the CKDS. Valid with DES and AES.</td>
</tr>
</tbody>
</table>

ICV Selection (optional)

| INITIAL | This specifies taking the initialization vector from the initialization_vector parameter. INITIAL is the default value. |
| CONTINUE | This specifies taking the initialization vector from the output chaining vector contained in the work area to which the chain_data parameter points. CONTINUE is valid for processing rules CBC, CFB, PKCS-PAD, IPS, and CUSP only. |

key_identifier_length

Direction: Input  Type: Integer

The length of the key_identifier parameter. For clear keys, the length is in bytes and includes only the value of the key.

For the KEYIDENT keyword, this parameter value must be 64.

key_identifier

Direction: Input  Type: String

For the KEY-CLR keyword, this specifies the cipher key. The parameter must be left justified.

For the KEYIDENT keyword, this specifies a internal clear token, or the label name of a clear key or an encrypted key in the CKDS. Normal CKDS label name syntax is required.

key_parms_length

Direction: Ignored  Type: Integer

The length of the key_parms parameter.

key_parms

Direction: Ignored  Type: String

This parameter contains key-related parameters specific to the encryption algorithm.

block_size

Direction: Input  Type: Integer

This parameter contains the processing size of the text block in bytes. This value will be algorithm specific.
Symmetric Key Encipher (CSNBSYE, CSNBSYE1, CSNESYE and CSNESYE1)

**initialization_vector_length**
Direction: Input  
Type: Integer

The length of the initialization_vector parameter. The length should be equal to the block length for the algorithm specified.

**initialization_vector**
Direction: Input  
Type: String

This initialization chaining value for CBC encryption. You must use the same ICV to decipher the data.

**chain_data_length**
Direction: Input/Output  
Type: Integer

The length of the chain_data parameter. On output, the actual length of the chaining vector will be stored in the parameter.

**chain_data**
Direction: Input/Output  
Type: String

This field is used as a system work area for the chaining vector. Your application program must not change the data in this string. The chaining vector holds the output chaining vector from the caller.

The direction is output if the ICV selection keyword is INITIAL.

The mapping of the chain_data depends on the algorithm specified. For AES, the chain_data field must be at least 32 bytes in length. The OCV is in the first 16 bytes in the chain_data. For DES, the chain_data field must be at least 16 bytes in length.

**clear_text_length**
Direction: Input  
Type: Integer

The length of the clear text. A zero value in the clear_text_length parameter is not valid. The length must be a multiple of the algorithm block size.

Input text may be any length with the CFB, CUSP, or IPS keywords.

**clear_text**
Direction: Input  
Type: String

The text to be enciphered.

**cipher_text_length**
Direction: Input/Output  
Type: Integer

On input, this parameter specifies the size of the storage pointed to by the cipher_text parameter. On output, this parameter has the actual length of the text stored in the buffer addressed by the cipher_text parameter.
Symmetric Key Encipher (CSNBSYE, CSNBSYE1, CSNESYE and CSNESYE1)

cipher_text
Direction: Output Type: String

The enciphered text the service returns.

optional_data_length
Direction: Ignored Type: Integer

The length of the optional_data parameter.

optional_data
Direction: Ignored Type: String

Optional data required by a specified algorithm.

clear_text_id
Direction: Input Type: Integer

For CSNBSYE1 only, the ALET of the clear text to be enciphered.

cipher_text_id
Direction: Input Type: Integer

For CSNBSYE1 only, the ALET of the ciphertext that the application supplied.

Usage Notes

- SAF may be invoked to verify the caller is authorized to use the specified key label stored in the CKDS.
- To use a CKDS encrypted key, the ICSF segment of the CSFKEYS class general resource profile associated with the specified key label must contain SYMCPACFWRAP(YES).
- No pre- or post-processing exits are enabled for this service.
- The master keys need to be loaded only when using this service with the encrypted key labels.
- The AES algorithm will use hardware if it is available. Otherwise, clear key operations will be performed in software.
- AES has the same availability restrictions as triple-DES.
- This service will fail if execution would cause destructive overlay of the clear_text field.

Table 90. Symmetric Key Encipher required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td></td>
<td>DES keyword is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Symmetric Key Encipher (CSNBSYE, CSNBSYE1, CSNESYE and CSNESYE1)

Table 90. Symmetric Key Encipher required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 990</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td>Encrypted keys require the CEX3C with the November 2009 or later licensed internal code (LIC).</td>
</tr>
</tbody>
</table>

### Related Information

You **cannot** overlap the plaintext and ciphertext fields. For example:

```
pppppp
   cccccc  is not supported.

cccccc
    pppppp  is not supported.

ppppppcccccc is supported.
```

P represents the plaintext and c represents the ciphertext.

The method used to produce the OCV is the same with the CBC and X9.23 processing rules. However, that method is different from the method used by the CUSP and IPS processing rules.

"Cipher Processing Rules" on page 680 discusses the cipher processing rules.
Chapter 7. Verifying Data Integrity and Authenticating Messages

ICSF provides several methods to verify the integrity of transmitted messages and stored data:
- Message authentication code (MAC)
- Hash functions, including modification detection code (MDC) processing and one-way hash generation

**Note:** You can also use digital signatures (see Chapter 9, “Using Digital Signatures,” on page 379) to authenticate messages.

The choice of callable service depends on the security requirements of the environment in which you are operating. If you need to ensure the authenticity of the sender as well as the integrity of the data, and both the sender and receiver can share a secret key, consider message authentication code processing. If you need to ensure the integrity of transmitted data in an environment where it is not possible for the sender and the receiver to share a secret cryptographic key, consider hashing functions, such as the modification detection code process.

The callable services are described in the following topics:
- “MAC Generate (CSNBGMGN and CSNBGMGN1)” on page 271
- “MAC Verify (CSNBGMVR and CSNBGMVR1)” on page 276
- “MDC Generate (CSNBMDG and CSNBMDG1)” on page 281
- “One-Way Hash Generate (CSNBOWH, CSNEOWH and CSNBOWH1)” on page 286
- “Symmetric MAC Generate (CSNBMSG, CSNBMSG1, CSNESMG, and CSNESMG1)” on page 290
- “Symmetric MAC Verify (CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1)” on page 294

How MACs are Used

When a message is sent, an application program can generate an authentication code for it using the MAC generation callable service. ICSF supports the ANSI X9.9-1 basic procedure and both the ANSI X9.19 basic procedure and optional double key MAC procedure. The service computes the text of the message authentication code using the algorithm and a key. The ANSI X9.9-1 or ANSI X9.19 basic procedures accept either a single-length MAC generation (MAC) key or a data-encrypting (DATA) key, and the message text. The ANSI X9.19 optional double key MAC procedure accepts a double-length MAC key and the message text. The message text may be in clear or encrypted form. The originator of the message sends the MAC with the message text.

When the receiver gets the message, an application program calls the **MAC verification callable service**. The callable service generates a MAC using the same algorithm as the sender and either the single-length or double-length MAC verification key, the single-length or double-length MAC generation key, or DATA key, and the message text. The MACVER callable service compares the MAC it generates with the one sent with the message and issues a return code that indicates whether the MACs match. If the return code indicates that the MACs match, the receiver can accept the message as genuine and unaltered. If the return code indicates that the MACs do not match, the receiver can assume that the
message is either bogus or has been altered. The newly computed MAC is not revealed outside the cryptographic feature.

In a similar manner, MACs can be used to ensure the integrity of data stored on the system or on removable media, such as tape.

Secure use of the MAC generation and MAC verification services requires the use of MAC and MACVER keys in these services, respectively. To accomplish this, the originator of the message generates a MAC/MACVER key pair, uses the MAC key in the MAC generation service, and exports the MACVER key to the receiver. The originator of the message enforces key separation on the link by encrypting the MACVER key under a transport key that is not an NOCV key before exporting the key to the receiver. With this type of key separation enforced, the receiver can only receive a MACVER key and can use only this key in the MAC verification service. This ensures that the receiver cannot alter the message and produce a valid MAC with the altered message. These security features are not present if DATA keys are used in the MAC generation service, or if DATA or MAC keys are used in the MAC verification service.

By using MACs, you get the following benefits:

• **For data transmitted over a network**, you can validate the authenticity of the message as well as ensure that the data has not been altered during transmission. For example, an active eavesdropper can tap into a transmission line, and interject bogus messages or alter sensitive data being transmitted. If the data is accompanied by a MAC, the recipient can use a callable service to detect whether the data has been altered. Since both the sender and receiver share a secret key, the receiver can use a callable service that calculates a MAC on the received message and compares it to the MAC transmitted with the message. If the comparison is equal, the message may be accepted as unaltered. Furthermore, since the shared key is secret, when a MAC is verified it can be assumed that the sender was, in fact, the other person who knew the secret key.

• **For data stored on tape or DASD**, you can ensure that the data read back onto the system was the same as the data written onto the tape or DASD. For example, someone might be able to bypass access controls. Such an access might escape the notice of auditors. However, if a MAC is stored with the data, and verified when the data is read, you can detect alterations to the data.

---

**How Hashing Functions Are Used**

Hashing functions include the MDC and one-way hash. You need to hash text before submitting it to digital signature services (see Chapter 9, “Using Digital Signatures,” on page 379).

**How MDCs Are Used**

When a message is sent, an application program can generate a modification detection code for it using the MDC generation callable service. The service computes the modification detection code, a 128-bit value, using a one-way cryptographic function and the message text (which itself may be in clear or encrypted form). The originator of the message ensures that the MDC is transmitted with integrity to the intended receiver of the message. For example, the MDC could be published in a reliable source of public information.

When the receiver gets the message, an application program calls the MDC callable service. The callable service generates an MDC by using the same one-way cryptographic function and the message text. The application program can
compare the new MDC with the one generated by the originator of the message. If
the MDCs match, the receiver knows that the message was not altered.

In a similar manner, MDCs can be used to ensure the integrity of data stored on the
system or on removable media, such as tape.

By using MDCs, you get the following benefits:

- **For data transmitted over a network between locations that do not share a secret key,** you can ensure that the data has not been altered during transmission. It is easy to compute an MDC for specific data, yet hard to find data that will result in a given MDC. In effect, the problem of ensuring the integrity of a large file is reduced to ensuring the integrity of a 128-bit value.

- **For data stored on tape or DASD,** you can ensure that the data read back onto the system was the same as the data written onto the tape or DASD. Once an MDC has been established for a file, the MDC generation callable service can be run at any later time on the file. The resulting MDC can be compared with the stored MDC to detect deliberate or inadvertent modification.

SHA-1 is a FIPS standard required for DSS. MD5 is a hashing algorithm used to derive Message Digests in Digital Signature applications.

---

**MAC Generate (CSNBMGN and CSNBMGN1)**

Use the MAC generate callable service to generate a 4-, 6-, or 8-byte message authentication code (MAC) for an application-supplied text string. You can specify that the callable service uses either the ANSI X9.9-1 procedure or the ANSI X9.19 optional double key MAC procedure to compute the MAC. For the ANSI X9.9-1 procedure you identify either a MAC generate key or a DATA key, and the message text. For the ANSI X9.19 optional double key MAC procedure, you identify a double-length MAC key and the message text.

The MAC generate callable service also supports the padding rules specified in the EMV Specification and ISO 16609. For the EMV MAC procedure, you identify a single- or double-length MAC key and the message text. For the ISO 16609 procedure you identify a double-length MAC or DATA key and the message text.

**Choosing Between CSNBMGN and CSNBMGN1**

CSNBMGN and CSNBMGN1 provide identical functions. When choosing which service to use, consider the following:

- **CSNBMGN** requires the application-supplied text to reside in the caller’s primary address space. Also, a program using CSNBMGN adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.

- **CSNBMGN1** allows the application-supplied text to reside either in the caller’s primary address space or in a data space. This can allow you to process more data with one call. However, a program using CSNBMGN1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified before it can run with other cryptographic products that follow this programming interface.

For CSNBMGN1, **text_id_in** is an access list entry token (ALET) parameter of the data space containing the application-supplied text.
MAC Generate (CSNBMGN and CSNBMGN1)

Format

```call
CALL CSNBMGN(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  key_identifier,
  text_length,
  text,
  rule_array_count,
  rule_array,
  chaining_vector,
  mac )
```

```call
CALL CSNBMGN1(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  key_identifier,
  text_length,
  text,
  rule_array_count,
  rule_array,
  chaining_vector,
  mac,
  text_id_in )
```

Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. [Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.](#)

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. [Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.](#)

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the **exit_data** parameter.

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation exit.
MAC Generate (CSNBMGN and CSNBMGN1)

key_identifier
Direction: Input/Output 
Type: String

The 64-byte key label or internal key token that identifies a single or double-length MAC generate key, a DATAM key, or a single-length DATA key. The type of key depends on the MAC process rule in the rule_array parameter.

text_length
Direction: Input 
Type: Integer

The length of the text you supply in the text parameter. The maximum length of text is 214783647 bytes. If the text_length is not a multiple of 8 bytes and if the ONLY or LAST keyword of the rule_array parameter is called, the text is padded in accordance with the processing rule specified.

Note: Beginning in z/OS V1 R2, the MAXLEN value may still be specified in the options data set, but only the maximum value limit will be enforced.

text
Direction: Input 
Type: String

The application-supplied text for which the MAC is generated.

rule_array_count
Direction: Input 
Type: Integer

The number of keywords specified in the rule_array parameter. The value can be 0, 1, 2, or 3.

rule_array
Direction: Input 
Type: Character string

Zero to three keywords that provide control information to the callable service. The keywords are shown in Table 91. The keywords must be in 24 bytes of contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks. For example,

'X9.9-1 MIDDLE MACLEN4 '

The order of the rule_array keywords is not fixed.

You can specify one of the MAC processing rules and then choose one of the segmenting control keywords and one of the MAC length keywords.

Table 91. Keywords for MAC generate Control Information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC Process Rules (optional)</td>
<td></td>
</tr>
<tr>
<td>EMVMAC</td>
<td>EMV padding rule with a single-length MAC key. The key_identifier parameter must identify a single-length MAC or a single-length DATA key. The text is always padded with 1 to 8 bytes so that the resulting text length is a multiple of 8 bytes. The first pad character is X'80'. The remaining 0 to 7 pad characters are X'00'.</td>
</tr>
</tbody>
</table>
### Table 91. Keywords for MAC generate Control Information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMVMACD</td>
<td>EMV padding rule with a double-length MAC key. The \texttt{key_identifier} parameter must identify a double-length MAC key. The padding rules are the same as for EMVMAC.</td>
</tr>
<tr>
<td>X9.19OPT</td>
<td>ANSI X9.19 optional double key MAC procedure. The \texttt{key_identifier} parameter must identify a double-length MAC key. The padding rules are the same as for X9.9-1.</td>
</tr>
<tr>
<td>X9.9-1</td>
<td>ANSI X9.9-1 and X9.19 basic procedure. The \texttt{key_identifier} parameter must identify a single-length MAC or a single-length DATA key. X9.9-1 causes the MAC to be computed from all of the data. The text is padded only if the text length is not a multiple of 8 bytes. If padding is required, the pad character X'00' is used. This is the default value.</td>
</tr>
<tr>
<td>TDES-MAC</td>
<td>ISO 16609 procedure. The \texttt{key_identifier} must identify a double-length MAC or a double-length DATA key. The text is padded only if the text length is not a multiple of 8 bytes.</td>
</tr>
</tbody>
</table>

#### Segmenting Control (optional)

<table>
<thead>
<tr>
<th>First Call</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIRST</td>
<td>First call, this is the first segment of data from the application program.</td>
</tr>
<tr>
<td>LAST</td>
<td>Last call; this is the last data segment.</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>Middle call; this is an intermediate data segment.</td>
</tr>
<tr>
<td>ONLY</td>
<td>Only call; segmenting is not employed by the application program. This is the default value.</td>
</tr>
</tbody>
</table>

#### MAC Length and Presentation (optional)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEX-8</td>
<td>Generates a 4-byte MAC value and presents it as 8 hexadecimal characters.</td>
</tr>
<tr>
<td>HEX-9</td>
<td>Generates a 4-byte MAC value and presents it as 2 groups of 4 hexadecimal characters with a space between the groups.</td>
</tr>
<tr>
<td>MACLEN4</td>
<td>Generates a 4-byte MAC value. This is the default value.</td>
</tr>
<tr>
<td>MACLEN6</td>
<td>Generates a 6-byte MAC value.</td>
</tr>
<tr>
<td>MACLEN8</td>
<td>Generates an 8-byte MAC value.</td>
</tr>
</tbody>
</table>

#### chaining_vector

<table>
<thead>
<tr>
<th>Direction: Input/Output</th>
<th>Type: String</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An 18-byte string that ICSF uses as a system work area. Your application program must not change the data in this string. The chaining vector permits data to be chained from one invocation call to another.</td>
</tr>
<tr>
<td></td>
<td>On the first call, initialize this parameter as binary zeros.</td>
</tr>
</tbody>
</table>

#### mac

<table>
<thead>
<tr>
<th>Direction: Output</th>
<th>Type: String</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The 8-byte or 9-byte field in which the callable service returns the MAC value if the segmenting rule is ONLY or LAST. Allocate an 8-byte field for MAC values of 4 bytes, 6 bytes, 8 bytes, or HEX-8. Allocate a 9-byte MAC field if you specify HEX-9 in the rule_array parameter.</td>
</tr>
</tbody>
</table>
MAC Generate (CSNBMGN and CSNBMGN1)

text_id_in

Direction: Input  Type: Integer

For CSNBMGN1 only, the ALET of the text for which the MAC is generated.

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

CCF Systems: To use a DATA key, the NOCV-enablement keys must be present in the CKDS. Using a DATA key instead of a MAC generate key in this service substantially increases the path length for generating the MAC.

To calculate a MAC in one call, specify the ONLY keyword for segmenting control for the rule_array parameter. For two or more calls, specify the FIRST keyword for the first input block, the MIDDLE keyword for intermediate blocks (if any), and the LAST keyword for the last block.

For a given text string, the resulting MAC is the same whether the text is segmented or not.

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 92. MAC generate required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
</table>
| IBM @server zSeries 800 | Cryptographic Coprocessor Feature | ICSF routes the request to a PCI Cryptographic Coprocessor if the control vector in the supplied key identifier cannot be processed on the Cryptographic Coprocessor Feature. If no PCI Cryptographic Coprocessor is online in this case, the request fails. The request must meet the following restrictions:  
  • The MAC Process Rule is X9.19OPT or EMVMACD.  
  • The MAC key is a valid double-length MAC generate key.  
  • The text_length must be less than or equal to 4K bytes for the FIRST and MIDDLE keywords, and the text length must be a multiple of 8 bytes.  
  • The text_length on the final call (ONLY or LAST) can not be greater than 4K including padding.  
  TDES-MAC not supported. |
| IBM @server zSeries 900 | PCI X Cryptographic Coprocessor | TDES-MAC not supported. |
| IBM @server zSeries 990 | Crypto Express2 Coprocessor | TDES-MAC not supported. |
| IBM @server zSeries 890 | | |
Table 92. MAC generate required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

Related Information
For more information about MAC processing rules and segmenting control, refer to IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface Reference.

The MAC verification callable service is described in MAC Verify (CSNBMVR and CSNBMVR1).

MAC Verify (CSNBMVR and CSNBMVR1)
Use the MAC verify callable service to verify a 4-, 6-, or 8-byte message authentication code (MAC) for an application-supplied text string. You can specify that the callable service uses either the ANSI X9.9-1 procedure or the ANSI X9.19 optional double key MAC procedure to compute the MAC. For the ANSI X9.9-1 procedure you identify either a MAC verify key, a MAC generation key, or a DATA key, and the message text. For the ANSI X9.19 optional double key MAC procedure, you identify either a double-length MAC verify key or a double-length MAC generation key and the message text. The cryptographic feature compares the generated MAC with the one sent with the message. A return code indicates whether the MACs are the same. If the MACs are the same, the receiver knows the message was not altered. The generated MAC never appears in storage is not revealed outside the cryptographic feature.

The MAC verify callable service also supports the padding rules specified in the EMV Specification and ISO 16609. For the EMV MAC procedure, you identify a single- or double-length MAC key and the message text. For the ISO 16609 procedure you identify a double-length MAC or DATA key and the message text.

Choosing Between CSNBMVR and CSNBMVR1
CSNBMVR and CSNBMVR1 provide identical functions. When choosing which service to use, consider the following:

- **CSNBMVR** requires the application-supplied text to reside in the caller's primary address space. Also, a program using CSNBMVR adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.
- **CSNBMVR1** allows the application-supplied text to reside either in the caller's primary address space or in a data space. This can allow you to verify more data with one call. However, a program using CSNBMVR1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified before it can run with other cryptographic products that follow this programming interface.
MAC Verify (CSNBMVR and CSNBMVR1)

For CSNBMVR1, \text{text\_id\_in} is an access list entry token (ALET) parameter of the data space containing the application-supplied text.

Format

\begin{verbatim}
CALL CSNBMVR(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier,
    text_length,
    text,
    rule_array_count,
    rule_array,
    chaining_vector,
    mac )
\end{verbatim}

\begin{verbatim}
CALL CSNBMVR1(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier,
    text_length,
    text,
    rule_array_count,
    rule_array,
    chaining_vector,
    mac,
    text\_id\_in )
\end{verbatim}

Parameters

\text{return\_code}

Direction: Output  
Type: Integer

The \text{return\_code} specifies the general result of the callable service. \textbf{Appendix A, “ICSF and TSS Return and Reason Codes”} lists the return codes.

\text{reason\_code}

Direction: Output  
Type: Integer

The \text{reason\_code} specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. \textbf{Appendix A, “ICSF and TSS Return and Reason Codes”} lists the reason codes.

\text{exit\_data\_length}

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the installation exit. The length can be from \text{X'00000000'} to \text{X'7FFFFFFF'} (2 gigabytes). The data is identified in the \text{exit\_data} parameter.
MAC Verify (CSNBMVVR and CSNBMVVR1)

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**key_identifier**

Direction: Input/Output  
Type: String

The 64-byte key label or internal key token that identifies a single or double-length MAC verify key, a single or double-length MAC verify key, a single or double-length MAC generation key, a DATAM or DATAMV key, or a single-length DATA key. The type of key depends on the MAC process rule in the rule_array parameter.

**text_length**

Direction: Input  
Type: Integer

The length of the text you supply in the text parameter. The maximum length of text is 214783647 bytes. If the text_length parameter is not a multiple of 8 bytes and if the ONLY or LAST keyword of the rule_array parameter is called, the text is padded in accordance with the processing rule specified.

**Note:** Beginning in z/OS V1 R2, the MAXLEN value may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647).

**text**

Direction: Input  
Type: String

The application-supplied text for which the MAC is generated.

**rule_array_count**

Direction: Input  
Type: Integer

The number of keywords specified in the rule_array parameter. The value can be 0, 1, 2, or 3.

**rule_array**

Direction: Input  
Type: String

Zero to three keywords that provide control information to the callable service. The keywords are shown in Table 93 on page 279. The keywords must be in 24 bytes of contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks. For example:

'X9.9-1  MIDDLE  MACLEN4  '

The order of the rule_array keywords is not fixed.

You can specify one of the MAC processing rules and then choose one of the segmenting control keywords and one of the MAC length keywords.
Table 93. Keywords for MAC verify Control Information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAC Process Rules (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>EMVMAC</td>
<td>EMV padding rule with a single-length MAC key. The key_identifier parameter must identify a single-length MAC, MACVER, or DATA key. The text is always padded with 1 to 8 bytes so that the resulting text length is a multiple of 8 bytes. The first pad character is X'80'. The remaining 0 to 7 pad characters are X'00'.</td>
</tr>
<tr>
<td>EMVMACD</td>
<td>EMV padding rule with a double-length MAC key. The key_identifier parameter must identify a double-length MAC or MACVER key. The padding rules are the same as for EMVMAC.</td>
</tr>
<tr>
<td>X9.19OPT</td>
<td>ANSI X9.9-1 and X9.19 basic procedure. The key_identifier parameter must identify a single-length MAC, MACVER, or DATA key. X9.9-1 causes the MAC to be computed from all of the data. The text is padded only if the text length is not a multiple of 8 bytes. If padding is required, the pad character X'00' is used. This is the default value.</td>
</tr>
<tr>
<td>X9.9-1</td>
<td>ANSI X9.9-1 and X9.19 basic procedure. The key_identifier parameter must identify a single-length MAC, or single-length DATA key. X9.9-1 causes the MAC to be computed from all of the data. The text is padded only if the text length is not a multiple of 8 bytes. If padding is required, the pad character X'00' is used. This is the default value.</td>
</tr>
<tr>
<td>TDES-MAC</td>
<td>ISO 16609 procedure. The key_identifier must identify a double-length MAC or a double-length DATA key. The text is padded only if the text length is not a multiple of 8 bytes.</td>
</tr>
<tr>
<td><strong>Segmenting Control (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>FIRST</td>
<td>First call; this is the first segment of data from the application program.</td>
</tr>
<tr>
<td>LAST</td>
<td>Last call; this is the last data segment.</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>Middle call; this is an intermediate data segment.</td>
</tr>
<tr>
<td>ONLY</td>
<td>Only call; the application program does not employ segmenting. This is the default value.</td>
</tr>
<tr>
<td><strong>MAC Length and Presentation (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>HEX-8</td>
<td>Verifies a 4-byte MAC value that is represented as 8 hexadecimal characters.</td>
</tr>
<tr>
<td>HEX-9</td>
<td>Verifies a 4-byte MAC value that is represented as 2 groups of 4 hexadecimal characters with a space character between the groups.</td>
</tr>
<tr>
<td>MACLEN4</td>
<td>Verifies a 4-byte MAC value. This is the default value.</td>
</tr>
<tr>
<td>MACLEN6</td>
<td>Verifies a 6-byte MAC value.</td>
</tr>
<tr>
<td>MACLEN8</td>
<td>Verifies an 8-byte MAC value.</td>
</tr>
</tbody>
</table>

**chaining_vector**

Direction: Input/Output   Type: String
MAC Verify (CSNBMV R and CSNBMV R1)

An 18-byte string that ICSF uses as a system work area. Your application program must not change the data in this string. The chaining vector permits data to be chained from one invocation call to another.

On the first call, initialize this parameter to binary zeros.

mac
Direction: Output  Type: String

The 8- or 9-byte field that contains the MAC value you want to verify. The value in the field must be left-justified and padded with zeros. If you specified the X'09' keyword in the rule_array parameter, the input MAC is 9 bytes.

text_id_in
Direction: Input  Type: Integer

For CSNBMV R1 only, the ALET of the text for which the MAC is to be verified.

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

To verify a MAC in one call, specify the ONLY keyword on the segmenting rule keyword for the rule_array parameter. For two or more calls, specify the FIRST keyword for the first input block, MIDDLE for intermediate blocks (if any), and LAST for the last block.

For a given text string, the MAC resulting from the verification process is the same regardless of how the text is segmented, or how it was segmented when the original MAC was generated.

CCF Systems only: To use a MAC generation key or a DATA key, the NOCV enablement keys must be present in the CKDS. Using either a MAC generation key or a DATA key instead of a MAC verify key in this service substantially increases the path length for verifying the MAC.

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
Table 94. MAC verify required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
</table>
| IBM @server zSeries 800 | Cryptographic Coprocessor Feature | ICSF routes the request to a PCI Cryptographic Coprocessor if the control vector in the supplied key identifier cannot be processed on the Cryptographic Coprocessor Feature. The request must meet the following restrictions:  
  • The MAC Process Rule is X9.19OPT or EMVMACD.  
  • The MAC key is a valid double-length MAC generate key.  
  • The text_length on the final call (ONLY or LAST) can not be greater than 4K including padding.  
  • The text_length must be less than or equal to 4K bytes for the FIRST and MIDDLE keywords, and the text length must be a multiple of 8 bytes.  
  TDES-MAC not supported. |
| IBM @server zSeries 900 | PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor | TDES-MAC not supported. |
| IBM System z9 EC      | Crypto Express2 Coprocessor     |                                                                             |
| IBM System z9 BC      | Crypto Express2 Coprocessor     |                                                                             |
| IBM System z10 EC     | Crypto Express2 Coprocessor     |                                                                             |
| IBM System z10 BC     | Crypto Express2 Coprocessor     |                                                                             |
| IBM System z10 BC     | Crypto Express3 Coprocessor     |                                                                             |

Related Information

For more information about MAC processing rules and segmenting control, refer to IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface Reference.

The MAC generation callable service is described in “MAC Generate (CSNBMGN and CSNBMGN1)” on page 271.

MDC Generate (CSNBMDG and CSNBMDG1)

A modification detection code (MDC) can be used to provide a form of support for data integrity.

Use the MDC generate callable service to generate a 128-bit modification detection code (MDC) for an application-supplied text string.

The returned MDC value should be securely stored and/or sent to another user. To validate the integrity of the text string at a later time, the MDC generate callable
MDC Generate (CSNBMDG and CSNBMDG1)

Service is again used to generate a 128-bit MDC. The new MDC value is compared with the original MDC value. If the values are equal, the text is accepted as unchanged.

Choosing Between CSNBMDG and CSNBMDG1

CSNBMDG and CSNBMDG1 provide identical functions. When choosing which service to use, consider the following:

- **CSNBMDG** requires the application-supplied text to reside in the caller’s primary address space. Also, a program using CSNBMDG adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.

- **CSNBMDG1** allows the application-supplied text to reside either in the caller’s primary address space or in a data space. This can allow you to process more data with one call. However, a program using CSNBMDG1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface and may need to be modified before it can run with other cryptographic products that follow this programming interface.

For CSNBMDG1, **text_id_in** parameter specifies the access list entry token (ALET) for the data space containing the application-supplied text.

Format

```call csnbmdg(  return_code,  reason_code,  exit_data_length,  exit_data,  text_length,  text,  rule_array_count,  rule_array,  chaining_vector,  mdc )```

```call csnbmdg1(  return_code,  reason_code,  exit_data_length,  exit_data,  text_length,  text,  rule_array_count,  rule_array,  chaining_vector,  mdc,  text_id_in )```

Parameters

**return_code**

Direction: Output  
Type: Integer

The **return_code** specifies the general result of the callable service. [Appendix A, "ICSF and TSS Return and Reason Codes," on page 557](#) lists the return codes.
reason_code
Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes," on page 557 lists the reason codes.

exit_data_length
Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

exit_data
Direction: Input/Output Type: String

The data that is passed to the installation exit.

text_length
Direction: Input Type: Integer

The length of the text you supply in the text parameter. The maximum length of text is 214783647 bytes.

Note: Beginning in z/OS V1 R2, the MAXLEN value may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647).

Additional restrictions on length of the text depend on whether padding of the text is requested, and on the segmenting control used.

- When padding is requested (by specifying a process rule of PADMDC-2 or PADMDC-4 in the rule_array parameter), a text length of 0 is valid for any segment control specified in the rule_array parameter (FIRST, MIDDLE, LAST, or ONLY). When LAST or ONLY is specified, the supplied text will be padded with X’FF’s and a padding count in the last byte to bring the total text length to the next multiple of 8 that is greater than or equal to 16.

- When no padding is requested (by specifying a process rule of MDC-2 or MDC-4), the total length of the text provided (over a single or segmented calls) must be at least 16 bytes, and a multiple of 8.

  For segmented calls with no padding, text length of 0 is valid on any of the calls provided the total length over the segmented calls is at least 16 and a multiple of 8.

  For a single call (that is, segment control is ONLY) with no padding, the length the text provided must be at least 16, and a multiple of 8.

text
Direction: Input Type: String

The application-supplied text for which the MDC is generated.
MDC Generate (CSNBMDG and CSNBMDG1)

**rule_array_count**

Direction: Input  
Type: Integer

The number of keywords specified in the `rule_array` parameter. This value must be 2.

**rule_array**

Direction: Input  
Type: Character string

The two keywords that provide control information to the callable service are shown in Table 95. The two keywords must be in 16 bytes of contiguous storage with each of the two keywords left-justified in its own 8-byte location and padded on the right with blanks. For example, 

```
'MDC-2    FIRST    '
```

Choose one of the MDC process rule control keywords and one of the segmenting control keywords from the following table.

*Table 95. Keywords for MDC Generate Control Information*

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MDC Process Rules (required)</strong></td>
<td></td>
</tr>
<tr>
<td>MDC-2</td>
<td>MDC-2 specifies two encipherments per 8 bytes of input text and no padding of the input text.</td>
</tr>
<tr>
<td>MDC-4</td>
<td>MDC-4 specifies four encipherments per 8 bytes of input text and no padding of the input text.</td>
</tr>
<tr>
<td>PADMDC-2</td>
<td>PADMDC-2 specifies two encipherments per 8 bytes of input text and padding of the input text.</td>
</tr>
<tr>
<td></td>
<td>When the segment rule specifies ONLY or LAST, the input text is padded with X'FF's and a padding count in the last byte to bring the total text length to the next even multiple of 8 that is greater than, or equal to, 16.</td>
</tr>
<tr>
<td>PADMDC-4</td>
<td>PADMDC-4 specifies four encipherments per 8 bytes of input text and padding of the input text.</td>
</tr>
<tr>
<td></td>
<td>When the segment rule specifies ONLY or LAST, the input text is padded with X'FF's and a padding count in the last byte to bring the total text length to the next even multiple of 8 that is greater than, or equal to, 16.</td>
</tr>
<tr>
<td><strong>Segmenting Control (required)</strong></td>
<td></td>
</tr>
<tr>
<td>FIRST</td>
<td>First call; this is the first segment of data from the application program.</td>
</tr>
<tr>
<td>LAST</td>
<td>Last call; this is the last data segment.</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>Middle call; this is an intermediate data segment.</td>
</tr>
<tr>
<td>ONLY</td>
<td>Only call; segmenting is not employed by the application program.</td>
</tr>
</tbody>
</table>

**chaining_vector**

Direction: Input/Output  
Type: String
MDC Generate (CSNBMDG and CSNBMDG1)

An 18-byte string that ICSF uses as a system work area. Your application program must not change the data in this string. The chaining vector permits data to be chained from one invocation call to another.

On the first call, initialize this parameter as binary zeros.

**mdc**

Direction: Input/Output  
Type: String

A 16-byte field in which the callable service returns the MDC value when the segmenting rule is ONLY or LAST. When the segmenting rule is FIRST or MIDDLE, the value returned in this field is an intermediate MDC value that will be used as input for a subsequent call and must not be changed by the application program.

**text_id_in**

Direction: Input  
Type: Integer

For CSNBMDG1 only, the ALET for the data space containing the text for which the MDC is to be generated.

**Usage Notes**

To calculate an MDC in one call, specify the ONLY keyword for segmenting control in the **rule_array** parameter. For more than one call, specify the FIRST keyword for the first input block, the MIDDLE keyword for any intermediate blocks, and the LAST keyword for the last block. For a given text string, the resulting MDC is the same whether the text is segmented or not.

The two versions of MDC calculation (with two or four encipherments per 8 bytes of input text) allow the caller to trade a performance improvement for a decrease in security. Since 2 encipherments create results different from the results of 4 encipherments, ensure that you use the same number of encipherments to verify the MDC value.

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 96. MDC generate required hardware*

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td></td>
</tr>
<tr>
<td>IBM zSeries 900</td>
<td>Cryptographic Coprocessor Feature</td>
<td></td>
</tr>
<tr>
<td>IBM zSeries 990</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
<tr>
<td>IBM zSeries 890</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
</tbody>
</table>

Chapter 7. Verifying Data Integrity and Authenticating Messages 285
MDC Generate (CSNBMDG and CSNBMDG1)

Table 96. MDC generate required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM System z10 EC</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

One-Way Hash Generate (CSNBOWH, CSNEOWH and CSNBOWH1)

Use the one-way hash generate callable service to generate a one-way hash on specified text. This service supports the following methods:
- MD5 - software only
- SHA-1
- RIPEMD-160 - software only
- SHA-224
- SHA-256
- SHA-384
- SHA-512

This callable service supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSNEOWH.

Format

CALL CSNBOWH(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  rule_array_count,
  rule_array,
  text_length,
  text,
  chaining_vector_length,
  chaining_vector,
  hash_length,
  hash)

CALL CSNBOWH1(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  rule_array_count,
  rule_array,
  text_length,
  text,
  chaining_vector_length,
  chaining_vector,
  hash_length,
  hash,
  text_id_in)
One-Way Hash Generate (CSNBOWH, CSNEOWH and CSNBOWH1)

Parameters

return_code
Direction: Output   Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

reason_code
Direction: Output   Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

exit_data_length
Direction: Input/Output   Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

exit_data
Direction: Input/Output   Type: String

The data that is passed to the installation exit.

rule_array_count
Direction: Input   Type: Integer

The number of keywords you are supplying in the rule_array parameter. The value must be 1 or 2.

rule_array
Direction: Input   Type: String

Keywords that provide control information to the callable service are listed in Table 97. The optional chaining flag keyword indicates whether calls to this service are chained together logically to overcome buffer size limitations. Each keyword is left-justified in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage.

Table 97. Keywords for One-Way Hash Generate Rule Array Control Information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash Method (required)</td>
<td></td>
</tr>
<tr>
<td>MD5</td>
<td>Hash algorithm is MD5 algorithm. Use this hash method for PKCS-1.0 and PKCS-1.1. Length of hash generated is 16 bytes.</td>
</tr>
<tr>
<td>RPMD-160</td>
<td>Hash algorithm is RIPEMD-160. Length of hash generated is 20 bytes.</td>
</tr>
</tbody>
</table>
**One-Way Hash Generate (CSNBOWH, CSNEOWH and CSNBOWH1)**

Table 97. Keywords for One-Way Hash Generate Rule Array Control Information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-1</td>
<td>Hash algorithm is SHA-1 algorithm. Use this hash method for DSS. Length of hash generated is 20 bytes.</td>
</tr>
<tr>
<td>SHA-224</td>
<td>Hash algorithm is SHA-256 algorithm. Length of hash generated is 28 bytes.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>Hash algorithm is SHA-256 algorithm. Length of hash generated is 32 bytes.</td>
</tr>
<tr>
<td>SHA-384</td>
<td>Hash algorithm is SHA-384 algorithm. Length of hash generated is 48 bytes.</td>
</tr>
<tr>
<td>SHA-512</td>
<td>Hash algorithm is SHA-512 algorithm. Length of hash generated is 64 bytes.</td>
</tr>
</tbody>
</table>

**Chaining Flag (optional)**

<table>
<thead>
<tr>
<th>Chaining Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIRST</td>
<td>Specifies this is the first call in a series of chained calls. Intermediate results are stored in the hash field.</td>
</tr>
<tr>
<td>LAST</td>
<td>Specifies this is the last call in a series of chained calls.</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>Specifies this is a middle call in a series of chained calls. Intermediate results are stored in the hash field.</td>
</tr>
<tr>
<td>ONLY</td>
<td>Specifies this is the only call and the call is not chained. This is the default.</td>
</tr>
</tbody>
</table>

**text_length**

Direction: Input  
Type: Integer

The length of the text parameter in bytes.

**Note:** If you specify the FIRST or MIDDLE keyword, then the text length must be a multiple of the blocksize of the hash method. For MD5, RPMD-160, SHA-1, SHA-224 and SHA-256, this is a multiple of 64 bytes. For SHA-384 and SHA-512, this is a multiple of 128 bytes.

For ONLY and LAST, this service performs the required padding according to the algorithm specified.

**text**

Direction: Input  
Type: String

The application-supplied text on which this service performs the hash.

**chaining_vector_length**

Direction: Input  
Type: Integer

The byte length of the chaining_vector parameter. This must be 128 bytes.

**chaining_vector**

Direction: Input/Output  
Type: String

This field is a 128-byte work area. Your application must not change the data in this string. The chaining vector permits chaining data from one call to another.
One-Way Hash Generate (CSNBOWH, CSNEOWH and CSNBOWH1)

hash_length

Direction: Input  Type: Integer

The length of the supplied hash field in bytes.

**Note:** For SHA-1 and RPMD-160 this must be at least 20 bytes; for MD5 this must be at least 16 bytes. For SHA-224 and SHA-256, the length must be at least 32 bytes long. Even though the length of the SHA-224 hash is less than SHA-256, the extra bytes are used as a work area during the generation of the hash value. The SHA-224 value is left-justified and padded with zeroes.

For SHA-384 and SHA-512, the length must be at least 64 bytes long. Even though the length of the SHA-384 hash is less than SHA-512, the extra bytes are used as a work area during the generation of the hash value. The SHA-384 value is left-justified and padded with zeroes.

hash

Direction: Input/Output  Type: String

This field contains the hash, left-justified. The processing of the rest of the field depends on the implementation. If you specify the FIRST or MIDDLE keyword, this field contains the intermediate hash value. Your application must not change the data in this field between the sequence of FIRST, MIDDLE, and LAST calls for a specific message.

text_id_in

Direction: Input  Type: Integer

For CSNBOWH1 only, the ALET for the data space containing the text for which to generate the hash.

**Usage Notes**

Although MD5, SHA-1 and SHA-256 allow it, bit length text is not supported for any hashing method.

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

**Table 98. One-way hash generate required hardware**

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>SHA-1 requires CCF</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td>SHA-224 keyword not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHA-256 keyword not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHA-384 keyword not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHA-512 keyword not supported</td>
</tr>
</tbody>
</table>
One-Way Hash Generate (CSNBOWH, CSNEOWH and CSNBOWH1)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 990</td>
<td>CP Assist for Cryptographic Functions</td>
<td>SHA-1 requires CPACF</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>CP Assist for Cryptographic Functions</td>
<td>SHA-224 keyword not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHA-256 keyword not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHA-384 keyword not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHA-512 keyword not supported</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>CP Assist for Cryptographic Functions</td>
<td>SHA-384 keyword not supported</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td>SHA-512 keyword not supported</td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>CP Assist for Cryptographic Functions</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Symmetric MAC Generate (CSNBSMG, CSNBSMG1, CSNESMG, and CSNESMG1)

Use the symmetric MAC generate callable service to generate a 96- or 128-bit message authentication code (MAC) for an application-supplied text string using an AES key.

The callable service Symmetric MAC Generate supports invocation in AMODE (64). The callable service names for AMODE (64) invocation are CSNESMG and CSNESMG1.

Choosing Between CSNBSMG and CSNBSMG1 or CSNESMG and CSNESMG1

CSNBSMG, CSNBSMG1, CSNESMG, and CSNESMG1 provide identical functions. When choosing which service to use, consider this:

- CSNBSMG and CSNESMG require the text to reside in the caller’s primary address space. Also, a program using CSNBSMG adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.
- CSNBSMG1 and CSNESMG1 allow the text to reside either in the caller’s primary address space or in a data space. This can allow you to decipher more data with one call. However, a program using CSNBSMG1 and CSNESMG1 do not adhere to the IBM CCA: Cryptographic API and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

For CSNBSMG1 and CSNESMG1, text_id_in is an access list entry token (ALET) parameter of the data spaces containing the text.
Symmetric MAC Generate (CSNBSMG, CSNBSMG1, CSNESMG, and CSNESMG1)

Format

CALL CSNBSMG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier_length,
    key_identifier,
    text_length,
    text,
    rule_array_count,
    rule_array,
    chaining_vector_length,
    chaining_vector,
    reserved_data_length,
    reserved_data
    mac_length
    mac )

CALL CSNBSMG1(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier_length,
    key_identifier,
    text_length,
    text,
    rule_array_count,
    rule_array,
    chaining_vector_length,
    chaining_vector,
    reserved_data_length,
    reserved_data
    mac_length
    mac
    text_id_in)

Parameters

return_code
Direction: Output Type: Integer

The return code specifies the general result of the callable service. [Appendix A, “ICSF and TSS Return and Reason Codes”] lists the return codes.

reason_code
Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. [Appendix A, “ICSF and TSS Return and Reason Codes”] lists the reason codes.

exit_data_length
Direction: Input/Output Type: Integer
Symmetric MAC Generate (CSNBSMG, CSNBSMG1, CSNESMG, and CSNESMG1)

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**
Direction: Input/Output Type: String

The data that is passed to the installation exit.

**key_identifier_length**
Direction: Input Type: String

The length of the key_identifier parameter. For the KEY-CLR keyword, the length is in bytes and includes only the value of the key length. The key length value can be 16, 24, or 32. For the KEYIDENT keyword, the length must be 64.

**key_identifier**
Direction: Input Type: String

For the KEY-CLR keyword, this specifies the clear AES key. The parameter must be left justified. For the KEYIDENT keyword, this specifies an internal clear AES token or the label name of a clear AES key in the CKDS. Normal CKDS label name syntax is required.

**text_length**
Direction: Input Type: Integer

The length of the text you supply in the text parameter. The maximum length of text is 2147483647 bytes. If the text_length is not a multiple of 8 bytes and if the ONLY or LAST keyword of the rule_array parameter is called, the text is padded in accordance with the processing rule specified.

**text**
Direction: Input Type: String

The application-supplied text for which the MAC is generated.

**rule_array_count**
Direction: Input Type: Integer

The number of keywords specified in the rule_array parameter. The value can be 1, 2, 3 or 4.

**rule_array**
Direction: Input Type: Character string

This keyword provides control information to the callable service. The keywords must be eight bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks.

You can specify one of the MAC processing rules and then choose one of the segmenting control keywords and one of the MAC length keywords.
### Table 99. Keywords for symmetric MAC generate control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithm (required)</strong></td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>Specifies that the Advanced Encryption Standard (AES) algorithm is to be used.</td>
</tr>
<tr>
<td><strong>MAC processing rule (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>CBC-MAC</td>
<td>CBC MAC with padding for any key length. This is the default value.</td>
</tr>
<tr>
<td><strong>Key rule (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>KEY-CLR</td>
<td>This specifies that the key parameter contains a clear key value. This is the default value.</td>
</tr>
<tr>
<td>KEYIDENT</td>
<td>This specifies that the key_identifier field will be an internal clear token or the label name of a clear key in the CKDS. Normal CKDS label name syntax is required.</td>
</tr>
<tr>
<td><strong>Segmenting Control (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>FIRST</td>
<td>First call, this is the first segment of data from the application program.</td>
</tr>
<tr>
<td>LAST</td>
<td>Last call; this is the last data segment.</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>Middle call; this is an intermediate data segment.</td>
</tr>
<tr>
<td>ONLY</td>
<td>Only call; segmenting is not employed by the application program. This is the default value.</td>
</tr>
</tbody>
</table>

**chaining_vector_length**

Direction: Input/Output  
Type: Integer

The length of the `chaining_vector` parameter. On output, the actual length of the chaining vector will be stored in the parameter.

**chaining_vector**

Direction: Input/Output  
Type: String

This field is used as a system work area for the chaining vector. Your application program must not change the data in this string. The chaining vector holds the output chaining vector from the caller.

The mapping of the `chaining_vector` depends on the algorithm specified. For AES, the `chaining_vector` field must be at least 36 bytes in length.

**reserved_data_length**

Direction: Input  
Type: Integer

Reserved for future use. Value must be zero.

**reserved_data**

Direction: Ignored  
Type: String

Reserved for future use.
Symmetric MAC Generate (CSNBSMG, CSNBSMG1, CSNESMG, and CSNESMG1)

**mac_length**

*Direction:* Input  
*Type:* Integer

The length in bytes of the MAC to be returned in the mac field. The allowable values are 12 and 16 bytes.

**mac**

*Direction:* Output  
*Type:* String

The 12-byte or 16-byte field in which the callable service returns the MAC value if the segmenting rule is ONLY or LAST.

**text_id_in**

*Direction:* Input  
*Type:* Integer

For CSNBSMG1 and CSNESMG1 only, the ALET of the text for which the MAC is generated.

**Usage Notes**

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 100. Symmetric MAC generate required hardware*

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>CPACF</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>CPACF</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>CPACF</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Symmetric MAC Verify (CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1)

Use the symmetric MAC verify callable service to verify a 96- or 128-bit message authentication code (MAC) for an application-supplied text string using an AES key.

The callable service Symmetric MAC Verify supports invocation in AMODE (64). The callable service names for AMODE (64) invocation are CSNESMV and CSNESMV1.
Symmetric MAC Verify (CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1)

Choosing Between CSNBSMV and CSNBSMV1 or CSNESMV and CSNESMV1

CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1 provide identical functions. When choosing which service to use, consider this:

- CSNBSMV and CSNESMV require the text to reside in the caller's primary address space. Also, a program using CSNBSMV adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.
- CSNBSMV1 and CSNESMV1 allow the text to reside either in the caller's primary address space or in a data space. This can allow you to decipher more data with one call. However, a program using CSNBSMV1 and CSNESMV1 do not adhere to the IBM CCA: Cryptographic API and may need to be modified prior to it running with other cryptographic products that follow this programming interface.

For CSNBSMV1 and CSNESMV1, text_id_in is an access list entry token (ALET) parameter of the data spaces containing the text.

Format

CALL CSNBSMV(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier_length,
    key_identifier,
    text_length,
    text,
    rule_array_count,
    rule_array,
    chaining_vector_length,
    chaining_vector,
    reserved_data_length,
    reserved_data,
    mac_length,
    mac
)

CALL CSNBSMV1(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    key_identifier_length,
    key_identifier,
    text_length,
    text,
    rule_array_count,
    rule_array,
    chaining_vector_length,
    chaining_vector,
    reserved_data_length,
    reserved_data,
    mac_length,
    mac
    text_id_in )
Symmetric MAC Verify (CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1)

Parameters

return_code
Direction: Output  Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code
Direction: Output  Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

exit_data_length
Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

exit_data
Direction: Input/Output  Type: String

The data that is passed to the installation exit.

key_identifier_length
Direction: Input  Type: Integer

The length of the key_identifier parameter. For the KEY-CLR keyword, the length is in bytes and includes only the value of the key length. The key length value can be 16, 24, or 32. For the KEYIDENT keyword, the length must be 64.

key_identifier
Direction: Input  Type: String

For the KEY-CLR keyword, this specifies the clear AES key. The parameter must be left justified. For the KEYIDENT keyword, this specifies an internal clear AES token or the label name of a clear AES key in the CKDS. Normal CKDS label name syntax is required.

text_length
Direction: Input  Type: Integer

The length of the text you supply in the text parameter. The maximum length of text is 2147483647 bytes. If the text_length parameter is not a multiple of 8 bytes and if the ONLY or LAST keyword of the rule_array parameter is called, the text is padded in accordance with the processing rule specified.
Symmetric MAC Verify (CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1)

text
Direction: Input                Type: String

The application-supplied text for which the MAC is verified.

rule_array_count
Direction: Input                Type: Integer

The number of keywords specified in the rule_array parameter. The value can be 1, 2, 3 or 4.

rule_array
Direction: Input                Type: String

This keyword provides control information to the callable service. The keywords must be eight bytes of contiguous storage with the keyword left-justified in its 8-byte location and padded on the right with blanks. The order of the rule_array keywords is not fixed.

You can specify one of the MAC processing rules and then choose one of the segmenting control keywords and one of the MAC length keywords.

Table 101. Keywords for symmetric MAC verify control information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm (required)</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>Specifies that the Advanced Encryption Standard (AES) algorithm is to be used.</td>
</tr>
<tr>
<td>MAC processing rule (optional)</td>
<td></td>
</tr>
<tr>
<td>CBC-MAC</td>
<td>CBC MAC with padding for any key length. This is the default value.</td>
</tr>
<tr>
<td>Key rule (optional)</td>
<td></td>
</tr>
<tr>
<td>KEY-CLR</td>
<td>This specifies that the key parameter contains a clear key value. This is the default value.</td>
</tr>
<tr>
<td>KEYIDENT</td>
<td>This specifies that the key_identifier field will be an internal clear token or the label name of a clear key in the CKDS. Normal CKDS label name syntax is required.</td>
</tr>
<tr>
<td>Segmenting Control (optional)</td>
<td></td>
</tr>
<tr>
<td>FIRST</td>
<td>First call, this is the first segment of data from the application program.</td>
</tr>
<tr>
<td>LAST</td>
<td>Last call; this is the last data segment.</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>Middle call; this is an intermediate data segment.</td>
</tr>
<tr>
<td>ONLY</td>
<td>Only call; segmenting is not employed by the application program. This is the default value.</td>
</tr>
</tbody>
</table>

chaining_vector_length
Direction: Input/Output               Type: String
Symmetric MAC Verify (CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1)

The length of the chaining_vector parameter. On output, the actual length of the chaining vector will be stored in the parameter.

**chaining_vector**

Direction: Input/Output  Type: String

This field is used as a system work area for the chaining vector. Your application program must not change the data in this string. The chaining vector holds the output chaining vector from the caller.

The mapping of the chaining_vector depends on the algorithm specified. For AES, the chaining_vector field must be at least 36 bytes in length.

**reserved_data_length**

Direction: Input  Type: Integer

Reserved for future use. Value must be zero.

**reserved_data**

Direction: Ignored  Type: String

Reserved for future use.

**mac_length**

Direction: Input  Type: Integer

The length in bytes of the MAC to be verified the mac field. The allowable values are 12 and 16 bytes.

**mac**

Direction: Input  Type: String

The 12-byte or 16-byte field that contains the MAC value you want to verify. The value must be left-justified and padded with zeros.

**text_id_in**

Direction: Input  Type: Integer

For CSNBSMV1 and CSNESMV1 only, the ALET of the text for which the MAC is to be verified.

**Usage Notes**

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>Cryptographic Coprocessor Feature</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>CPACF</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>CPACF</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>CPACF</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>CPACF</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>CPACF</td>
<td></td>
</tr>
</tbody>
</table>
Symmetric MAC Verify (CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1)
Chapter 8. Financial Services

The process of validating personal identities in a financial transaction system is called personal authentication. The personal identification number (PIN) is the basis for verifying the identity of a customer across financial industry networks. ICSF provides callable services to translate, verify, and generate PINs. You can use the callable services to prevent unauthorized disclosures when organizations handle PINs.

These callable services are described in these topics:

- "Clear PIN Encrypt (CSNBCPE)" on page 309
- "Clear PIN Generate (CSNBPGN)" on page 313
- "Clear PIN Generate Alternate (CSNBCPA)" on page 317
- "Encrypted PIN Generate (CSNBEPG)" on page 323
- "Encrypted PIN Translate (CSNBPTR)" on page 328
- "Encrypted PIN Verify (CSNBPVr)" on page 335
- "PIN Change/Unblock (CSNBPCU)" on page 341
- "Secure Messaging for Keys (CSNBSKY)" on page 347
- "Secure Messaging for PINs (CSNBSPN)" on page 350
- "SET Block Compose (CSNDSCB)" on page 355
- "SET Block Decompose (CSNDSD)" on page 360
- "Transaction Validation (CSNBTTRV)" on page 365
- "VISA CVV Service Generate (CSNCVSG)" on page 369
- "VISA CVV Service Verify (CSNCSV)" on page 373

How Personal Identification Numbers (PINs) are Used

Many people are familiar with PINs, which allow them to use an automated teller machine (ATM). From the system point of view, PINs are used primarily in financial networks to authenticate users — typically, a user is assigned a PIN, and enters the PIN at automated teller machines (ATMs) to gain access to his or her accounts. It is extremely important that the PIN be kept private, so that no one other than the account owner can use it. ICSF allows your applications to generate PINs, to verify supplied PINs, and to translate PINs from one format to another.

How VISA Card Verification Values Are Used

The Visa International Service Association (VISA) and MasterCard International, Incorporated have specified a cryptographic method to calculate a value that relates to the personal account number (PAN), the card expiration date, and the service code. The VISA card-verification value (CVV) and the MasterCard card-verification code (CVC) can be encoded on either track 1 or track 2 of a magnetic striped card and are used to detect forged cards. Because most online transactions use track-2, the ICSF callable services generate and verify the CVV by the track-2 method.

The VISA CVV service generate callable service calculates a 1- to 5-byte value through the DES-encryption of the PAN, the card expiration date, and the service code using two data-encrypting keys or two MAC keys. The VISA CVV service verify callable service calculates the CVV by the same method, compares it to the CVV supplied by the application (which reads the credit card’s magnetic stripe) in the CVV_value, and issues a return code that indicates whether the card is authentic.

4. The VISA CVV and the MasterCard CVC refer to the same value. CVV is used here to mean both CVV and CVC.
Translating Data and PINs in Networks

More and more data is being transmitted across networks where, for various reasons, the keys used on one network cannot be used on another network. Encrypted data and PINs that are transmitted across these boundaries must be “translated” securely from encryption under one key to encryption under another key. For example, a traveler visiting a foreign city might wish to use an ATM to access an account at home. The PIN entered at the ATM might need to be encrypted at the ATM and sent over one or more financial networks to the traveler’s home bank. At the home bank, the PIN must be verified prior to access being allowed. On intermediate systems (between networks), applications can use the Encrypted PIN translate callable service to re-encrypt a PIN block from one key to another. Running on ICSF, such applications can ensure that PINs never appear in the clear and that the PIN-encrypting keys are isolated on their own networks.

Working with Europay–MasterCard–Visa smart cards

There are several services you can use in secure communications with EMV smart cards. The processing capabilities are consistent with the specifications provided in these documents:

- EMV 2000 Integrated Circuit Card Specification for Payment Systems Version 4.0 (EMV4.0) Book 2
- Integrated Circuit Card Specification (VIS) 1.4.0 Corrections

EMV smart cards include the following processing capabilities:

- The diversified key generate (CSNBDKG) callable service with rule-array options TDES-XOR, TDESEMV2, and TDESEMV4 enables you to derive a key used to cipher and authenticate messages, and more particularly message parts, for exchange with an EMV smart card. You use the derived key with services such as encipher, decipher, MAC generate, MAC verify, secure messaging for keys, and secure messaging for PINs. These message parts can be combined with message parts created using the secure messaging for keys and secure messaging for PINs services.

- The secure messaging for keys (CSNBSKY) service enables you to securely incorporate a key into a message part (generally the value portion of a TLV component of a secure message for a card). Similarly, the secure messaging for PINs (CSNBSPN) service enables secure incorporation of a PIN block into a message part.

- The PIN change/unblock (CSNBPCU) service enables you to encrypt a new PIN to send to a new EMV card, or to update the PIN value on an initialized EMV card. This verb generates both the required session key (from the master encryption key) and the required authentication code (from the master authentication key).

- The ZERO-PAD option of the PKA encrypt (CS NDPKE) service enables you to validate a digital signature created according to ISO 9796-2 standard by encrypting information you format, including a hash value of the message to be validated. You compare the resulting enciphered data to the digital signature accompanying the message to be validated.

- The MAC generate and MAC verify services post-pad a X'80'...X'00' string to a message as required for authenticating messages exchanged with EMV smart cards.
PIN Callable Services

You use the PIN callable services to generate, verify, and translate PINs. This topic discusses the PIN callable services, as well as the various PIN algorithms and PIN block formats supported by ICSF. It also explains the use of PIN-encrypting keys.

Generating a PIN

To generate personal identification numbers, call the Clear PIN Generate or Encrypted PIN Generate callable service. Using a PIN generation algorithm, data used in the algorithm, and the PIN generation key, the Clear PIN generate callable service generates a clear PIN and a PIN verification value, or offset. The Clear PIN Generate callable service can only execute in special secure mode. For a description of this mode, see "Special Secure Mode" on page 10. Using a PIN generation algorithm, data used in the algorithm, the PIN generation key, and an outbound PIN encrypting key, the encrypted PIN generate callable service generates and formats a PIN and encrypts the PIN block.

Encrypting a PIN

To format a PIN into a supported PIN block format and encrypt the PIN block, call the Clear PIN encrypt callable service.

Generating a PIN Validation Value from an Encrypted PIN Block

To generate a clear VISA PIN validation value (PVV) from an encrypted PIN block, call the clear PIN generate alternate callable service. The PIN block can be encrypted under an input PIN-encrypting key (IPINENC) or an output PIN encrypting key (OPINENC). Using an IPINENC key requires that NOCV keys are enabled in the CKDS.

Verifying a PIN

To verify a supplied PIN, call the Encrypted PIN verify callable service. You supply the enciphered PIN, the PIN-encrypting key that enciphers the PIN, and other data. You must also specify the PIN verification key and PIN verification algorithm. The callable service generates a verification PIN. The service compares the two personal identification numbers and if they are the same, it verifies the supplied PIN.

Translating a PIN

To translate a PIN block format from one PIN-encrypting key to another or from one PIN block format to another, call the Encrypted PIN translate callable service. You must identify the input PIN-encrypting key that originally enciphered the PIN. You also need to specify the output PIN-encrypting key that you want the callable service to use to encipher the PIN. If you want to change the PIN block format, specify a different output PIN block format from the input PIN block format.

Algorithms for Generating and Verifying a PIN

ICSF supports these algorithms for generating and verifying personal identification numbers:
- IBM 3624 institution-assigned PIN
- IBM 3624 customer-selected PIN (through a PIN offset)
- IBM German Bank Pool PIN (verify through an institution key)
- IBM German Bank Pool PIN (verify through a pool key and a PIN offset). This algorithm is supported when the service using the PIN is processed on the Cryptographic Coprocessor Feature. **Restriction:** This algorithm is not supported on a z990, z890, z9 EC or z9 BC.
- VISA PIN through a VISA PIN validation value
- Interbank PIN

The algorithms are discussed in detail in [“PIN Formats and Algorithms” on page 669.](#)

**Using PINs on Different Systems**

ICSF allows you to translate different PIN block formats, which lets you use personal identification numbers on different systems. ICSF supports these formats:
- IBM 3624
- IBM 3621 (same as IBM 5906)
- IBM 4704 encrypting PINPAD format
- ISO 0 (same as ANSI 9.8, VISA 1, and ECI 1)
- ISO 1 (same as ECI 4)
- ISO 2
- ISO 3
- VISA 2
- VISA 3
- VISA 4
- ECI 2
- ECI 3

The formats are discussed in [“PIN Formats and Algorithms” on page 669.](#)

**PIN-Encrypting Keys**

A unique master key variant enciphers each type of key. For further key separation, an installation can choose to have each PIN block format enciphered under a different PIN-encrypting key. The PIN-encrypting keys can have a 16-byte PIN block variant constant exclusive ORed on them prior to using to translate or verify PIN blocks. This is specified in the format control field in the Encrypted PIN translate and Encrypted PIN verify callable services.

You should only use PIN block variant constants when you are communicating with another host processor with the Integrated Cryptographic Service Facility.

**Derived Unique Key Per Transaction Algorithms**

ICSF supports ANSI X9.24 derived unique key per transaction algorithms to generate PIN-encrypting keys from user data. ICSF supports both single- and double-length key generation. Keywords for single- and double-length key generation can not be mixed. A PCICC, PCIXCC, CEX2C, or CEX3C is required for this support. Double-length key generation is only supported on z990 with the May 2004 LIC or higher, z890, z9 EC, z9 BC and IBM System z10 EC.

**Encrypted PIN Translate**

The UKPTIPIN, IPKTOPIN and UKPTBOTH keywords will cause the service to generate single-length keys. DUKPT-IP, DKPT-OP and DUKPT-BH are the respective keywords to generate double-length keys. The `input_PIN_profile` and `output_PIN_profile` must supply the current key serial number when these keywords are specified.
Encrypted PIN Verify
The UKPTIPIN keyword will cause the service to generate single-length keys. DUKPT-IP is the keyword for double-length key generation. The input_PIN_profile must supply the current key serial number when these keywords are specified.

For more information about PIN-encrypting keys, see [z/OS Cryptographic Services](ICSF Administrator's Guide).

The PIN Profile
The PIN profile consists of:
- PIN block format (see “PIN Block Format”)
- Format control (see “Format Control” on page 308)
- Pad digit (see “Pad Digit” on page 308)
- Current Key Serial Number (for UKPT and DUKPT – see “Current Key Serial Number” on page 309)

Table 103 shows the format of a PIN profile.

Table 103. Format of a PIN Profile

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–7</td>
<td>PIN block format</td>
</tr>
<tr>
<td>8–15</td>
<td>Format control</td>
</tr>
<tr>
<td>16–23</td>
<td>Pad digit</td>
</tr>
<tr>
<td>24–47</td>
<td>Current Key Serial Number (for UKPT and DUKPT )</td>
</tr>
</tbody>
</table>

PIN Block Format
This keyword specifies the format of the PIN block. The 8-byte value must be left-justified and padded with blanks. Refer to Table 104 for a list of valid values.

Table 104. Format Values of PIN Blocks

<table>
<thead>
<tr>
<th>Format Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECI-2</td>
<td>Eurocheque International format 2</td>
</tr>
<tr>
<td>ECI-3</td>
<td>Eurocheque International format 3</td>
</tr>
<tr>
<td>ISO-0</td>
<td>ISO format 0, ANSI X9.8, VISA 1, and ECI 1</td>
</tr>
<tr>
<td>ISO-1</td>
<td>ISO format 1 and ECI 4</td>
</tr>
<tr>
<td>ISO-2</td>
<td>ISO format 2</td>
</tr>
<tr>
<td>ISO-3</td>
<td>ISO format 3</td>
</tr>
<tr>
<td>VISA-2</td>
<td>VISA format 2</td>
</tr>
<tr>
<td>VISA-3</td>
<td>VISA format 3</td>
</tr>
<tr>
<td>VISA-4</td>
<td>VISA format 4</td>
</tr>
<tr>
<td>3621</td>
<td>IBM 3621 and 5906</td>
</tr>
<tr>
<td>3624</td>
<td>IBM 3624</td>
</tr>
<tr>
<td>4704-EPP</td>
<td>IBM 4704 with encrypting PIN pad</td>
</tr>
</tbody>
</table>

PIN Block Format and PIN Extraction Method Keywords
In the Clear PIN Generate Alternate, Encrypted PIN Translate and Encrypted PIN Verify callable services, you may specify a PIN extraction keyword for a given PIN...
block format. In this table, the allowable PIN extraction methods are listed for each PIN block format. The first PIN extraction method keyword listed for a PIN block format is the default. If you specify a PIN extraction method keyword that is not the default, the request will be routed to the PCI Cryptographic Coprocessor.

<table>
<thead>
<tr>
<th>PIN Block Format</th>
<th>PIN Extraction Method Keywords</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECI-2</td>
<td>PINLEN04</td>
<td>The PIN extraction method keywords specify a PIN extraction method for a PINLEN04 format.</td>
</tr>
<tr>
<td>ECI-3</td>
<td>PINBLOCK</td>
<td>The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.</td>
</tr>
<tr>
<td>ISO-0</td>
<td>PINBLOCK</td>
<td>The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.</td>
</tr>
<tr>
<td>ISO-1</td>
<td>PINBLOCK</td>
<td>The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.</td>
</tr>
<tr>
<td>ISO-2</td>
<td>PINBLOCK</td>
<td>The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.</td>
</tr>
<tr>
<td>ISO-3</td>
<td>PINBLOCK</td>
<td>The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.</td>
</tr>
<tr>
<td>VISA-2</td>
<td>PINBLOCK</td>
<td>The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.</td>
</tr>
<tr>
<td>VISA-3</td>
<td>PINBLOCK</td>
<td>The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.</td>
</tr>
<tr>
<td>VISA-4</td>
<td>PINBLOCK</td>
<td>The PIN extraction method keywords specify a PIN extraction method for a PINBLOCK format.</td>
</tr>
<tr>
<td>3621</td>
<td>PADDIGIT, HEXDIGIT, PINLEN04 to PINLEN12, PADEXIST</td>
<td>The PIN extraction method keywords specify a PIN extraction method for an IBM 3621 PIN block format. The first keyword, PADDIGIT, is the default PIN extraction method for the PIN block format.</td>
</tr>
</tbody>
</table>
Table 106. Callable Services Affected by Enhanced PIN Security Mode

<table>
<thead>
<tr>
<th>PIN-block format and PIN-extraction method</th>
<th>Callable Services Affected</th>
<th>PIN processing changes when Enhanced PIN Security Mode enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECI-2, 3621, or 3624 formats AND PINLENxx</td>
<td>PIN-block format and PIN-extraction method&lt;br&gt;Clear_PIN_Generate_Alternate&lt;br&gt;Encrypted_PIN_Translate&lt;br&gt;Encrypted_PIN_Verify</td>
<td>The PINLENxx keyword in rule_array parameter for PIN extraction method is not allowed if the Enhanced PIN Security Mode is enabled. <strong>Note:</strong> The services will fail with return code 8 reason code ‘7E0’x.</td>
</tr>
<tr>
<td>3621 or 3624 format and PADDIGIT</td>
<td>Clear_PIN_Generate_Alternate&lt;br&gt;Encrypted_PIN_Translate&lt;br&gt;Encrypted_PIN_Verify&lt;br&gt;PIN Change/Unblock</td>
<td>PIN extraction determines the PIN length by scanning from right to left until a digit, not equal to the pad digit, is found. The minimum PIN length is set at four digits, so scanning ceases one digit past the position of the 4th PIN digit in the block.</td>
</tr>
<tr>
<td>3621 or 3624 format and PADDIGIT</td>
<td>Clear_PIN_Encrypt&lt;br&gt;Clear_PIN_Generate&lt;br&gt;Encrypted_PIN_Translate&lt;br&gt;Encrypted_PIN_Translate</td>
<td>PIN formatting does not examine the PIN, in the output PIN block, to see if it contains the pad digit.</td>
</tr>
<tr>
<td>3621 or 3624 format and PADDIGIT</td>
<td>Encrypted_PIN_Translate</td>
<td>Restricted to non-decimal digit for PAD digit.</td>
</tr>
</tbody>
</table>
Format Control

This keyword specifies whether there is any control on the user-supplied PIN format. The 8-byte value must be left-justified and padded with blanks. Specify one of these values:

NONE  No format control.

PBVC  A PIN block variant constant (PBVC) enforces format control. Use the PBVC value only if you have coded PBVC in the encrypted PIN translate callable service. For the PBVC, the clear PIN key-encrypting key has been exclusive ORed with one of the PIN block formats. The cryptographic feature removes the pattern from the clear PIN key-encrypting key prior to it decrypting the PIN block.

Restriction: PBVC is not supported on IBM @server zSeries 990 and subsequent releases.

Notes:
1. Only control vectors and extraction methods valid for the Cryptographic Coprocessor Feature may be used if the PBVC format control is desired.
2. PBVC is supported for compatibility with prior releases of OS/390 ICSF and existing ICSF applications. It is recommended that a format control of NONE be specified for maximum flexibility to run on PCI Cryptographic Coprocessors.

If you do not specify a value for the format control parameter, ICSF uses hexadecimal zeros.

Table 120 on page 322 lists the PIN block variant constants.

Pad Digit

Some PIN formats require this parameter. If the PIN format does not need a pad digit, the callable service ignores this parameter. Table 107 shows the format of a pad digit. The PIN profile pad digit must be specified in upper case.

Table 107. Format of a Pad Digit

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16–22</td>
<td>Seven space characters</td>
</tr>
<tr>
<td>23</td>
<td>Character representation of a hexadecimal pad digit or a space if a pad digit is not needed. Characters must be one of these: 0–9, A–F, or a blank.</td>
</tr>
</tbody>
</table>

Each PIN format supports only a pad digit in a certain range. This table lists the valid pad digits for each PIN block format.

Table 108. Pad Digits for PIN Block Formats

<table>
<thead>
<tr>
<th>PIN Block Format</th>
<th>Output PIN Profile</th>
<th>Input PIN Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECI-2</td>
<td>Pad digit is not used</td>
<td>Pad digit is not used</td>
</tr>
<tr>
<td>ECI-3</td>
<td>Pad digit is not used</td>
<td>Pad digit is not used</td>
</tr>
<tr>
<td>ISO-0</td>
<td>F</td>
<td>Pad digit is not used</td>
</tr>
<tr>
<td>ISO-1</td>
<td>Pad digit is not used</td>
<td>Pad digit is not used</td>
</tr>
<tr>
<td>ISO-2</td>
<td>Pad digit is not used</td>
<td>Pad digit is not used</td>
</tr>
</tbody>
</table>
Table 108. Pad Digits for PIN Block Formats (continued)

<table>
<thead>
<tr>
<th>PIN Block Format</th>
<th>Output PIN Profile</th>
<th>Input PIN Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO-3</td>
<td>Pad digit is not used</td>
<td>Pad digit is not used</td>
</tr>
<tr>
<td>VISA-2</td>
<td>0 through 9</td>
<td>Pad digit is not used</td>
</tr>
<tr>
<td>VISA-3</td>
<td>0 through F</td>
<td>Pad digit is not used</td>
</tr>
<tr>
<td>VISA-4</td>
<td>F</td>
<td>Pad digit is not used</td>
</tr>
<tr>
<td>3621</td>
<td>0 through F</td>
<td>0 through F</td>
</tr>
<tr>
<td>3624</td>
<td>0 through F</td>
<td>0 through F</td>
</tr>
<tr>
<td>4704-EPP</td>
<td>F</td>
<td>Pad digit is not used</td>
</tr>
</tbody>
</table>

The callable service returns an error indicating that the PAD digit is not valid if all of these conditions are met:

1. The PTR Enhanced Security access control point is enabled in the active role
2. The output PIN profile specifies 3621 or 3624 as the PIN-block format
3. The output PIN profile specifies a decimal digit (0-9) as the PAD digit

**Recommendations for the Pad Digit**

IBM recommends that you use a nondecimal pad digit in the range of A through F when processing IBM 3624 and IBM 3621 PIN blocks. If you use a decimal pad digit, the creator of the PIN block must ensure that the calculated PIN does not contain the pad digit, or unpredictable results may occur.

For example, you can exclude a specific decimal digit from being in any calculated PIN by using the IBM 3624 calculation procedure and by specifying a decimalization table that does not contain the desired decimal pad digit.

**Current Key Serial Number**

The current key serial number is the concatenation of the initial key serial number (a 59-bit value) and the encryption counter (a 21-bit value). The concatenation is an 80-bit (10-byte) value. Table 109 shows the format of the current key serial number.

When UKPT or DUKPT is specified, the PIN profile parameter is extended to a 48-byte field and must contain the current key serial number.

<table>
<thead>
<tr>
<th>Table 109. Format of the Current Key Serial Number Field</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bytes</strong></td>
</tr>
<tr>
<td>24–47</td>
</tr>
</tbody>
</table>

**Clear PIN Encrypt (CSNBCPE)**

The Clear PIN Encrypt callable service formats a PIN into one of these PIN block formats and encrypts the results. You can use this service to create an encrypted PIN block for transmission. With the RANDOM keyword, you can have the service generate random PIN numbers.
Clear PIN Encrypt (CSNBCPE)

Note: A clear PIN is a sensitive piece of information. Ensure that your application program and system design provide adequate protection for any clear PIN value.

- IBM 3621 format
- IBM 3624 format
- ISO-0 format (same as the ANSI X9.8, VISA-1, and ECI formats)
- ISO-1 format (same as the ECI-4 format)
- ISO-2 format
- ISO-3 format
- IBM 4704 encrypting PINPAD (4704-EPP) format
- VISA 2 format
- VISA 3 format
- VISA 4 format
- ECI-2 format
- ECI-3 format

An enhanced PIN security mode, on PCICC, PCIXCC, CEX2C, or CEX3C, is available for formatting an encrypted PIN block into IBM 3621 format or IBM 3624 format. To do this, you must enable the PTR Enhanced PIN Security access control point in the default role. When activated, this mode limits checking of the PIN to decimal digits. No other PIN block consistency checking will occur.

Format

CALL CSNBCPE(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    PIN_encrypting_key_identifier,
    rule_array_count,
    rule_array,
    clear_PIN,
    PIN_profile,
    PAN_data,
    sequence_number
    encrypted_PIN_block )

Parameters

return_code
Direction: Output    Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

reason_code
Direction: Output    Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code can have different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.
Clear PIN Encrypt (CSNBCPE)

exit_data_length
Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is defined in the exit_data parameter.

exit_data
Direction: Input/Output  Type: String

The data that is passed to the installation exit.

PIN_encrypting_key_identifier
Direction: Input/Output  Type: String

The 64-byte string containing an internal key token or a key label of an internal key token. The internal key token contains the key that encrypts the PIN block. The control vector in the internal key token must specify an OPINENC key type and have the CPINENC usage bit set to 1.

rule_array_count
Direction: Input  Type: Integer

The number of keywords you are supplying in the rule_array parameter. Valid values are 0, 1 and 2.

rule_array
Direction: Input  Type: Character string

Keywords that provide control information to the callable service. The keyword is left-justified in an 8-byte field, and padded on the right with blanks. All keywords must be in contiguous storage. The rule array keywords are shown as follows:

<table>
<thead>
<tr>
<th>Process Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENCRYPT</td>
<td>This is the default. Use of this keyword is optional.</td>
</tr>
<tr>
<td>RANDOM</td>
<td>Causes the service to generate a random PIN value. The length of the PIN is based on the value in the clear_PIN variable. Set the value of the clear PIN to zero and use as many digits as the desired random PIN; pad the remainder of the clear PIN variable with space characters.</td>
</tr>
</tbody>
</table>

Table 110. Process Rules for the Clear PIN Encryption Callable Service

clear_PIN
Direction: Input  Type: String

A 16-character string with the clear PIN. The value in this variable must be left-justified and padded on the right with space characters.
Clear PIN Encrypt (CSNBCPE)

**PIN_profile**

Direction: Input  
Type: String

A 24-byte string containing three 8-byte elements with a PIN block format keyword, the format control keyword, NONE, and a pad digit as required by certain formats. See "The PIN Profile" on page 305 for additional information.

**PAN_data**

Direction: Input  
Type: String

A 12-byte PAN in character format. The service uses this parameter if the PIN profile specifies the ISO-0 or VISA-4 keyword for the PIN block format. Otherwise, ensure that this parameter is a 12-byte variable in application storage. The information in this variable will be ignored, but the variable must be specified.

**Note:** When using the ISO-0 keyword, use the 12 rightmost digits of the PAN data, excluding the check digit. When using the VISA-4 keyword, use the 12 leftmost digits of the PAN data, excluding the check digit.

**sequence_number**

Direction: Input  
Type: Integer

The 4-byte character integer. The service currently ignores the value in this variable. For future compatibility, the suggested value is 99999.

**encrypted_PIN_block**

Direction: Output  
Type: String

The field that receives the 8-byte encrypted PIN block.

**Restrictions**

The format control specified in the PIN profile must be NONE. If PBVC is specified as the format control, the service will fail.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

SAF will be invoked to check authorization to use the Clear PIN encrypt service and the label of the PIN_encrypting_key_identifier.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
Table 111. Clear PIN encrypt required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>PCI Cryptographic Coprocessor</td>
<td>ISO-3 PIN block format is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>ISO-3 PIN block format is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

Clear PIN Generate (CSNBPGN)

Use the Clear PIN generate callable service to generate a clear PIN, a PIN validation value (PVV), or an offset according to an algorithm. You supply the algorithm or process rule using the rule_array parameter.

- IBM 3624 (IBM-PIN or IBM-PINO)
- IBM German Bank Pool (GBP-PIN or GBP-PINO) - not supported on an IBM @server zSeries 990 and subsequent releases.
- VISA PIN validation value (VISA-PVV)
- Interbank PIN (INBK-PIN)

The callable service can execute only when ICSF is in special secure mode. This mode is described in "Special Secure Mode" on page 10.

For guidance information about VISA, see their appropriate publications. The Interbank PIN algorithm is available only on S/390 Enterprise Servers, the S/390 Multiprise, and the IBM @server Zseries.

Format

```
CALL CSNBPGN(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    PIN_generating_key_identifier,
    rule_array_count,
    rule_array,
    PIN_length,
    PIN_check_length,
    data_array,
    returned_result )
```
Clear PIN Generate (CSNBPGN)

Parameters

**return_code**
Direction: Output Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**
Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**
Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFFFF' (2 gigabytes). The data is defined in the exit_data parameter.

**exit_data**
Direction: Input/Output Type: String

The data that is passed to the installation exit.

**PIN_generating_key_identifier**
Direction: Input/Output Type: Character string

The 64-byte key label or internal key token that identifies the PIN generation (PINGEN) key. If the PIN_generating_key_identifier identifies a key which does not have the default PIN generation key control vector, the request will be routed to a PCI Cryptographic Coprocessor.

**rule_array_count**
Direction: Input Type: Integer

The number of process rules specified in the rule_array parameter. The value must be 1.

**rule_array**
Direction: Input Type: Character string

The process rule provides control information to the callable service. Specify one of the values in Table 112 on page 315. The keyword is left-justified in an 8-byte field, and padded on the right with blanks.
Table 112. Process Rules for the Clear PIN Generate Callable Service

<table>
<thead>
<tr>
<th>Process Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GBP-PIN</td>
<td>The IBM German Bank Pool PIN, which uses the institution PINGEN key to generate an institution PIN (IPIN).</td>
</tr>
<tr>
<td>GBP-PINO</td>
<td>The IBM German Bank Pool PIN offset, which uses the pool PINGEN key to generate a pool PIN (PPIN). It uses the institution PIN (IPIN) as input and calculates the PIN offset, which is the output. GBP-PINO is not supported on an IBM @server zSeries 990 and subsequent releases.</td>
</tr>
<tr>
<td>IBM-PIN</td>
<td>The IBM 3624 PIN, which is an institution-assigned PIN. It does not calculate the PIN offset.</td>
</tr>
<tr>
<td>IBM-PINO</td>
<td>The IBM 3624 PIN offset, which is a customer-selected PIN and calculates the PIN offset (the output).</td>
</tr>
<tr>
<td>INBK-PIN</td>
<td>The Interbank PIN is generated.</td>
</tr>
<tr>
<td>VISA-PVV</td>
<td>The VISA PIN validation value. Input is the customer PIN.</td>
</tr>
</tbody>
</table>

**PIN_length**

| Type: Integer |

The length of the PIN used for the IBM algorithms only, IBM-PIN or IBM-PINO. Otherwise, this parameter is ignored. Specify an integer from 4 through 16. If the length is greater than 12, the request will be routed to the PCI Cryptographic Coprocessor.

**PIN_check_length**

| Type: Integer |

The length of the PIN offset used for the IBM-PINO process rule only. Otherwise, this parameter is ignored. Specify an integer from 4 through 16.

**Note:** The PIN check length must be less than or equal to the integer specified in the **PIN_length** parameter.

**data_array**

| Type: String |

Three 16-byte data elements required by the corresponding rule_array parameter. The data array consists of three 16-byte fields or elements whose specification depends on the process rule. If a process rule only requires one or two 16-byte fields, then the rest of the data array is ignored by the callable service. Table 113 on page 316 describes the array elements.
Clear PIN Generate (CSNBPGN)

Table 113. Array Elements for the Clear PIN Generate Callable Service

<table>
<thead>
<tr>
<th>Array Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear_PIN</td>
<td>Clear user selected PIN of 4 to 12 digits of 0 through 9. Left-justified and padded with spaces. For IBM-PINO, this is the clear customer PIN (CSPIN). For GBP-PINO, this is the institution PIN. For IBM-PIN and GBP-PIN, this field is ignored.</td>
</tr>
<tr>
<td>Decimalization_table</td>
<td>Decimalization table for IBM and GBP only. Sixteen digits of 0 through 9.</td>
</tr>
<tr>
<td>Trans_sec_parm</td>
<td>For VISA only, the leftmost sixteen digits. Eleven digits of the personal account number (PAN). One digit key index. Four digits of customer selected PIN. For Interbank only, sixteen digits. Eleven right-most digits of the personal account number (PAN). A constant of 6. One digit key selector index. Three digits of PIN validation data.</td>
</tr>
<tr>
<td>Validation_data</td>
<td>Validation data for IBM and IBM German Bank Pool padded to 16 bytes. One to sixteen characters of hexadecimal account data left-justified and padded on the right with blanks.</td>
</tr>
</tbody>
</table>

Table 114 lists the data array elements required by the process rule (rule_array parameter). The numbers refer to the process rule’s position within the array.

Table 114. Array Elements Required by the Process Rule

<table>
<thead>
<tr>
<th>Process Rule</th>
<th>IBM-PIN</th>
<th>IBM-PINO</th>
<th>GBP-PIN</th>
<th>GBP-PINO</th>
<th>VISA-PVV</th>
<th>INBK-PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimalization_table</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Validation_data</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clear_PIN</td>
<td></td>
<td>3</td>
<td></td>
<td>3</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Trans_sec_parm</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Note: Generate offset for GBP algorithm is equivalent to IBM offset generation with PIN_check_length of 4 and PIN_length of 6.

returned_result

Direction: Output Type: Character string

The 16-byte generated output, left-justified and padded on the right with blanks.

Restrictions

PIN lengths of 13-16 require the optional PCI Cryptographic Coprocessor.

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

If you are using the IBM 3624 PIN and IBM German Bank Pool PIN algorithms, you can supply an unencrypted customer selected PIN to generate a PIN offset.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
Table 115. Clear PIN generate required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>ICSF routes this service to a PCI Cryptographic Coprocessor if the control vector of the PIN generating key cannot be processed on the Cryptographic Coprocessor Feature.</td>
</tr>
<tr>
<td>IBM zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM zSeries 990</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM zSeries 890</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Related Information

PIN algorithms are shown in PIN Formats and Algorithms.

Clear PIN Generate Alternate (CSNBCCPA)

Use the clear PIN generate alternate service to generate a clear VISA PVV (PIN validation value) from an input encrypted PIN block, or to produce a 3624 offset from a customer-selected encrypted PIN. The PIN block can be encrypted under either an input PIN-encrypting key (IPINENC) or an output PIN-encrypting key (OPINENC).

An enhanced PIN security mode, on PCICC, PCIXCC, CEX2C, or CEX3C, is available for extracting PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. To do this, you must enable the PTR Enhanced PIN Security access control point in the default role. When activated, this mode limits checking of the PIN to decimal digits and a PIN length minimum of 4 is enforced. No other PIN-block consistency checking will occur.
Clear PIN Generate Alternate (CSNBCPA)

Format

```
CALL CSNBCPA(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    PIN_encryption_key_identifier,
    PIN_generation_key_identifier,
    PIN_profile,
    PAN_data,
    encrypted_PIN_block,
    rule_array_count,
    rule_array,
    PIN_check_length,
    data_array,
    returned_PVV)
```

Parameters

**return_code**

Direction: Output Type: Integer


**reason_code**

Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. [Appendix A, "ICSF and TSS Return and Reason Codes"](https://www.ibm.com/support/knowledgecenter/en/SSLTBK_22.1.0/com.ibm.zos.v2r11.jes.doc/icsf/guide/icsf_g042.html) lists the reason codes.

**exit_data_length**

Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the `exit_data` parameter.

**exit_data**

Direction: Input/Output Type: String

The data that is passed to the installation exit.

**PIN_encryption_key_identifier**

Direction: Input/Output Type: String

A 64-byte string consisting of an internal token that contains an IPINENC or OPINENC key or the label of an IPINENC or OPINENC key that is used to encrypt the PIN block. If you specify a label, it must resolve uniquely to either an IPINENC or OPINENC key. If the `PIN_encryption_key_identifier` identifies a
Clear PIN Generate Alternate (CSNBCPA)

key which does not have the default PIN encrypting control vector (either IPINENC or OPINENC), the request will be routed to the PCI Cryptographic Coprocessor for processing.

**PIN_generation_key_identifier**

Direction: Input/Output  
Type: String

A 64-byte string that consists of an internal token that contains a PIN generation (PINGEN) key or the label of a PINGEN key. If the **PIN_generation_key_identifier** identifies a key which does not have the default PIN generating control vector, the request will be routed to the PCI Cryptographic Coprocessor for processing.

**PIN_profile**

Direction: Input  
Type: Character string

The three 8-byte character elements that contain information necessary to extract a PIN from a formatted PIN block. The pad digit is needed to extract the PIN from a 3624 or 3621 PIN block in the clear PIN generate alternate service. See "The PIN Profile" on page 305 for additional information.

**PAN_data**

Direction: Input  
Type: String

A 12-byte field that contains 12 characters of PAN data. The personal account number recovers the PIN from the PIN block if the PIN profile specifies ISO-0 or VISA-4 block formats. Otherwise it is ignored, but you must specify this parameter.

For ISO-0, use the rightmost 12 digits of the PAN, excluding the check digit. For VISA-4, use the leftmost 12 digits of the PAN, excluding the check digit.

**encrypted_PIN_block**

Direction: Input  
Type: String

An 8-byte field that contains the encrypted PIN that is input to the VISA PVV generation algorithm. The service uses the IPINENC or OPINENC key that is specified in the **PIN_encryption_key_identifier** parameter to encrypt the block.

**rule_array_count**

Direction: Input  
Type: Integer

The number of process rules specified in the **rule_array** parameter. The value may be 1 or 2. If the default extraction method for a PIN block format is desired, you may code the rule array count value as 1.

**rule_array**

Direction: Input  
Type: Character string

The process rule for the PIN generation algorithm. Specify IBM-PINO or “VISA-PVV” (the VISA PIN verification value) in an 8-byte field, left-justified, and padded with blanks. The **rule_array** points to an array of one or two 8-byte elements as follows:
Clear PIN Generate Alternate (CSNBCPA)

Table 116. Rule Array Elements for the Clear PIN Generate Alternate Service

<table>
<thead>
<tr>
<th>Rule Array Element</th>
<th>Function of Rule Array keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PIN calculation method</td>
</tr>
<tr>
<td>2</td>
<td>PIN extraction method</td>
</tr>
</tbody>
</table>

The first element in the rule array must specify one of the keywords that indicate the PIN calculation method as shown:

Table 117. Rule Array Keywords (First Element) for the Clear PIN Generate Alternate Service

<table>
<thead>
<tr>
<th>PIN Calculation Method Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM-PINO</td>
<td>This keyword specifies use of the IBM 3624 PIN Offset calculation method.</td>
</tr>
<tr>
<td>VISA-PVV</td>
<td>This keyword specifies use of the VISA PVV calculation method.</td>
</tr>
</tbody>
</table>

If the second element in the rule array is provided, one of the PIN extraction method keywords shown in Table 105 on page 306 may be specified for the given PIN block format. See "PIN Block Format and PIN Extraction Method Keywords" on page 305 for additional information. If the default extraction method for a PIN block format is desired, you may code the rule array count value as 1.

The PIN extraction methods operate as follows:

**PINBLOCK**
Specifies that the service use one of these:
- the PIN length, if the PIN block contains a PIN length field
- the PIN delimiter character, if the PIN block contains a PIN delimiter character.

**PADDIGIT**
Specifies that the service use the pad value in the PIN profile to identify the end of the PIN.

**HEXDIGIT**
Specifies that the service use the first occurrence of a digit in the range from X'A' to X'F' as the pad value to determine the PIN length.

**PINLENxx**
Specifies that the service use the length specified in the keyword, where xx can range from 4 to 16 digits, to identify the PIN.

**PADEXIST**
Specifies that the service use the character in the 16th position of the PIN block as the value of the pad value.

**PIN_check_length**

Direction: Input  Type: Integer

The length of the PIN offset used for the IBM-PINO process rule only. Otherwise, this parameter is ignored. Specify an integer from 4 through 16.
Clear PIN Generate Alternate (CSNBCPA)

**Note:** The PIN check length must be less than or equal to the integer specified in the `PIN_length` parameter. If the `PIN_check_length` variable is greater than the PIN length, the `PIN_check_length` variable will be set to the PIN length.

**data_array**

<table>
<thead>
<tr>
<th>Direction: Input</th>
<th>Type: String</th>
</tr>
</thead>
</table>

Three 16-byte elements. Table 118 describes the format when IBM-PINO is specified. Table 119 describes the format when VISA-PVV is specified.

*Table 118. Data Array Elements for the Clear PIN Generate Alternate Service (IBM-PINO)*

<table>
<thead>
<tr>
<th>Array Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>decimalization_table</td>
<td>This element contains the decimalization table of 16 characters (0 to 9) that are used to convert hexadecimal digits (X'0' to X'F') of the enciphered validation data to the decimal digits X'0' to X'9').</td>
</tr>
<tr>
<td>validation_data</td>
<td>This element contains one to 16 characters of account data. The data must be left justified and padded on the right with space characters.</td>
</tr>
<tr>
<td>Reserved-3</td>
<td>This field is ignored, but you must specify it.</td>
</tr>
</tbody>
</table>

When using the VISA-PVV keyword, identify these elements in the data array.

*Table 119. Data Array Elements for the Clear PIN Generate Alternate Service (VISA-PVV)*

<table>
<thead>
<tr>
<th>Array Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trans_sec_parm</td>
<td>For VISA-PVV only, the leftmost twelve digits. Eleven digits of the personal account number (PAN). One digit key index. The rest of the field is ignored.</td>
</tr>
<tr>
<td>Reserved-2</td>
<td>This field is ignored, but you must specify it.</td>
</tr>
<tr>
<td>Reserved-3</td>
<td>This field is ignored, but you must specify it.</td>
</tr>
</tbody>
</table>

**returned_PVV**

<table>
<thead>
<tr>
<th>Direction: Output</th>
<th>Type: Character</th>
</tr>
</thead>
</table>

A 16-byte area that contains the 4-byte PVV left-justified and padded with blanks.

**Restrictions**

The IBM-PINO PIN calculation method requires the optional PCICC, PCIXCC, CEX2C, or CEX3C.

On CCF systems, to use an IPINENC key, you must install the NOCV-enablement keys in the CKDS.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.
Use of the Visa-PVV PIN-calculation method will always output four digits rather than padding the output with binary zeros to the length of the PIN.

On CCF systems, to use an IPINENC key, you must install the NOCV-enablement keys in the CKDS.

This table lists the PIN block variant constants (PBVC) to use.

**Note:** PBVC is supported for compatibility with prior releases of OS/390 ICSF and existing ICSF applications. If PBVC is specified in the format control parameter of the PIN profile, the Clear PIN Generate Alternate service will not be routed to a PCI Cryptographic Coprocessor for processing. This means that only control vectors and extraction methods valid for the Cryptographic Coprocessor Feature may be used if PBVC formatting is desired. It is recommended that a format control of NONE be used for maximum flexibility.

**Restriction:** PBVC is supported only on an IBM zSeries 900 and 800.

### Table 120. PIN Block Variant Constants (PBVCs)

<table>
<thead>
<tr>
<th>PIN Format Name</th>
<th>PIN Block Variant Constant (PBVC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECI-2</td>
<td>X'00000000000093000000000009300'</td>
</tr>
<tr>
<td>ECI-3</td>
<td>X'00000000000095000000000009500'</td>
</tr>
<tr>
<td>ISO-0</td>
<td>X'0000000000008800000000008800'</td>
</tr>
<tr>
<td>ISO-1</td>
<td>X'00000000000008B000000008B000'</td>
</tr>
<tr>
<td>VISA-2</td>
<td>X'0000000000008D000000008D000'</td>
</tr>
<tr>
<td>VISA-3</td>
<td>X'0000000000008E000000008E000'</td>
</tr>
<tr>
<td>VISA-4</td>
<td>X'000000000000090000000009000'</td>
</tr>
<tr>
<td>3621</td>
<td>X'000000000000840000000084000'</td>
</tr>
<tr>
<td>3624</td>
<td>X'000000000000820000000082000'</td>
</tr>
<tr>
<td>4704-EPP</td>
<td>X'000000000000870000000087000'</td>
</tr>
</tbody>
</table>

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
## Clear PIN Generate Alternate (CSNBCPA)

Table 121. Clear PIN generate alternate required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>If PBVC is specified for format control, the request will be routed to a Cryptographic Coprocessor Feature.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>Cryptographic Coprocessor Feature</td>
<td>ICSF routes the request to a PCI Cryptographic Coprocessor if:</td>
</tr>
<tr>
<td></td>
<td>v The PIN_encryption_key_identifier identifies a key which does not have the default PIN encrypting control vector (either IPINENC or OPINENC).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>v IBM-PINO PIN calculation method is specified.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>v Anything is specified other than the default in the PIN extraction method keyword for the given PIN block format in rule_array.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>Format control in the PIN profile parameter must specify NONE.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td>Format control in the PIN profile parameter must specify NONE.</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Format control in the PIN profile parameter must specify NONE.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td>Format control in the PIN profile parameter must specify NONE.</td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

## Encrypted PIN Generate (CSNBEPG)

The Encrypted PIN Generate callable service formats a PIN and encrypts the PIN block. To generate the PIN, the service uses one of these PIN calculation methods:
- IBM 3624 PIN
- IBM German Bank Pool Institution PIN
- Interbank PIN

To format the PIN, the service uses one of these PIN block formats:
- IBM 3621 format
- IBM 3624 format
- ISO-0 format (same as the ANSI X9.8, VISA-1, and ECI-1 formats)
- ISO-1 format (same as the ECI-4 format)
- ISO-2 format
- ISO-3 format
- IBM 4704 encrypting PINPAD (4704-EPP) format
- VISA 2 format
- VISA 3 format
- VISA 4 format
- ECI-2 format
- ECI-3 format
Encrypted PIN Generate (CSNBEPG)

An enhanced PIN security mode, on PCICC, PCIXCC, CEX2C, and CEX3C, is available for formatting an encrypted PIN block into IBM 3621 format or IBM 3624 format. To do this, you must enable the PTR Enhanced PIN Security access control point in the default role. When activated, this mode limits checking of the PIN to decimal digits. No other PIN block consistency checking will occur.

Format

```
CALL CSNBEPG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    PIN_generating_key_identifier,
    outbound_PIN_encrypting_key_identifier
    rule_array_count,
    rule_array,
    PIN_length,
    data_array,
    PIN_profile,
    PAN_data,
    sequence_number
    encrypted_PIN_block )
```

Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the installation exit. The length can be from 'X'00000000' to 'X'7FFFFFFF' (2 gigabytes). The data is defined in the exit_data parameter.

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**PIN_generating_key_identifier**

Direction: Input/Output  
Type: String
Encrypted PIN Generate (CSNBEPG)

The 64-byte internal key token or a key label of an internal key token in the CKDS. The internal key token contains the PIN-generating key. The control vector must specify the PINGEN key type and have the EPINGEN usage bit set to 1.

**outbound_PIN_encrypting_key_identifier**

Direction: Input  
Type: String

A 64-byte internal key token or a key label of an internal key token in the CKDS. The internal key token contains the key to be used to encrypt the formatted PIN and must contain a control vector that specifies the OPINENC key type and has the EPINGEN usage bit set to 1.

**rule_array_count**

Direction: Input  
Type: Integer

The number of keywords you are supplying in the `rule_array` parameter. The value must be 1.

**rule_array**

Direction: Input  
Type: Character string

Keywords that provide control information to the callable service. Each keyword is left-justified in an 8-byte field, and padded on the right with blanks. All keywords must be in contiguous storage. The rule array keywords are shown as follows:

Table 122. Process Rules for the Encrypted PIN Generate Callable Service

<table>
<thead>
<tr>
<th>Process Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GBP-PIN</td>
<td>This keyword specifies the IBM German Bank Pool Institution PIN calculation method is to be used to generate a PIN.</td>
</tr>
<tr>
<td>IBM-PIN</td>
<td>This keyword specifies the IBM 3624 PIN calculation method is to be used to generate a PIN.</td>
</tr>
<tr>
<td>INBK-PIN</td>
<td>This keyword specifies the Interbank PIN calculation method is to be used to generate a PIN.</td>
</tr>
</tbody>
</table>

**PIN_length**

Direction: Input  
Type: Integer

A integer defining the PIN length for those PIN calculation methods with variable length PINs; otherwise, the variable should be set to zero.

**data_array**

Direction: Input  
Type: String

Three 16-byte character strings, which are equivalent to a single 48-byte string. The values in the data array depend on the keyword for the PIN calculation method. Each element is not always used, but you must always declare a complete data array. The numeric characters in each 16-byte string must be from 1 to 16 bytes in length, uppercase, left-justified, and padded on the right with space characters. Table 123 on page 326 describes the array elements.
Table 123. Array Elements for the Encrypted PIN Generate Callable Service

<table>
<thead>
<tr>
<th>Array Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimalization_table</td>
<td>Decimalization table for IBM and GBP only. Sixteen characters that are used to map the hexadecimal digits (X'0' to X'F') of the encrypted validation data to decimal digits (X'0' to X'9').</td>
</tr>
<tr>
<td>Trans_sec_parm</td>
<td>For Interbank only, sixteen digits. Eleven right-most digits of the personal account number (PAN). A constant of 6. One digit key selector index. Three digits of PIN validation data.</td>
</tr>
<tr>
<td>Validation_data</td>
<td>Validation data for IBM and IBM German Bank Pool padded to 16 bytes. One to sixteen characters of hexadecimal account data left-justified and padded on the right with blanks.</td>
</tr>
</tbody>
</table>

Table 124 lists the data array elements required by the process rule (rule_array parameter). The numbers refer to the process rule’s position within the array.

Table 124. Array Elements Required by the Process Rule

<table>
<thead>
<tr>
<th>Process Rule</th>
<th>IBM-PIN</th>
<th>GBP-PIN</th>
<th>INBK-PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimalization_table</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Validation_data</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Trans_sec_parm</td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

**PIN_profile**

**Direction:** Input  
**Type:** String array

A 24-byte string containing the PIN profile including the PIN block format. See "The PIN Profile" on page 305 for additional information.

**PAN_data**

**Direction:** Input  
**Type:** String

A 12-byte string that contains 12 digits of Personal Account Number (PAN) data. The service uses this parameter if the PIN profile specifies the ISO-0 or VISA-4 keyword for the PIN block format. Otherwise, ensure that this parameter is a 4-byte variable in application storage. The information in this variable will be ignored, but the variable must be specified.

**Note:** When using the ISO-0 keyword, use the 12 rightmost digit of the PAN data, excluding the check digit. When using the VISA-4 keyword, use the 12 leftmost digits of the PAN data, excluding the check digit.

**sequence_number**

**Direction:** Input  
**Type:** Integer

The 4-byte string that contains the sequence number used by certain PIN block formats. The service uses this parameter if the PIN profile specifies the 3621 or 4704-EPP keyword for the PIN block format. Otherwise, ensure that this
### Encrypted PIN Generate (CSNBEPG)

Parameter is a 4-byte variable in application data storage. The information in the variable will be ignored, but the variable must be declared. To enter a sequence number, do this:
- Enter 99999 to use a random sequence number that the service generates.
- For the 3621 PIN block format, enter a value in the range from 0 to 65535.
- For the 4704-EPP PIN block format, enter a value in the range from 0 to 255.

**encrypted_PIN_block**

Direction: Output  
Type: String

The field where the service returns the 8-byte encrypted PIN.

### Restrictions

The format control specified in the PIN profile must be NONE. If PBVC is specified as the format control, the service will fail.

### Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

SAF will be invoked to check authorization to use the Encrypted PIN Generate service and any key labels specified as input.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>PCI Cryptographic Coprocessor</td>
<td>ISO-3 PIN block format is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor, Crypto Express2 Coprocessor</td>
<td>ISO-3 PIN block format is not supported.</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>
Encrypted PIN Translate (CSNBPTR)

Encrypted PIN Translate (CSNBPTR)

Use the encrypted PIN translate callable service to reencipher a PIN block from one PIN-encrypting key to another and, optionally, to change the PIN block format, such as the pad digit or sequence number.

The unique-key-per-transaction key derivation for single and double-length keys is available for the encrypted PIN translate service. This support is available for the input_PIN_encrypting_key_identifier and the output_PIN_encrypting_key_identifier parameters for both REFORMAT and TRANSLAT process rules. The rule_array keyword determines which PIN key(s) are derived key(s).

The encrypted PIN translate service can be used for unique-key-per-transaction key derivation.

An enhanced PIN security mode, on PCICC, PCIXCC, CEX2C, and CEX3C, is available for formatting an encrypted PIN block into IBM 3621 format or IBM 3624 format. To do this, you must enable the PTR Enhanced PIN Security access control point in the default role. When activated, this mode limits checking of the PIN to decimal digits. No other PIN block consistency checking will occur.

The enhanced PIN security mode also extracts PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. You must enable the Enhanced PIN Security access control point in the default role. When activated, this mode limits checking of the PIN to decimal digits and a PIN length minimum of 4 is enforced. As with formatting an encrypted PIN block, no other PIN-block consistency checking will occur.

Format

```call csnbptr(  return_code,  reason_code,  exit_data_length,  exit_data,  input_PIN_encrypting_key_identifier,  output_PIN_encrypting_key_identifier,  input_PIN_profile,  PAN_data_in,  PIN_block_in,  rule_array_count,  rule_array,  output_PIN_profile,  PAN_data_out,  sequence_number,  PIN_block_out )```

Parameters

**return_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.
Encrypted PIN Translate (CSNBPTR)

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the `exit_data` parameter.

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**input_PIN_encrypting_key_identifier**

Direction: Input/Output  
Type: String

The input PIN-encrypting key (IPINENC) for the `PIN_block_in` parameter specified as a 64-byte internal key token or a key label. If keyword UKPTIPIN, UKPTBOTH, DUKPT-IP or DUKPT-BH is specified in the `rule_array`, then the `input_PIN_encrypting_key_identifier` must specify a key token or key label of a KEYGENKY with the UKPT usage bit enabled.

**output_PIN_encrypting_key_identifier**

Direction: Input/Output  
Type: String

The output PIN-encrypting key (OPINENC) for the `PIN_block_out` parameter specified as a 64-byte internal key token or a key label. If keyword UKPTOPIN, UKPTBOTH, DUKPT-OP or DUKPT-BH is specified in the `rule_array`, then the `output_PIN_encrypting_key_identifier` must specify a key token or key label of a KEYGENKY with the UKPT usage bit enabled.

**input_PIN_profile**

Direction: Input  
Type: Character string

The three 8-byte character elements that contain information necessary to either create a formatted PIN block or extract a PIN from a formatted PIN block. A particular PIN profile can be either an input PIN profile or an output PIN profile depending on whether the PIN block is being enciphered or deciphered by the callable service. See “The PIN Profile” on page 305 for additional information.

If you choose the TRANSLAT processing rule (this is not enforced on the PCIXCC, CEX2C, or CEX3C) in the `rule_array` parameter, the `input_PIN_profile` and the `output_PIN_profile` must specify the same PIN block format. If you choose the REFORMAT processing rule in the `rule_array` parameter, the input PIN profile and output PIN profile can have different PIN block formats. If you
specify UKPTPIN/DUKPT-IP or UKPTBOTH/DUKPT-BH in the rule_array parameter, then the input_PIN_profile is extended to a 48-byte field and must contain the current key serial number. See "The PIN Profile" on page 305 for additional information.

The pad digit is needed to extract the PIN from a 3624 or 3621 PIN block in the Encrypted PIN translate callable service with a process rule (rule_array parameter) of REFORMAT. If the process rule is TRANSLAT, the pad digit is ignored.

**PAN_data_in**

Direction: Input  
Type: Character string

The personal account number (PAN) if the process rule (rule_array parameter) is REFORMAT and the input PIN format is ISO-0 or VISA-4 only. Otherwise, this parameter is ignored. Specify 12 digits of account data in character format. For ISO-0, use the rightmost 12 digits of the PAN, excluding the check digit. For VISA-4, use the leftmost 12 digits of the PAN, excluding the check digit.

**PIN_block_in**

Direction: Input  
Type: String

The 8-byte enciphered PIN block that contains the PIN to be translated.

**rule_array_count**

Direction: Input  
Type: Integer

The number of process rules specified in the rule_array parameter. The value may be 1, 2 or 3.

**rule_array**

Direction: Input  
Type: Character string

The process rule for the callable service.

*Table 126. Keywords for Encrypted PIN Translate*

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processing Rules (required)</strong></td>
<td></td>
</tr>
<tr>
<td>REFORMAT</td>
<td>Changes the PIN format, the contents of the PIN block, and the PIN-encrypting key.</td>
</tr>
<tr>
<td>TRANSLAT</td>
<td>Changes the PIN-encrypting key only. It does not change the PIN format and the contents of the PIN block.</td>
</tr>
<tr>
<td><strong>PIN Block Format and PIN Extraction Method (optional)</strong></td>
<td>See &quot;PIN Block Format and PIN Extraction Method Keywords&quot; on page 305 for additional information and a list of PIN block formats and PIN extraction method keywords. Note: If a PIN extraction method is not specified, the first one listed in Table 105 on page 308 for the PIN block format will be the default.</td>
</tr>
</tbody>
</table>

**DUKPT Keywords - Single length key derivation (optional)**
### Encrypted PIN Translate (CSNBPTR)

#### Table 126. Keywords for Encrypted PIN Translate (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UKPTIPIN</strong></td>
<td>The <code>input_PIN_encrypting_key_identifier</code> is derived as a single length key. The <code>input_PIN_encrypting_key_identifier</code> must be a KEYGENKY key with the UKPT usage bit enabled. The <code>input_PIN_profile</code> must be 48 bytes and contain the key serial number.</td>
</tr>
<tr>
<td><strong>UKPTOPIN</strong></td>
<td>The <code>output_PIN_encrypting_key_identifier</code> is derived as a single length key. The <code>output_PIN_encrypting_key_identifier</code> must be a KEYGENKY key with the UKPT usage bit enabled. The <code>output_PIN_profile</code> must be 48 bytes and contain the key serial number.</td>
</tr>
<tr>
<td><strong>UKPTBOTH</strong></td>
<td>Both the <code>input_PIN_encrypting_key_identifier</code> and the <code>output_PIN_encrypting_key_identifier</code> are derived as a single length key. Both the <code>input_PIN_encrypting_key_identifier</code> and the <code>output_PIN_encrypting_key_identifier</code> must be KEYGENKY keys with the UKPT usage bit enabled. Both the <code>input_PIN_profile</code> and the <code>output_PIN_profile</code> must be 48 bytes and contain the respective key serial number.</td>
</tr>
</tbody>
</table>

**DUKPT Keywords - double length key derivation (optional) - requires May 2004 or later version of Licensed Internal Code (LIC)**

| **DUKPT-IP** | The `input_PIN_encrypting_key_identifier` is derived as a double length key. The `input_PIN_encrypting_key_identifier` must be a KEYGENKY key with the UKPT usage bit enabled. The `input_PIN_profile` must be 48 bytes and contain the key serial number. |
| **DUKPT-OP** | The `output_PIN_encrypting_key_identifier` is derived as a double length key. The `output_PIN_encrypting_key_identifier` must be a KEYGENKY key with the UKPT usage bit enabled. The `output_PIN_profile` must be 48 bytes and contain the key serial number. |
| **DUKPT-BH** | Both the `input_PIN_encrypting_key_identifier` and the `output_PIN_encrypting_key_identifier` are derived as a double length key. Both the `input_PIN_encrypting_key_identifier` and the `output_PIN_encrypting_key_identifier` must be KEYGENKY keys with the UKPT usage bit enabled. Both the `input_PIN_profile` and the `output_PIN_profile` must be 48 bytes and contain the respective key serial number. |

**output_PIN_profile**

**Direction:** Input  
**Type:** Character string

The three 8-byte character elements that contain information necessary to either create a formatted PIN block or extract a PIN from a formatted PIN block. A particular PIN profile can be either an input PIN profile or an output PIN profile, depending on whether the PIN block is being enciphered or deciphered by the callable service.
Encrypted PIN Translate (CSNBPTR)

- If you choose the TRANSLAT processing rule in the rule_array parameter, the input_PIN_profile and the output_PIN_profile must specify the same PIN block format, except on a PCIXCC, CEX2C, or CEX3C.
- If you choose the REFORMAT processing rule in the rule_array parameter, the input PIN profile and output PIN profile can have different PIN block formats.
- If you specify UKPTOPIN or UKPTBOTH in the rule_array parameter, then the output_PIN_profile is extended to a 48-byte field and must contain the current key serial number. See "The PIN Profile" on page 305 for additional information.
- If you specify DUKPT-OP or DUKPT-BH in the rule_array parameter, then the output_PIN_profile is extended to a 48-byte field and must contain the current key serial number. See "The PIN Profile" on page 305 for additional information.

**PAN_data_out**

Direction: Input  
Type: Character string

The personal account number (PAN) if the process rule (rule_array parameter) is REFORMAT and the output PIN format is ISO-0 or VISA-4 only. Otherwise, this parameter is ignored. Specify 12 digits of account data in character format.

For ISO-0, use the rightmost 12 digits of the PAN, excluding the check digit.

For VISA-4, use the leftmost 12 digits of the PAN, excluding the check digit.

**sequence_number**

Direction: Input  
Type: Integer

The sequence number if the process rule (rule_array parameter) is REFORMAT and the output PIN block format is 3621 or 4704-EPP only. Specify the integer value 99999. Otherwise, this parameter is ignored.

**PIN_block_out**

Direction: Output  
Type: String

The 8-byte output PIN block that is reenciphered.

**Restrictions**

Use of the ISO-2 PIN block format requires the optional PCICC, PCIXCC, CEX2C, or CEX3C.

Use of the UKPT keywords require the optional PCICC, PCIXCC, CEX2C, or CEX3C. Use of the DUKPT keywords require a PCIXCC, CEX2C, or CEX3C.

PAD digit restricted to non-decimal digit when Enhanced PIN Security access control point is enabled and if the output PIN profile specifies 3624 or 3621 as the PIN-block format.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.
PIN block formats are more rigorously validated on the IBM @server zSeries 990 and subsequent releases than on CCF systems.

Some PIN block formats are known by several names. This table shows the additional names.

<table>
<thead>
<tr>
<th>PIN Format</th>
<th>Additional Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO-0</td>
<td>ANSI X9.8, VISA format 1, ECI format 1</td>
</tr>
<tr>
<td>ISO-1</td>
<td>ECI format 4</td>
</tr>
</tbody>
</table>

This table lists the PIN block variant constants (PBVC) to be used.

**Note:** PBVC is NOT supported on the IBM @server zSeries 990 and subsequent releases. If PBVC is specified in the format control parameter of the PIN profile, the Encrypted PIN translate service will not be routed to a PCI Cryptographic Coprocessor for processing. This means that only control vectors and extraction methods valid for the Cryptographic Coprocessor Feature may be used if PBVC formatting is desired. It is recommended that a format control of NONE be used for maximum flexibility.

<table>
<thead>
<tr>
<th>PIN Format Name</th>
<th>PIN Block Variant Constant (PBVC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECI-2</td>
<td>X'00000000000009300000000000009300'</td>
</tr>
<tr>
<td>ECI-3</td>
<td>X'00000000000009500000000000009500'</td>
</tr>
<tr>
<td>ISO-0</td>
<td>X'00000000000008800000000000008800'</td>
</tr>
<tr>
<td>ISO-1</td>
<td>X'00000000000008B00000000000008B00'</td>
</tr>
<tr>
<td>VISA-2</td>
<td>X'00000000000008D0000000000008D00'</td>
</tr>
<tr>
<td>VISA-3</td>
<td>X'00000000000008E0000000000008E00'</td>
</tr>
<tr>
<td>VISA-4</td>
<td>X'0000000000000900000000000009000'</td>
</tr>
<tr>
<td>3621</td>
<td>X'0000000000000840000000000008400'</td>
</tr>
<tr>
<td>3624</td>
<td>X'0000000000000820000000000008200'</td>
</tr>
<tr>
<td>4704-EPP</td>
<td>X'0000000000000870000000000008700'</td>
</tr>
</tbody>
</table>

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
### Encrypted PIN Translate (CSNBPTR)

#### Table 129. Encrypted PIN translate required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>If PBVC is specified for format control, the request will be routed to the Cryptographic Coprocessor Feature. ISO-3 PIN block format is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>PCI Cryptographic Coprocessor</td>
<td>ICSF routes this service to a PCI Cryptographic Coprocessor if: &lt;br&gt; - The control vector in a supplied PIN encrypting key cannot be processed on the Cryptographic Coprocessor Feature. &lt;br&gt; - UKPT support is requested. &lt;br&gt; - The PIN profile specifies the ISO-2 PIN block format. &lt;br&gt; - if the <code>input_PIN_encrypting_key_identifier</code> identifies a key which does not have the default input PIN encrypting key control vector (IPINENC) &lt;br&gt; - if the <code>output_PIN_encrypting_key_identifier</code> identifies a key which does not have the default output PIN encrypting key control vector (OPINENC) &lt;br&gt; - if anything is specified other than the default in the PIN extraction method keyword for the given PIN block format in <code>rule_array</code>&lt;br&gt; DUKPT-IP, DUKPT-OP and DUKPT-BH keywords are not supported. ISO-3 PIN block format is not supported.</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Format control in the PIN profile parameter must specify NONE. ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Format control in the PIN profile parameter must specify NONE. ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Format control in the PIN profile parameter must specify NONE. ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td>Format control in the PIN profile parameter must specify NONE. ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
</tbody>
</table>
Encrypted PIN Verify (CSNBPVR)

Encrypted PIN Verify (CSNBPVR)

Use the Encrypted PIN verify callable service to verify that one of these customer selected trial PINs is valid:

- IBM 3624 (IBM-PIN)
- IBM 3624 PIN offset (IBM-PINO)
- IBM German Bank Pool (GBP-PIN)
- IBM German Bank Pool PIN offset (GBP-PINO) - not supported on the IBM @server zSeries 990
- VISA PIN validation value (VISA-PVV)
- VISA PIN validation value (VISAPVV4)
- Interbank PIN (INBK-PIN)

The unique-key-par-transaction key derivation for single and double-length keys is available for the input_PIN_encrypting_key_identifier parameter.

An enhanced PIN security mode, on PCICC, PCIXCC, CEX2C, and CEX3C, is available for extracting PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. To do this, you must enable the PTR Enhanced PIN Security access control point in the default role. When activated, this mode limits checking of the PIN to decimal digits and a PIN length minimum of 4 is enforced. No other PIN-block consistency checking will occur.

Format

```
CALL CSNBPVR(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    input_PIN_encrypting_key_identifier,
    PIN_verifying_key_identifier,
    input_PIN_profile,
    PAN_data,
    encrypted_PIN_block,
    rule_array_count,
    rule_array,
    PIN_check_length,
    data_array )
```

Parameters

**return_code**

Direction: Output Type: Integer

The return code specifies the general result of the callable service. [Appendix A, "ICSF and TSS Return and Reason Codes"] lists the return codes.

**reason_code**

Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. [Appendix A, "ICSF and TSS Return and Reason Codes"] lists the reason codes.
Encrypted PIN Verify (CSNEBPVR)

exit_data_length
Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from 'X'00000000' to 'X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

exit_data
Direction: Input/Output Type: String

The data that is passed to the installation exit.

input_PIN_encrypting_key_identifier
Direction: Input/Output Type: String

The 64-byte key label or internal key token containing the PIN-encrypting key (IPINENC) that enciphers the PIN block. If keyword UKPTIPIN or DUKPT-IP is specified in the rule_array, then the input_PIN_encrypting_key_identifier must specify a key token or key label of a KEYGENKY with the UKPT usage bit enabled.

PIN_verifying_key_identifier
Direction: Input/Output Type: String

The 64-byte key label or internal key token that identifies the PIN verify (PINVER) key.

input_PIN_profile
Direction: Input Type: Character string

The three 8-byte character elements that contain information necessary to either create a formatted PIN block or extract a PIN from a formatted PIN block. A particular PIN profile can be either an input PIN profile or an output PIN profile depending on whether the PIN block is being enciphered or deciphered by the callable service. If you specify UKPTIPIN in the rule_array parameter, then the input_PIN_profile is extended to a 48-byte field and must contain the current key serial number. See "The PIN Profile" on page 305 for additional information.

If you specify DUKPT-IP in the rule_array parameter, then the input_PIN_profile is extended to a 48-byte field and must contain the current key serial number. See "The PIN Profile" on page 305 for additional information.

The pad digit is needed to extract the PIN from a 3624 or 3621 PIN block in the encrypted PIN verify callable service.

PAN_data
Direction: Input Type: Character string

The personal account number (PAN) is required for ISO-0 and VISA-4 only. Otherwise, this parameter is ignored. Specify 12 digits of account data in character format.
Encrypted PIN Verify (CSNBPVVR)

For ISO-0, use the rightmost 12 digits of the PAN, excluding the check digit.
For VISA-4, use the leftmost 12 digits of the PAN, excluding the check digit.

encrypted_PIN_block
Direction: Input Type: String
The 8-byte enciphered PIN block that contains the PIN to be verified.

rule_array_count
Direction: Input Type: Integer
The number of process rules specified in the rule_array parameter. The value may be 1, 2 or 3.

rule_array
Direction: Input Type: Character string
The process rule for the PIN verify algorithm.

Table 130. Keywords for Encrypted PIN Verify

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithm Value (required)</strong></td>
<td></td>
</tr>
<tr>
<td>GBP-PIN</td>
<td>The IBM German Bank Pool PIN. It verifies the PIN entered by the customer and compares that PIN with the institution generated PIN by using an institution key.</td>
</tr>
<tr>
<td>GBP-PINO</td>
<td>The IBM German Bank Pool PIN offset. It verifies the PIN entered by the customer by comparing with the calculated institution PIN (IPIN) and adding the specified offset to the pool PIN (PPIN) generated by using a pool key. GBP-PINO is not supported on the IBM @server zSeries 990.</td>
</tr>
<tr>
<td>IBM-PIN</td>
<td>The IBM 3624 PIN, which is an institution-assigned PIN. It does not calculate the PIN offset.</td>
</tr>
<tr>
<td>IBM-PINO</td>
<td>The IBM 3624 PIN offset, which is a customer-selected PIN and calculates the PIN offset.</td>
</tr>
<tr>
<td>INBK-PIN</td>
<td>The Interbank PIN verify algorithm.</td>
</tr>
<tr>
<td>VISA-PVV</td>
<td>The VISA PIN verify value.</td>
</tr>
<tr>
<td>VISAPVV4</td>
<td>The VISA PIN verify value. If the length is 4 digits, normal processing for VISA-PVV will occur. The VISAPVV4 requires a PCICC, PCIXCC, CEX2C, or CEX3C. If one is not available, the service will fail. If the length is greater than 4 digits, the service will fail.</td>
</tr>
<tr>
<td><strong>PIN Block Format and PIN Extraction Method (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>See &quot;PIN Block Format and PIN Extraction Method Keywords&quot; on page 305 for additional information and a list of PIN block formats and PIN extraction method keywords. Note: If a PIN extraction method is not specified, the first one listed in Table 105 on page 306 for the PIN block format will be the default.</td>
<td></td>
</tr>
</tbody>
</table>

DUKPT Keyword - Single length key derivation (optional)
Table 130. Keywords for Encrypted PIN Verify (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>UKPTIPIN</td>
<td>The input_PIN_encrypting_key_identifier is derived as a single length key. The input_PIN_encrypting_key_identifier must be a KEYGENKY key with the UKPT usage bit enabled. The input_PIN_profile must be 48 bytes and contain the key serial number.</td>
</tr>
</tbody>
</table>

**DUKPT Keyword - double length key derivation (optional) - requires Requires May 2004 or later version of Licensed Internal Code (LIC)**

| DUKPT-IP   | The input_PIN_encrypting_key_identifier is to be derived using the DUKPT algorithm. The input_PIN_encrypting_key_identifier must be a KEYGENKY key with the DUKPT usage bit enabled. The input_PIN_profile must be 48 bytes and contain the key serial number. |

**PIN_check_length**

| Type: Integer |

The PIN check length for the IBM-PIN or IBM-PINO process rules only. Otherwise, it is ignored. Specify the rightmost digits, 4 through 16, for the PIN to be verified.

**data_array**

| Type: String |

Three 16-byte elements required by the corresponding rule_array parameter. The data array consists of three 16-byte fields whose specification depend on the process rule. If a process rule only requires one or two 16-byte fields, then the rest of the data array is ignored by the callable service. Table 131 describes the array elements.

Table 131. Array Elements for the Encrypted PIN Verify Callable Service

<table>
<thead>
<tr>
<th>Array Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimalization_table</td>
<td>Decimalization table for IBM and GBP only. Sixteen decimal digits of 0 through 9.</td>
</tr>
<tr>
<td>PIN_offset</td>
<td>Offset data for IBM-PINO and GBP-PINO. One to twelve numeric characters, 0 through 9, left-justified and padded on the right with blanks. For IBM-PINO, the PIN offset length is specified in the PIN_check_length parameter. For GBP-PINO, the PIN offset is always 4 digits. For IBM-PIN and GBP-PIN, the field is ignored.</td>
</tr>
<tr>
<td>trans_sec_parm</td>
<td>For VISA, only the leftmost twelve digits of the 16-byte field are used. These consist of the rightmost eleven digits of the personal account number (PAN) and a one-digit key index. The remaining four characters are ignored. For Interbank only, all 16 bytes are used. These consist of the rightmost eleven digits of the PAN, a constant of X'6', a one-digit key index, and three numeric digits of PIN validation data.</td>
</tr>
</tbody>
</table>
Table 131. Array Elements for the Encrypted PIN Verify Callable Service (continued)

<table>
<thead>
<tr>
<th>Array Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPVV</td>
<td>For VISA-PVV only, referenced PVV (4 bytes) that is left-justified. The rest of the field is ignored.</td>
</tr>
<tr>
<td>Validation_data</td>
<td>Validation data for IBM and GBP padded to 16 bytes. One to sixteen characters of hexadecimal account data left-justified and padded on the right with blanks.</td>
</tr>
</tbody>
</table>

Table 132 lists the data array elements required by the process rule (rule_array parameter). The numbers refer to the process rule's position within the array.

Table 132. Array Elements Required by the Process Rule

<table>
<thead>
<tr>
<th>Process Rule</th>
<th>IBM-PIN</th>
<th>IBM-PINO</th>
<th>GBP-PIN</th>
<th>GBP-PINO</th>
<th>VISA-PVV</th>
<th>INBK-PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimalization_table</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Validation_data</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIN_offset</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trans_sec_parm</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RPVV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**Restrictions**

GBP-PINO is only supported if the CSNBPVVR service is processed on the Cryptographic Coprocessor Feature. If the service is routed to a PCI Cryptographic Coprocessor, the service request will fail if the GBP-PINO calculation method is specified. GBP-PINO is not supported on the IBM @server zSeries 990, IBM @server zSeries 890, z9 EC or z9 BC.

- Use of the ISO-2 PIN block format requires the optional PCICC, PCIXCC, CEX2C, or CEX3C.
- Use of the UKPTIPIN keyword requires the optional PCICC, PCIXCC, CEX2C, or CEX3C.
- Use of the VISAPVV4 keyword requires the optional PCICC, PCIXCC, CEX2C, or CEX3C.
- Use of the DUKPT-IP keyword requires a PCIXCC, CEX2C, or CEX3C.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

PIN block formats are more rigorously validated on the IBM @server zSeries 990 than on CCF systems.

This table lists the PIN block variant constants (PBVC) to be used.

**Restriction:** PBVC is not supported on an IBM @server zSeries 990. If PBVC is specified in the format control parameter of the PIN profile, the Encrypted PIN Verify service will not be routed to a PCI Cryptographic Coprocessor for processing. This means that only control vectors and extraction methods valid for the Cryptographic Coprocessor Feature...
Encrypted PIN Verify (CSNBPRV)

may be used if PBVC formatting is desired. It is recommended that a format control of NONE be used for maximum flexibility.

Table 133. PIN Block Variant Constants (PBVCs)

<table>
<thead>
<tr>
<th>PIN Format Name</th>
<th>PIN Block Variant Constant (PBVC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECI-2</td>
<td>X'000000000000930000000009300'</td>
</tr>
<tr>
<td>ECI-3</td>
<td>X'000000000000950000000009500'</td>
</tr>
<tr>
<td>ISO-0</td>
<td>X'000000000000088000000008800'</td>
</tr>
<tr>
<td>ISO-1</td>
<td>X'00000000000008B0000000000008B00'</td>
</tr>
<tr>
<td>VISA-2</td>
<td>X'00000000000008D00000000008D00'</td>
</tr>
<tr>
<td>VISA-3</td>
<td>X'00000000000008E00000000008E00'</td>
</tr>
<tr>
<td>VISA-4</td>
<td>X'000000000000090000000009000'</td>
</tr>
<tr>
<td>3621</td>
<td>X'00000000000084000000084000'</td>
</tr>
<tr>
<td>3624</td>
<td>X'000000000000820000000082000'</td>
</tr>
<tr>
<td>4704-EPP</td>
<td>X'000000000000087000000008700'</td>
</tr>
</tbody>
</table>

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 134. Encrypted PIN verify required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>If PBVC is specified for format control, the request will be routed to the Cryptographic Coprocessor Feature. ISO-3 PIN block format is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>PCI Cryptographic Coprocessor</td>
<td>ICSF routes the request to a PCI Cryptographic Coprocessor if: • The PIN profile specifies the ISO-2 PIN block format. • Anything is specified other than the default in the PIN extraction method keyword for the given PIN block format in rule_array. • The input_PIN_encrypting_key_identifier identifies a key which does not have the default PIN encrypting key control vector (IPINENC). • The PIN_verifying_key_identifier identifies a key which does not have the default PIN verify key control vector. • The VISAPVV4 rule array keyword is specified. • You request UKPT support. The DUKPT-IP keyword is not supported. ISO-3 PIN block format is not supported.</td>
</tr>
</tbody>
</table>
Table 134. Encrypted PIN verify required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>Format control in the PIN profile parameter must specify NONE. GBP-PINO rule array parameter is not supported. ISO-3 PIN block format is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td>ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Format control in the PIN profile parameter must specify NONE. GBP-PINO rule array parameter is not supported. ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td>ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Format control in the PIN profile parameter must specify NONE. GBP-PINO rule array parameter is not supported. ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td>ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
</tbody>
</table>

Related Information

PIN Formats and Algorithms discusses the PIN algorithms in detail.

PIN Change/Unblock (CSNBPCU)

The PIN Change/Unblock callable service is used to generate a special PIN block to change the PIN accepted by an integrated circuit card (smartcard). The special PIN block is based on the new PIN and the card-specific diversified key and, optionally, on the current PIN of the smartcard. The new PIN block is encrypted with a session key. The session key is derived in a two-step process. First, the card-specific diversified key (ICC Master Key) is derived using the TDES-ENC algorithm of the diversified key generation callable service. The session key is then generated according to the rule array algorithm:

- TDES-XOR - XOR ICC Master Key with the Application Transaction Counter (ATC)
- TDESEMV2 - use the EMV2000 algorithm with a branch factor of 2
- TDESEMV4 - use the EMV2000 algorithm with a branch factor of 4

The generating DKYGENKY cannot have replicated halves. The encryption_issuer_master_key_identifier is a DKYGENKY that permits generation of a SMPIN key. The authentication_issuer_master_key_identifier is also a DKYGENKY that permits generation of a double length MAC key.

The PIN block format is specified by the VISA ICC Card specification: two mutually exclusive rule array keywords, VISAPCU1 and VISAPCU2. They refer to whether the current PIN is used in the generation of the new PIN. For VISAPCU1, it is not used, for VISAPCU2 it is used.

An enhanced PIN security mode, on PCICC, PCIXCC, CEX2C, or CEX3C is available for extracting PINs from encrypted PIN blocks. This mode only applies when specifying a PIN-extraction method for an IBM 3621 or an IBM 3624 PIN-block. To do this, you must enable the PTR Enhanced PIN Security access.
control point in the default role. When activated, this mode limits checking of the PIN to decimal digits and a PIN length minimum of 4 is enforced. No other PIN-block consistency checking will occur.

### Format

```c
CALL CSNBPCU(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    authentication_issuer_master_key_length,
    authentication_issuer_master_key_identifier,
    encryption_issuer_master_key_length,
    encryption_issuer_master_key_identifier,
    key_generation_data_length,
    key_generation_data,
    new_reference_PIN_key_length,
    new_reference_PIN_key_identifier,
    new_reference_PIN_block,
    new_reference_PIN_profile,
    new_reference_PIN_PAN_data,
    current_reference_PIN_key_length,
    current_reference_PIN_key_identifier,
    current_reference_PIN_block,
    current_reference_PIN_profile,
    current_reference_PIN_PAN_data,
    output_PIN_data_length,
    output_PIN_data,
    output_PIN_profile,
    output_PIN_message_length,
    output_PIN_message )
```

### Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the `exit_data` parameter.
**PIN Change/Unblock (CSNBPCU)**

**exit_data**
Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**rule_array_count**
Direction: Input  
Type: Integer

The number of keywords you are supplying in the `rule_array` parameter. The valid values are 1 and 2.

**rule_array**
Direction: Input  
Type: String

Keywords that provides control information to the callable service. The keywords are left-justified in an 8-byte field and padded on the right with blanks. The keywords must be in contiguous storage. Specify one or two of these options:

**Table 135. Rule Array Keywords for PIN Change/Unblock**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithm (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>TDES-XOR</td>
<td>TDES encipher clear data to generate the intermediate (card-unique) key, followed by XOR of the final 2 bytes of each key with the ATC counter. This is the default.</td>
</tr>
<tr>
<td>TDESEMV2</td>
<td>Same processing as in the diversified key generate service.</td>
</tr>
<tr>
<td>TDESEMV4</td>
<td>Same processing as in the diversified key generate service.</td>
</tr>
<tr>
<td><strong>PIN processing method (required)</strong></td>
<td></td>
</tr>
<tr>
<td>VISAPCU1</td>
<td>Form the new PIN from the new reference PIN and the smart-card-unique, intermediate key.</td>
</tr>
<tr>
<td>VISAPCU2</td>
<td>Form the new PIN from the new reference PIN and the smart-card-unique, the intermediate (card-unique) key and the current reference PIN.</td>
</tr>
</tbody>
</table>

**authentication_issuer_master_key_length**
Direction: Input  
Type: Integer

The length of the `authentication_issuer_master_key_identifier` parameter. Currently, the value must be 64.

**authentication_issuer_master_key_identifier**
Direction: Input/Output  
Type: String

The label name or internal token of a DKYGENKY key type that is to be used to generate the card-unique diversified key. The control vector of this key must be a DKYL0 key that permits the generation of a double-length MAC key (DMAC). This DKYGENKY may not have replicated key halves.
PIN Change/Unblock (CSNBPCU)

**encryption_issuer_master_key_length**

Direction: Input  
Type: Integer

The length of the `encryption_issuer_master_key_identifier` parameter. Currently, the value must be 64.

**encryption_issuer_master_key_identifier**

Direction: Input/Output  
Type: String

The label name or internal token of a DKYGENKY key type that is to be used to generate the card-unique diversified key and the secure messaging session key for the protection of the output PIN block. The control vector of this key must be a DKYLO0 key that permits the generation of a DMPIN key type. This DKYGENKY may not have replicated key halves.

**key_generation_data_length**

Direction: Input  
Type: Integer

The length of the `key_generation_data` parameter. This value must be 10, 18, 26 or 34 bytes.

**key_generation_data**

Direction: Input  
Type: String

The data provided to generate the card-unique session key. For TDES-XOR, this consists of 8 or 16 bytes of data to be processed by TDES to generate the card-unique diversified key followed by a 16 bit ATC counter to offset the card-unique diversified key to form the session key. For TDESEMV2 and TDESEMV4, this may be 10, 18, 26 or 34 bytes. See "Diversified Key Generate (CSNBDDK3)" on page 102 for more information.

**new_reference_PIN_key_length**

Direction: Input  
Type: Integer

The length of the `new_reference_PIN_key_identifier` parameter. Currently, the value must be 64.

**new_reference_PIN_key_identifier**

Direction: Input/Output  
Type: String

The label name or internal token of a PIN encrypting key that is to be used to decrypt the `new_reference_PIN_block`. This must be an IPINENC or OPINENC key. If the label name is supplied, the name must be unique in the CKDS.

**new_reference_PIN_block**

Direction: Input  
Type: String

This is an 8-byte field that contains the enciphered PIN block of the new PIN.

**new_reference_PIN_profile**

Direction: Input  
Type: String
PIN Change/Unblock (CSNBPCU)

This is a 24-byte field that contains three 8-byte elements with a PIN block format keyword, a format control keyword (NONE) and a pad digit as required by certain formats.

new_reference_PIN_PAN_data
Direction: Input  Type: String

This is a 12-byte field containing PAN in character format. This data may be needed to recover the new reference PIN if the format is ISO-0 or VISA-4. If neither is used, this parameter may be blanks.

current_reference_PIN_key_length
Direction: Input  Type: Integer

The length of the current_reference_PIN_key_identifier parameter. For the current implementation, the value must be 64. If the rule_array contains VISAPCU1, this value must be 0.

current_reference_PIN_key_identifier
Direction: Input/Output  Type: String

The label name or internal token of a PIN encrypting key that is to be used to decrypt the current_reference_PIN_block. This must be an IPINENC or OPINENC key. If the labelname is supplied, the name must be unique on the CKDS. If the rule_array contains VISAPCU1, this value is ignored.

current_reference_PIN_block
Direction: Input  Type: String

This is an 8-byte field that contains the enciphered PIN block of the new PIN. If the rule_array contains VISAPCU1, this value is ignored.

current_reference_PIN_profile
Direction: Input  Type: String

This is a 24-byte field that contains three 8-byte elements with a PIN block format keyword, a format control keyword (NONE) and a pad digit as required by certain formats. If the rule_array contains VISAPCU1, this value is ignored.

current_reference_PIN_PAN_data
Direction: Input  Type: String

This is a 12-byte field containing PAN in character format. This data may be needed to recover the new reference PIN if the format is ISO-0 or VISA-4. If neither is used, this parameter may be blanks. If the rule_array contains VISAPCU1, this value is ignored.

output_PIN_data_length
Direction: Input  Type: Integer
PIN Change/Unblock (CSNBPDCU)

Currently this field is reserved. The value of this parameter should be 0.

**output_PIN_data**

Direction: Input  Type: String

Currently this field is reserved.

**output_PIN_profile**

Direction: Input  Type: String

This is a 24-byte field that contains three 8-byte elements with a PIN block format keyword (VISAPCU1 or VISAPCU2), a format control keyword, NONE, (left aligned and padded on the right with space characters) and 8 byte spaces.

**output_PIN_message_length**

Direction: Input/Output  Type: Integer

The length of the output_PIN_message field. Currently the value must be at least 16.

**output_PIN_message**

Direction: Output  Type: String

The reformatted PIN block with the new reference PIN enciphered under the SMPIN session key.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

There are additional access points for this service.

RACF will be invoked to check authorization to use the PIN change/unblock service and any labelname specified.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

**Table 136. PIN Change/Unblock hardware**

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td></td>
<td>Not supported</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>ISO-3 PIN block format is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>
PIN Change/Unblock (CSNBPCU)

Table 136. PIN Change/Unblock hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM System z9 EC and z9 BC</td>
<td>Crypto Express2 Coprocessor</td>
<td>ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express2 Coprocessor</td>
<td>ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td>ISO-3 PIN block format requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
</tbody>
</table>

Secure Messaging for Keys (CSNBSKY)

The Secure Messaging for Keys callable service will encrypt a text block including a clear key value decrypted from an internal or external DES token. The text block is normally a "Value" field of a secure message TLV (Tag/Length/Value) element of a secure message. TLV is defined in ISO/IEC 7816-4.

Format

```call
CALL CSNBSKY(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    input_key_identifier,
    key_encrypting_key_identifier,
    secmsg_key_identifier,
    text_length,
    clear_text,
    initialization_vector,
    key_offset,
    key_offset_field_length,
    enciphered_text,
    output_chaining_vector )
```

Parameters

`return_code`

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

`reason_code`

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.
Secure Messaging for Keys (CSNBSKY)

**exit_data_length**
Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**
Direction: Input/Output  Type: String

The data that is passed to the installation exit.

**rule_array_count**
Direction: Input  Type: Integer

The number of keywords you are supplying in the rule_array parameter. The valid values are 0 and 1.

**rule_array**
Direction: Input  Type: Character String

Keywords that provides control information to the callable service. The processing method is the encryption mode used to encrypt the message.

*Table 137. Rule Array Keywords for Secure Messaging for Keys*

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enciphering mode (optional)</td>
<td></td>
</tr>
<tr>
<td>TDES-CBC</td>
<td>Use CBC mode to encipher the message (default).</td>
</tr>
<tr>
<td>TDES-ECB</td>
<td>Use EBC mode to encipher the message.</td>
</tr>
</tbody>
</table>

**input_key_identifier**
Direction: Input/Output  Type: String

The internal token, external token, or key label of an internal token of a double length DES key. The key is recovered in the clear and placed in the text to be encrypted. The control vector of the DES key must not prohibit export.

**key_encrypting_key_identifier**
Direction: Input/Output  Type: String

If the input_key_identifier is an external token, then this parameter is the internal token or the key label of the internal token of IMPORTER or EXPORTER. If it is not, it is a null token. If a key label is specified, the key label must be unique.

**secmsg_key_identifier**
Direction: Input/Output  Type: String
Secure Messaging for Keys (CSNBSKY)

The internal token or key label of a secure message key for encrypting keys. This key is used to encrypt the updated clear_text containing the recovered DES key.

text_length
Direction: Input Type: Integer

The length of the clear_text parameter that follows. Length must be a multiple of eight. Maximum length is 4K.

clear_text
Direction: Input Type: String

Clear text that contains the recovered DES key at the offset specified and is then encrypted. Any padding or formatting of the message must be done by the caller on input.

initialization_vector
Direction: Input Type: String

The 8-byte supplied string for the TDES-CBC mode of encryption. The initialization_vector is XORed with the first 8 bytes of clear_text prior to encryption. This field is ignored for TDES-ECB mode.

key_offset
Direction: Input Type: Integer

The offset within the clear_text parameter at key_offset where the recovered clear input_key_identifier value is to be placed. The first byte of the clear_text field is offset 0.

key_offset_field_length
Direction: Input Type: Integer

The length of the field within clear_text parameter at key_offset where the recovered clear input_key_identifier value is to be placed. Length must be a multiple of eight and is equal to the key length of the recovered key. The key must fit entirely within the clear_text.

enciphered_text
Direction: Output Type: String

The field where the enciphered text is returned. The length of this field must be at least as long as the clear_text field.

output_chaining_vector
Direction: Output Type: String

This field contains the last 8 bytes of enciphered text and is used as the initialization_vector for the next encryption call if data needs to be chained for TDES-CBC mode. No data is returned for TDES-ECB.
Secure Messaging for Keys (CSNBSKY)

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

SAF will be invoked to check authorization to use the secure messaging for keys service and any key labels specified as input.

Keys only appear in the clear within the secure boundary of the cryptographic coprocessor and never in host storage.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 138. Secure messaging for keys required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>PCI Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

Secure Messaging for PINs (CSNBSPN)

The Secure Messaging for PINs callable service will encrypt a text block including a clear PIN block recovered from an encrypted PIN block. The input PIN block will be reformatted if the block format in the input_PIN_profile is different than the block format in the output_PIN_profile. The clear PIN block will only be self encrypted if the SELFENC keyword is specified in the rule_array. The text block is normally a 'Value' field of a secure message TLV (Tag/Length/Value) element of a secure message. TLV is defined in ISO/IEC 7816-4.
Secure Messaging for PINs (CSNBSN)

Format

CALL CSNBSN(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    input_PIN_block,
    PIN_encrypting_key_identifier,
    input_PIN_profile,
    input_PAN_data,
    secmsg_key_identifier,
    output_PIN_profile,
    output_PAN_data,
    text_length,
    clear_text,
    initialization_vector,
    PIN_offset,
    PIN_offset_field_length,
    enciphered_text,
    output_chaining_vector
)

Parameters

**return_code**

Direction: Output  Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

Direction: Output  Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**

Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**

Direction: Input/Output  Type: String

The data that is passed to the installation exit.

**rule_array_count**

Direction: Input  Type: Integer
Secure Messaging for PINs (CSNBSPI)

The number of keywords you are supplying in the rule_array parameter. The valid values are 0, 1, or 2.

**rule_array**

Direction: Input Type: Character String

Keywords that provide control information to the callable service. The processing method is the algorithm used to create the generated key. The keywords are left justified and padded on the right with blanks.

Table 139. Rule Array Keywords for Secure Messaging for PINs

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enciphering mode (optional)</td>
<td></td>
</tr>
<tr>
<td>TDES-CBC</td>
<td>Use CBC mode to encipher the message (default).</td>
</tr>
<tr>
<td>TDES-ECB</td>
<td>Use EBC mode to encipher the message.</td>
</tr>
<tr>
<td>PIN encryption (optional)</td>
<td></td>
</tr>
<tr>
<td>CLEARPIN</td>
<td>Recovered clear input PIN block (may be reformatted) is placed in the clear in the message for encryption with the secure message key (default).</td>
</tr>
<tr>
<td>SELFENC</td>
<td>Recovered clear input PIN block (may be reformatted) is self-encrypted and then placed in the message for encryption with the secure message key.</td>
</tr>
</tbody>
</table>

**input_PIN_block**

Direction: Input Type: String

The 8-byte input PIN block that is to be recovered in the clear and perhaps reformatted, and then placed in the clear_text to be encrypted.

**PIN_encrypting_key_identifier**

Direction: Input/Output Type: String

The internal token or key label of the internal token of the PIN encrypting key used in encrypting the input_PIN_block. The key must be an IPINENC key.

**input_PIN_profile**

Direction: Input Type: Character String

The three 8-byte character elements that contain information necessary to extract the PIN from a formatted PIN block. The valid input PIN formats are ISO-0, ISO-1, ISO-2 and ISO-3. See “The PIN Profile on page 305” for additional information.

**input_PAN_data**

Direction: Input Type: Character String

The 12 digit personal account number (PAN) if the input PIN format is ISO-0 only. Otherwise, the parameter is ignored.
Secure Messaging for PINs (CSNBSPN)

secmsg_key_identifier
Direction: Input/Output Type: String
The internal token or key label of an internal token of a secure message key for encrypting PINs. This key is used to encrypt the updated clear_text.

output_PIN_profile
Direction: Input Type: String
The three 8-byte character elements that contain information necessary to create a formatted PIN block. If reformatting is not required, the input_PIN_profile and the output_PIN_profile must specify the same PIN block format. Output PIN block formats supported are ISO-0, ISO-1, ISO-2 and ISO-3.

output_PAN_data
Direction: Input Type: String
The 12 digit personal account number (PAN) if the output PIN format is ISO-0 only. Otherwise, this parameter is ignored.

text_length
Direction: Input Type: Integer
The length of the clear_text parameter that follows. Length must be a multiple of eight. Maximum length is 4K.

clear_text
Direction: Input Type: String
Clear text that contains the recovered and/or reformatted/encrypted PIN at offset specified and then encrypted. Any padding or formatting of the message must be done by the caller on input.

initialization_vector
Direction: Input Type: String
The 8-byte supplied string for the TDES-CBC mode of encryption. The initialization_vector is XORed with the first 8 bytes of clear_text prior to encryption. This field is ignored for TDES-ECB mode.

PIN_offset
Direction: Input Type: Integer
The offset within the clear_text parameter where the reformatted PIN block is to be placed. The first byte of the clear_text field is offset 0.

PIN_offset_field_length
Direction: Input Type: Integer
The length of the field within clear_text parameter at PIN_offset where the recovered clear input_PIN_block value is to be placed. The PIN block may be
Secure Messaging for PINs (CSNBSPN)

self-encrypted if requested by the rule array. Length must be eight. The PIN block must fit entirely within the clear_text.

**enciphered_text**

Direction: Output  
Type: String

The field where the enciphered text is returned. The length of this field must be at least as long as the clear_text field.

**output_chaining_vector**

Direction: Output  
Type: String

This field contains the last 8 bytes of enciphered text and is used as the initialization_vector for the next encryption call if data needs to be chained for TDES-CBC mode. No data is returned for TDES-ECB.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

SAF will be invoked to check authorization to use the secure messaging for PINs service and any key labels specified as input.

Keys only appear in the clear within the secure boundary of the cryptographic coprocessors and never in host storage.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 140. Secure messaging for PINs required hardware*

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries</td>
<td>PCI Cryptographic Coprocessor</td>
<td>ISO-3 PIN block format is not supported.</td>
</tr>
<tr>
<td>800</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>ISO-3 PIN block format is not supported.</td>
</tr>
<tr>
<td>990</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>890</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>ISO-3 PIN block format requires the Nov.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td>2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>ISO-3 PIN block format requires the Nov.</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td></td>
<td>2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The SET Block Compose callable service performs DES-encryption of data, OAEP-formattting through a series of SHA-1 hashing operations, and the RSA-encryption of the Optimal Asymmetric Encryption Padding (OAEP) block.

Format

```
CALL CSNDSBC(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  rule_array_count,
  rule_array,
  block_contents_identifier,
  XData_string_length,
  XData_string,
  data_to_encrypt_length,
  data_to_encrypt,
  data_to_hash_length,
  data_to_hash,
  initialization_vector,
  RSA_public_key_identifier_length,
  RSA_public_key_identifier,
  DES_key_block_length,
  DES_key_block,
  RSA_OAEP_block_length,
  RSA_OAEP_block,
  chaining_vector,
  DES_encrypted_data_block )
```

Parameters

**return_code**

*Direction: Output*  
*Type: Integer*

The return code specifies the general result of the callable service. [Appendix A, "ICSF and TSS Return and Reason Codes"](#) lists the return codes.

**reason_code**

*Direction: Output*  
*Type: Integer*

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. [Appendix A, "ICSF and TSS Return and Reason Codes"](#) lists the reason codes.

**exit_data_length**

*Direction: Input/Output*  
*Type: Integer*

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit_data* parameter.
SET Block Compose (CSNDSBC)

exit_data
Direction: Input/Output Type: String

The data that is passed to the installation exit.

rule_array_count
Direction: Input Type: Integer

The number of keywords you are supplying in the rule_array parameter. The value must be 1 or 2.

rule_array
Direction: Input Type: Character String

Keywords that provides control information to the callable service. The keyword must be in 8 bytes of contiguous storage, left-justified and padded on the right with blanks.

Table 141. Keywords for SET Block Compose Control Information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Block Type (required)</strong></td>
<td></td>
</tr>
<tr>
<td>SET1.00</td>
<td>The structure of the RSA-OAEP encrypted block is defined by SET protocol.</td>
</tr>
<tr>
<td>Formatting Information (optional)</td>
<td></td>
</tr>
<tr>
<td>DES-ONLY</td>
<td>DES encryption only is to be performed; no RSA-OAEP formatting will be performed. (See Usage Notes.)</td>
</tr>
</tbody>
</table>

block_contents_identifier
Direction: Input Type: String

A one-byte string, containing a binary value that will be copied into the Block Contents (BC) field of the SET DB data block (indicates what data is carried in the Actual Data Block, ADB, and the format of any extra data (XData_string)). This parameter is ignored if DES-ONLY is specified in the rule-array.

XData_string_length
Direction: Input Type: Integer

The length in bytes of the data contained within XData_string. The maximum length is 94 bytes. This parameter is ignored if DES-ONLY is specified in the rule-array.

XData_string
Direction: Input Type: String

Extra-encrypted data contained within the OAEP-processed and RSA-encrypted block. The format is indicated by block_contents_identifier. For a XData_string_length value of zero, XData_string must still be specified, but will be ignored by ICSF. The string is treated as a string of hexadecimal digits. This parameter is ignored if DES-ONLY is specified in the rule-array.
**SET Block Compose (CSNDSBC)**

**data_to_encrypt_length**

Direction: Input/Output  
Type: Integer

The length in bytes of data that is to be DES-encrypted. The length has a maximum value of 32 MB minus 8 bytes to allow for up to 8 bytes of padding. The data is identified in the `data_to_encrypt` parameter. On output, this value is updated with the length of the encrypted data in the `DES_encrypted_data_block`.

**data_to_encrypt**

Direction: Input  
Type: String

The data that is to be DES-encrypted (with a 64-bit DES key generated by this service). The data will be padded by this service according to the PKSC #5 padding rules.

**data_to_hash_length**

Direction: Input  
Type: Integer

The length in bytes of the data to be hashed. The hash is an optional part of the OAEP block. If the `data_to_hash_length` is 0, no hash will be included in the OAEP block. This parameter is ignored if DES-ONLY is specified in the `rule_array` parameter.

**data_to_hash**

Direction: Input  
Type: String

The data that is to be hashed and included in the OAEP block. No hash is computed or inserted in the OAEP block if the `data_to_hash_length` is 0. This parameter is ignored if DES-ONLY is specified in the `rule_array` parameter.

**initialization_vector**

Direction: Input  
Type: String

An 8-byte string containing the initialization vector to be used for the cipher block chaining for the DES encryption of the data in the `data_to_encrypt` parameter. The same initialization vector must be used to perform the DES decryption of the data.

**RSA_public_key_identifier_length**

Direction: Input  
Type: Integer

The length of the `RSA_public_key_identifier` field. The maximum size is 2500 bytes. This parameter is ignored if DES-ONLY is specified in the rule-array.

**RSA_public_key_identifier**

Direction: Input  
Type: String
SET Block Compose (CSNDSBC)

A string containing either the key label of the RSA public key or the RSA public key token to be used to perform the RSA encryption of the OAEP block. The modulus bit length of the key must be 1024 bytes. This parameter is ignored if DES-ONLY is specified in the rule-array.

**DES_key_block_length**

Direction: Input/Output  Type: Integer

The length of the *DES_key_block*. The current length of this field is defined to be exactly 64 bytes.

**DES_key_block**

Direction: Input/Output  Type: String

The DES key information returned from a previous SET Block Compose service. The contents of the *DES_key_block* is the 64-byte DES internal key token (containing the DES key enciphered under the host master key). Your application program must not change the data in this string.

**RSA_OAEP_block_length**

Direction: Input/Output  Type: Integer

The length of a block of storage to hold the *RSA-OAEP_block*. The length must be at least 128 bytes on input. The length value will be updated on exit with the actual length of the *RSA-OAEP_block*, which is exactly 128 bytes. This parameter is ignored if DES-ONLY is specified in the rule-array.

**RSA_OAEP_block**

Direction: Output  Type: String

The OAEP-formatted data block, encrypted under the RSA public key passed as *RSA_public_key_identifier*. When the OAEP-formatted data block is returned, it is left justified within the *RSA-OAEP_block* field if the input field length (*RSA-OAEP_block_length*) was greater than 128 bytes. This parameter is ignored if DES-ONLY is specified in the rule-array.

**chaining_vector**

Direction: Input/Output  Type: String

An 18-byte field that ICSF uses as a system work area. Your application program must not change the data in this string. This field is ignored by this service, but must be specified.

**DES_encrypted_data_block**

Direction: Output  Type: String

The DES-encrypted data block (data passed in as *data_to_encrypt*). The length of the encrypted data is returned in *data_to_encrypt_length*. The *DES_encrypted_data_block* may be 8 bytes longer than the length of the *data_to_encrypt* because of padding added by this service.
Restrictions

Not all CCA implementations support a key label as input in the `RSA_public_key_identifier` parameter. Some implementations may only support a key token.

The `data_to_encrypt` and the `DES_encrypted_data_block` cannot overlap.

The maximum data block that can be supplied for DES encryption is the limit as expressed by the Encipher callable service (CSNBENC).

**CCF Systems only**: NOCV keys must be installed in the CKDS to use SET block compose service on a CDMF-only system.

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

The first time the SET Block Compose service is invoked to form an RSA-OAEP block and DES-encrypt data for communication between a specific source and destination (for example, between the merchant and payment gateway), do not specify the DES-ONLY keyword. A DES key will be generated by the service and returned in the key token contained in the `DES_key_block`. On subsequent calls to the Compose SET Block service for communication between the same source and destination, the DES key can be re-used. The caller of the service must supply the `DES_key_block`, the `DES_key_block_length`, the `data_to_encrypt`, the `data_to_encrypt_length`, and the rule-array keywords SET1.00 and DES-ONLY. You do not need to supply the block contents identifier, XDATA string and length, RSA-OAEP block and length, and RSA public key information, although you must still specify the parameters. For this invocation, the RSA-OAEP formatting is bypassed and only DES encryption is performed, using the supplied DES key.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>If there are no PCI Cryptographic Coprocessors online, the request is routed to the Cryptographic Coprocessor Feature.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>PCI Cryptographic Coprocessor</td>
<td>This service routes the request to a PCI Cryptographic Coprocessor to perform the RSA-OAEP processing.</td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>
Table 142. SET block compose required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

**SET Block Decompose (CSNDSBD)**

Decomposes the RSA-OAEP block and the DES-encrypted data block of the SET protocol to provide unencrypted data back to the caller.

**Format**

```
CALL CSNDSBD(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    RSA_OAEP_block_length,
    RSA_OAEP_block,
    DES_encrypted_data_block_length,
    DES_encrypted_data_block,
    initialization_vector,
    RSA_private_key_identifier_length,
    RSA_private_key_identifier,
    DES_key_block_length,
    DES_key_block,
    block_contents_identifier,
    XData_string_length,
    XData_string,
    chaining_vector,
    data_block,
    hash_block_length,
    hash_block)
```

**Parameters**

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes," on page 557 lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes," on page 557 lists the reason codes.
SET Block Decompose (CSNDSBD)

exit_data_length
Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

exit_data
Direction: Input/Output  Type: String

The data that is passed to the installation exit.

rule_array_count
Direction: Input  Type: Integer

The number of keywords you are supplying in the rule_array parameter. The value must be 1 or 2.

rule_array
Direction: Input  Type: String

One keyword that provides control information to the callable service. The keyword indicates the block type. The keyword must be in 8 bytes of contiguous storage, left-justified and padded on the right with blanks.

Table 143. Keywords for SET Block Compose Control Information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Block Type (required)</strong></td>
<td></td>
</tr>
<tr>
<td>SET1.00</td>
<td>The structure of the RSA-OAEP encrypted block is defined by SET protocol.</td>
</tr>
<tr>
<td><strong>Formatting Information (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>DES-ONLY</td>
<td>DES decryption only is to be performed; no RSA-OAEP block decryption will be performed. (See Usage Notes.)</td>
</tr>
<tr>
<td>PINBLOCK</td>
<td>Specifies that the OAEP block will contain PIN information in the XDATA field, including an ISO-0 format PIN block. The DES_key_block must be 128 bytes in length and contain a IPINENC or OPINENC key. The PIN block will be encrypted under the PIN encrypting key. The PIN information and the encrypted PIN block are returned in the XDATA_string parameter.</td>
</tr>
</tbody>
</table>

RSA_OAEP_block_length
Direction: Input  Type: Integer

The length of RSA-OAEP_block must be 128 bytes. This parameter is ignored if DES-ONLY is specified in the rule-array.

RSA_OAEP_block
Direction: Input  Type: String
SET Block Decompose (CSNDSBD)

The RSA-encrypted OAEP-formatted data block. This parameter is ignored if DES-ONLY is specified in the rule-array.

**DES_encrypted_data_block_length**

Direction: Input/Output  Type: Integer

The length in bytes of the DES-encrypted data block. The input length must be a multiple of 8 bytes. Updated on return to the length of the decrypted data returned in `data_block`. The maximum value of `DES_encrypted_data_block_length` is 32MB bytes.

**DES_encrypted_data_block**

Direction: Input  Type: String

The DES-encrypted data block. The data will be decrypted and passed back as `data_block`.

**initialization_vector**

Direction: Input  Type: String

An 8-byte string containing the initialization vector to be used for the cipher block chaining for the DES decryption of the data in the `DES_encrypted_data_block` parameter. You must use the same initialization vector that was used to perform the DES encryption of the data.

**RSA_private_key_identifier_length**

Direction: Input  Type: Integer

The length of the `RSA_private_key_identifier` field. The maximum size is 2500 bytes. This parameter is ignored if DES-ONLY is specified in the rule-array.

**RSA_private_key_identifier**

Direction: Input  Type: String

A key label of the RSA private key or an internal token of the RSA private key to be used to decipher the RSA-OAEP block passed in `RSA-OAEP_block`. The modulus bit length of the key must be 1024. This parameter is ignored if DES-ONLY is specified in the rule-array.

**DES_key_block_length**

Direction: Input/Output  Type: Integer

The length of the `DES_key_block`. The current length of this field may be 64 or 128 bytes. If rule array keyword PINBLOCK is specified, the length must be 128 bytes.

**DES_key_block**

Direction: Input/Output  Type: String

The `DES_key_block` contains either one or two DES internal key tokens. If only one token is specified on input, it contains either a null DES token (or binary zeros) or (if DES-ONLY is specified) the DES key information returned from a
previous SET Block Decompose service invocation. This is the 64-byte DES internal key token formed with the DES key which was retrieved from the RSA-OAEP block and enciphered under the host master key. Your application must not change this DES key information. If two tokens are specified in the DES_key_block, the first 64 bytes contain the DES token described previously. The second 64 bytes, used when PINBLOCK is specified in the rule array, contains the DES internal token or the CKDS key label of the IPINENC or OPINENC key used to encrypt the PIN block returned to the caller in the XDATA_string parameter. If a key label is specified, it must be left-justified and padded on the right with blanks.

**block_contents_identifier**

**Direction:** Output  
**Type:** String

A one-byte string, containing the binary value from the block contents (BC) field of the SET data block (DB). It indicates what data is carried in the actual data block (ADB) and the format of any extra data (XData_string). This parameter is ignored if DES-ONLY is specified in the rule-array.

**XData_string_length**

**Direction:** Input/Output  
**Type:** Integer

The length of a string where the data contained within XData_string will be returned. The string must be at least 94 bytes in length. The value will be updated upon exit with the actual length of the returned XData_string. This parameter is ignored if DES-ONLY is specified in the rule-array.

**XData_string**

**Direction:** Output  
**Type:** String

Extra-encrypted data contained within the OAEP-processed and RSA-encrypted block. The format is indicated by block_contents_identifier. The string is treated by ICSF as a string of hexadecimal digits. The service will always return the data from the beginning of the XDataString to the end of the SET DB block, a maximum of 94 bytes of data. The caller must examine the value returned in block_contents_identifier to determine the actual length of the XDataString. This parameter is ignored if DES-ONLY is specified in the rule-array.

**chaining_vector**

**Direction:** Input/Output  
**Type:** String

An 18-byte field that ICSF uses as a system work area. Your application program must not change the data in this string. This field is ignored by this service, but must be specified.

**data_block**

**Direction:** Output  
**Type:** String

The data that was decrypted (passed in as DES_encrypted_data_block). Any padding characters are removed.
SET Block Decompose (CSNDSBD)

**hash_block_length**

**Direction:** Input/Output  
**Type:** Integer

The length in bytes of the SHA-1 hash returned in *hash_block*. On input, this parameter must be set to the length of the *hash_block* field. The length must be at least 20 bytes. On output, this field is updated to reflect the length of the SHA-1 hash returned in the *hash_block* field (exactly 20 bytes). This parameter is ignored if DES-ONLY is specified in the *rule_array* parameter.

**hash_block**

**Direction:** Output  
**Type:** String

The SHA-1 hash extracted from the RSA-OAEP block. This parameter is ignored if DES-ONLY is specified in the *rule_array* parameter.

**Restrictions**

Not all CCA implementations support a key label as input in the *RSA_private_key_identifier* parameter. Some implementations may only support a key token.

The RSA private key used by this service must have been generated as a signature-only key. This restriction does not apply if you are running on the IBM zSeries 990 and subsequent releases.

The *data_block* and the *DES_encrypted_data_block* cannot overlap.

**CCF Systems only:** The ANSI system keys must be installed in the CKDS to use the SET block decompose service on a CDMF-only system.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

When the SET Block Decompose service is invoked without the DES-ONLY keyword, the DES key is retrieved from the RSA-OAEP block and returned in the key token contained in the *DES_key_block*. On subsequent calls to the SET Block Decompose service, a caller can re-use the DES key. The caller of the service must supply the *DES_key_block*, the *DES_key_block_length*, the *DES_encrypted_data_block*, the *DES_encrypted_data_block_length*, the initialization and chaining vectors, and the *rule_array* keywords SET1.00 and DES-ONLY. The RSA private key information, RSA-OAEP block and length, XData string and length, and hash block and length need not be supplied (although the parameters must still be specified). For this invocation, the decryption of the RSA-OAEP block is bypassed; only DES decryption is performed, using the supplied DES key.

When the SET Block Decompose service is invoked with the PINBLOCK keyword, DES-ONLY may not also be specified. If both of these rule array keywords are specified, the service will fail. If PINBLOCK is specified and the *DES_key_block_length* field is not 128, the service will fail.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
### Table 144. SET block decompose required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>If there is no PCI Cryptographic Coprocessor available, the request will be processed on the Cryptographic Coprocessor Feature.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>PCI Cryptographic Coprocessor</td>
<td>A PCI Cryptographic Coprocessor is required if:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• the RSA_private_key_identifier specifies a retained private key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• the RSA_private_key_identifier specifies a CRT private key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• the PINBLOCK rule array keyword is specified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The service has a preference for being processed on a PCI Cryptographic Coprocessor so that the symmetric key does not appear in the clear.</td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>
Transaction Validation (CSNBTRV)

Format

```
CALL CSNBTRV(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    transaction_key_identifier_length,
    transaction_key_identifier,
    transaction_info_length,
    transaction_info,
    validation_values_length,
    validation_values
)
```

Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. [Appendix A, “ICSF and TSS Return and Reason Codes”](#) lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. [Appendix A, “ICSF and TSS Return and Reason Codes”](#) lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the **exit_data** parameter.

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**rule_array_count**

Direction: Input  
Type: Integer

The number of keywords you are supplying in the **rule_array** parameter. The valid values are 1 or 2.

**rule_array**

Direction: Input  
Type: Character String
Keywords that provides control information to the callable service. The keywords are left-justified in an 8-byte field and padded on the right with blanks. The keywords must be in contiguous storage. Specify one or two of the values in Table 145.

Table 145. Rule Array Keywords for Transaction Validation

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>American Express card security codes (required)</td>
<td></td>
</tr>
<tr>
<td>CSC-3</td>
<td>3-digit card security code (CSC) located on the signature panel. VERIFY implied. This is the default.</td>
</tr>
<tr>
<td>CSC-4</td>
<td>4-digit card security code (CSC) located on the signature panel. VERIFY implied.</td>
</tr>
<tr>
<td>CSC-5</td>
<td>5-digit card security code (CSC) located on the signature panel. VERIFY implied.</td>
</tr>
<tr>
<td>CSC-345</td>
<td>Generate 5-byte, 4-byte, 3-byte values when given an account number an an expiration date, GENERATE implied.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation (optional)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VERIFY</td>
<td>Specifies verification of the value presented in the validation values variable.</td>
</tr>
<tr>
<td>GENERATE</td>
<td>Specifies generation of the value presented in the validation values variable.</td>
</tr>
</tbody>
</table>

**transaction_key_identifier_length**

Direction: Input  Type: Integer

The length of the `transaction_key_identifier` parameter.

**transaction_key_identifier**

Direction: Input  Type: String

The labelname or internal token of a MAV or MACVER class key. Key may be single or double length.

**transaction_info_length**

Direction: Input  Type: Integer

The length of the `transaction_info` parameter. For the American Express CSC codes, the length must be 19.

**transaction_info**

Direction: Input  Type: String

For American Express, this is a 19-byte field containing the concatenation of the 4-byte expiration data (in the format YYMM) and the 15-byte American Express account number. Provide the information in character format.

**validation_values_length**

Direction: Input/Output  Type: Integer
**Transaction Validation (CSNBTRV)**

The length of the `validation_values` parameter. Maximum value for this field is 64.

**validation_values**

Direction: Input  
Type: String

This variable contains American Express CSC values. The data is output for **GENERATE** and input for **VERIFY**.

*Table 146. Output description for validation values*

<table>
<thead>
<tr>
<th>Operation</th>
<th>Element Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENERATE and CSC-345</td>
<td>5555544444333 where:</td>
</tr>
<tr>
<td></td>
<td>55555 = CSC 5 value</td>
</tr>
<tr>
<td></td>
<td>44444 = CSC 4 value</td>
</tr>
<tr>
<td></td>
<td>33333 = CSC 3 value</td>
</tr>
<tr>
<td>VERIFY and CSC-3</td>
<td>333 = CSC 3 value</td>
</tr>
<tr>
<td>VERIFY and CSC-4</td>
<td>44444 = CSC 4 value</td>
</tr>
<tr>
<td>VERIFY and CSC-5</td>
<td>55555 = CSC 5 value</td>
</tr>
</tbody>
</table>

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

There are additional access control points for this service.

RACF will be invoked to check authorization for using this service and the label name specified.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 147. Transaction validation required hardware*

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td></td>
<td>Not supported</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>Requires May 2004 or later version of LIC</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Requires May 2004 or later version of LIC</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
VISA CVV Service Generate (CSNBCSG)

Use the VISA CVV Service Generate callable service to generate a:

- VISA Card Verification Value (CVV)
- MasterCard Card Verification Code (CVC)
- Diner's Club Card Verification Value (CVV)

as defined for track 2.

This service generates a CVV that is based upon the information that the PAN_data, the expiration_date, and the service_code parameters provide.

The service uses the Key-A and the Key-B keys to cryptographically process this information. Key-A and Key-B can be single-length DATA or MAC keys. If the requested CVV is shorter than 5 characters, the CVV is padded on the right by space characters. The CVV is returned in the 5-byte variable that the CVV_value parameter identifies. When you verify a CVV, compare the result to the value that the CVV_value supplies.

Format

```
CALL CSNBCSG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    PAN_data,
    expiration_date,
    service_code,
    CVV_key_A_Identifier,
    CVV_key_B_Identifier,
    CVV_value)
```

Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.
reason_code
Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

exit_data_length
Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

exit_data
Direction: Input/Output Type: String

The data that is passed to the installation exit.

rule_array_count
Direction: Input Type: Integer

The number of keywords you are supplying in the rule_array parameter. The parameter rule_array_count must be 0, 1, or 2.

rule_array
Direction: Input Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields, and padded on the right with blanks. All keywords must be in contiguous storage.

Table 148. CVV Generate Rule Array Keywords

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAN data length (optional)</td>
<td></td>
</tr>
<tr>
<td>PAN-13</td>
<td>Specifies that the length of the PAN data is 13 bytes. <strong>PAN-13 is the default value.</strong></td>
</tr>
<tr>
<td>PAN-14</td>
<td>Specifies that the length of the PAN data is 14 bytes.</td>
</tr>
<tr>
<td>PAN-15</td>
<td>Specifies that the length of the PAN data is 15 bytes.</td>
</tr>
<tr>
<td>PAN-16</td>
<td>Specifies that the length of the PAN data is 16 bytes.</td>
</tr>
<tr>
<td>PAN-17</td>
<td>Specifies that the length of the PAN data is 17 bytes.</td>
</tr>
<tr>
<td>PAN-18</td>
<td>Specifies that the length of the PAN data is 18 bytes.</td>
</tr>
<tr>
<td>PAN-19</td>
<td>Specifies that the length of the PAN data is 19 bytes. Requires z990, z890, z9 EC or z9 BC with January 2005 or higher version of Licensed Internal Code (LIC).</td>
</tr>
<tr>
<td>CVV length (optional)</td>
<td></td>
</tr>
<tr>
<td>CVV-1</td>
<td>Specifies that the CVV is to be computed as one byte, followed by 4 blanks. <strong>CVV-1 is the default value.</strong></td>
</tr>
</tbody>
</table>
**Table 148. CVV Generate Rule Array Keywords (continued)**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVV-2</td>
<td>Specifies that the CVV is to be computed as 2 bytes, followed by 3 blanks.</td>
</tr>
<tr>
<td>CVV-3</td>
<td>Specifies that the CVV is to be computed as 3 bytes, followed by 2 blanks.</td>
</tr>
<tr>
<td>CVV-4</td>
<td>Specifies that the CVV is to be computed as 4 bytes, followed by 1 blank.</td>
</tr>
<tr>
<td>CVV-5</td>
<td>Specifies that the CVV is to be computed as 5 bytes.</td>
</tr>
</tbody>
</table>

**PAN_data**

Direction: Input  
Type: String

The `PAN_data` parameter specifies an address that points to the place in application data storage that contains personal account number (PAN) information in character form. The PAN is the account number as defined for the track-2 magnetic-stripe standards.

- If the `PAN-13` keyword is specified in the rule array, 13 characters are processed.
- If the `PAN-14` keyword is specified in the rule array, 14 characters are processed.
- If the `PAN-15` keyword is specified in the rule array, 15 characters are processed.
- If the `PAN-16` keyword is specified in the rule array, 16 characters are processed.
- If the `PAN-17` keyword is specified in the rule array, 17 characters are processed.
- If the `PAN-18` keyword is specified in the rule array, 18 characters are processed.
- If the `PAN-19` keyword is specified in the rule array, 19 characters are processed.

Even if you specify the `PAN-13, PAN-14` or `PAN-15` keywords, the server might copy 16 bytes to a work area. Therefore ensure that the callable service can address 16 bytes of storage.

**expiration_date**

Direction: Input  
Type: String

The `expiration_date` parameter specifies an address that points to the place in application data storage that contains the card expiration date in numeric character form in a 4-byte field. The application programmer must determine whether the CVV will be calculated with the date form of YYMM or MMYY.

**service_code**

Direction: Input  
Type: String

The `service_code` parameter specifies an address that points to the place in application data storage that contains the service code in numeric character form in a 3-byte field. The service code is the number that the track-2 magnetic-stripe standards define. The service code of '000' is supported.
VISA CVV Service Generate (CSNBCSG)

**CVV_key_A_Identifier**

Direction: Input/Output  
Type: String

The `CVV_key_A_Identifier` parameter specifies an address that contains a 64-byte internal key token or a key label of a single-length DATA or MAC key that decrypts information in the CCV process. The internal key token contains the key-A key that encrypts information in the CVV process.

**CVV_key_B_Identifier**

Direction: Input/Output  
Type: String

The `CVV_key_B_Identifier` parameter specifies an address that contains a 64-byte internal key token or a key label of a single-length DATA or MAC key that decrypts information in the CCV process. The internal key token contains the key-B key that decrypts information in the CVV process.

**CVV_value**

Direction: Output  
Type: String

The `CVV_value` parameter specifies an address that points to the place in application data storage that will be used to store the computed 5-byte character output value.

**Restrictions**

The CVV generate callable service is not supported on CCF systems with a CDMF-only configuration.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

PAN-19 requires the z990, z890, z9 EC, z9 BC, z10 EC or z10 BC with January 2005 or higher version of Licensed Internal Code (LIC).

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 149. VISA CVV service generate required hardware*

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>The request is processed on the CCF if Key-A and Key-B are both DATA keys. MAC and MACVER keys are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td>PAN-14, PAN-15, PAN-17, PAN-18 and PAN-19 are not supported.</td>
</tr>
<tr>
<td>PCI Cryptographic Coprocessor</td>
<td></td>
<td>The request is processed on a PCICC if Key-A or Key-B is a MAC key. MACVER keys are not supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PAN-14, PAN-15, PAN-17, PAN-18 and PAN-19 are not supported.</td>
</tr>
</tbody>
</table>
Table 149. VISA CVV service generate required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>MACVER keys are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td>MACVER keys are not supported.</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>MACVER keys are not supported.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td>MACVER keys are not supported.</td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>MACVER keys are not supported.</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td>MACVER keys are not supported.</td>
</tr>
</tbody>
</table>

VISA CVV Service Verify (CSNBCCSV)

Use the VISA CVV Service Verify callable service to verify a:
- VISA Card Verification Value (CVV)
- MasterCard Card Verification Code (CVC)
- Diner's Club Card Verification Value (CVV)
as defined for track 2.

This service verifies a CVV that is based upon the information that the PAN_data, the expiration_date, and the service_code parameters provide.

The service uses the Key-A and the Key-B keys to cryptographically process this information. If the requested CVV is shorter than 5 characters, the CVV is padded on the right by space characters. On a IBM zSeries 800, IBM zSeries 900 or lower, the user must pad out the CVV_value parameter with blanks if the supplied CVV is less than 5 characters. The generated CVV is then compared to the value that the CVV_value supplies for verification.

Format

```
CALL CSNBCCSV(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  rule_array_count,
  rule_array,
  PAN_data,
  expiration_date,
  service_code,
  CVV_key_A_Identifier,
  CVV_key_B_Identifier,
  CVV_value)
```
VISA CVV Service Verify (CSNBCSV)

Parameters

**return_code**
Direction: Output  Type: Integer

The return code specifies the general result of the callable service. [Appendix A, “ICSF and TSS Return and Reason Codes”](#) lists the return codes.

**reason_code**
Direction: Output  Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. [Appendix A, “ICSF and TSS Return and Reason Codes”](#) lists the reason codes.

**exit_data_length**
Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**
Direction: Input/Output  Type: String

The data that is passed to the installation exit.

**rule_array_count**
Direction: Input  Type: Integer

The number of keywords you are supplying in the rule_array parameter. The parameter rule_array_count must be 0, 1, or 2.

**rule_array**
Direction: Input  Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields, and padded on the right with blanks. All keywords must be in contiguous storage.

*Table 150. CVV Verify Rule Array Keywords*

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAN data length (optional)</td>
<td>Specifications that the length of the PAN data is 13 bytes. <strong>PAN-13 is the default value.</strong></td>
</tr>
<tr>
<td>PAN-13</td>
<td>Specifies that the length of the PAN data is 13 bytes. <strong>PAN-13 is the default value.</strong></td>
</tr>
<tr>
<td>PAN-14</td>
<td>Specifies that the length of the PAN data is 14 bytes.</td>
</tr>
<tr>
<td>PAN-15</td>
<td>Specifies that the length of the PAN data is 15 bytes.</td>
</tr>
<tr>
<td>PAN-16</td>
<td>Specifies that the length of the PAN data is 16 bytes.</td>
</tr>
<tr>
<td>PAN-17</td>
<td>Specifies that the length of the PAN data is 17 bytes.</td>
</tr>
</tbody>
</table>
Table 150. CVV Verify Rule Array Keywords (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAN-18</td>
<td>Specifies that the length of the PAN data is 18 bytes.</td>
</tr>
<tr>
<td>PAN-19</td>
<td>Specifies that the length of the PAN data is 19 bytes. Requires z990, z890, z9 EC or z9 BC with January 2005 or higher version of Licensed Internal Code (LIC).</td>
</tr>
<tr>
<td><strong>CVV length (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>CVV-1</td>
<td>Specifies that the CVV is to be computed as one byte, followed by 4 blanks. <strong>CVV-1 is the default value.</strong></td>
</tr>
<tr>
<td>CVV-2</td>
<td>Specifies that the CVV is to be computed as 2 bytes, followed by 3 blanks.</td>
</tr>
<tr>
<td>CVV-3</td>
<td>Specifies that the CVV is to be computed as 3 bytes, followed by 2 blanks.</td>
</tr>
<tr>
<td>CVV-4</td>
<td>Specifies that the CVV is to be computed as 4 bytes, followed by 1 blank.</td>
</tr>
<tr>
<td>CVV-5</td>
<td>Specifies that the CVV is to be computed as 5 bytes.</td>
</tr>
</tbody>
</table>

**PAN_data**

Direction: Input  
Type: String

The `PAN_data` parameter specifies an address that points to the place in application data storage that contains personal account number (PAN) information in character form. The PAN is the account number as defined for the track-2 magnetic-stripe standards.

- If the **PAN-13** keyword is specified in the rule array, 13 characters are processed.
- If the **PAN-14** keyword is specified in the rule array, 14 characters are processed.
- If the **PAN-15** keyword is specified in the rule array, 15 characters are processed.
- If the **PAN-16** keyword is specified in the rule array, 16 characters are processed.
- If the **PAN-17** keyword is specified in the rule array, 17 characters are processed.
- If the **PAN-18** keyword is specified in the rule array, 18 characters are processed.
- If the **PAN-19** keyword is specified in the rule array, 19 characters are processed.

Even if you specify the **PAN-13**, **PAN-14** or **PAN-15** keywords, the server might copy 16 bytes to a work area. Therefore ensure that the callable service can address 16 bytes of storage.

**expiration_date**

Direction: Input  
Type: String

The `expiration_date` parameter specifies an address that points to the place in application data storage that contains the card expiration date in numeric character form in a 4-byte field. The application programmer must determine whether the CVV will be calculated with the date form of YYMM or MMYY.
VISA CVV Service Verify (CSNBCCSV)

**service_code**

Direction: Input  
Type: String  

The *service_code* parameter specifies an address that points to the place in application data storage that contains the service code in numeric character form in a 3-byte field. The service code is the number that the track-2 magnetic-stripe standards define. The service code of '000' is supported.

**CVV_key_A_Identifier**

Direction: Input/Output  
Type: String  

The *CVV_key_A_Identifier* parameter specifies an address that contains a 64-byte internal key token or a key label of a single-length DATA, MAC or MACVER key that decrypts information in the CVV process. The internal key token contains the key-A key that encrypts information in the CVV process.

**CVV_key_B_Identifier**

Direction: Input/Output  
Type: String  

The *CVV_key_B_Identifier* parameter specifies an address that contains a 64-byte internal key token or a key label of a single-length DATA, MAC or MACVER key that decrypts information in the CVV process. The internal key token contains the key-B key that decrypts information in the CVV process.

**CVV_value**

Direction: Input  
Type: String  

The *CVV_value* parameter specifies an address that contains the CVV value which will be compared to the computed CVV value. This is a 5-byte field.

On an IBM zSeries 800 or IBM zSeries 900, the user must pad out the *CVV_value* parameter with blanks if the supplied CVV is less than 5 characters.

**Restrictions**

The CVV verify callable service is not supported on CCF systems with a CDMF-only configuration.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

PAN-19 requires the z990, z890, z9 EC, z9 BC, z10 EC or z10 BC with January 2005 or higher version of Licensed Internal Code (LIC).

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
### VISA CVV Service Verify (CSNBCSV)

**Table 151. VISA CVV service verify required hardware**

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>The request is processed on the CCF if Key-A and Key-B are both DATA keys. MAC and MACVER keys are not supported. PAN-14, PAN-15, PAN-17, PAN-18 and PAN-19 are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>PCI Cryptographic Coprocessor</td>
<td>The request is processed on a PCICC if Key-A or Key-B is a MAC or MACVER key. PAN-14, PAN-15, PAN-17, PAN-18 and PAN-19 are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

Chapter 8. Financial Services 377
VISA CVV Service Verify (CSNBCSV)
Chapter 9. Using Digital Signatures

This topic describes the PKA callable services that support using digital signatures to authenticate messages.

- "Digital Signature Generate (CSNDDSG and CSNFLDSG)"
- "Digital Signature Verify (CSNDDSV and CSNFLDSV)" on page 385

Digital Signature Generate (CSNDDSG and CSNFLDSG)

Use the digital signature generate callable service to generate a digital signature using a PKA private key. The digital signature generate callable service may use either the RSA or DSS private key, depending on the algorithm you are using. DSS is not supported on the PCIXCC, CEX2C, or CEX3C.

The RSA private key must be valid for signature usage. This service supports these methods:
- ANSI X9.30 (DSS)
- ANSI X9.31 (RSA)
- ISO 9796-1 (RSA)
- RSA DSI PKCS 1.0 and 1.1 (RSA)
- Padding on the left with zeros (RSA)

Note: The maximum signature length is 512 bytes (4096 bits).

The input text should have been previously hashed using either the one-way hash generate callable service or the MDC generation callable service. If the signature formatting algorithm specifies ANSI X9.31, you must specify the hash algorithm used to hash the text (SHA-1 or RPMD-160). See "Formatting Hashes and Keys in Public-Key Cryptography" on page 695.

If the PKA_private_key_identifier specifies an RSA private key, you select the method of formatting the text through the rule_array parameter. If the PKA_private_key_identifier specifies a DSS private key, the DSS signature generated is according to ANSI X9.30. For DSS, the signature is generated on a 20-byte hash created from SHA-1 algorithm.

Note: For PKCS the message digest and the message-digest algorithm identifier are combined into an ASN.1 value of type DigestInfo, which is BER-encoded to give an octet string D (see Table 152). D is the text string supplied in the hash variable.

This callable service supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSNFLDSG.
Digital Signature Generate (CSNDDSG and CSNFDSG)

Format

```c
CALL CSNDDSG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    PKA_private_key_identifier_length,
    PKA_private_key_identifier,
    hash_length,
    hash,
    signature_field_length,
    signature_bit_length,
    signature_field)
```

Parameters

**return_code**
- Direction: Output
- Type: Integer

The return code specifies the general result of the callable service. [Appendix A, “ICSF and TSS Return and Reason Codes”](#) lists the return codes.

**reason_code**
- Direction: Output
- Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. [Appendix A, “ICSF and TSS Return and Reason Codes”](#) lists the reason codes.

**exit_data_length**
- Direction: Input/Output
- Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the `exit_data` parameter.

**exit_data**
- Direction: Input/Output
- Type: String

The data that is passed to the installation exit.

**rule_array_count**
- Direction: Input
- Type: Integer

The number of keywords you are supplying in the `rule_array` parameter. The value may be 0, 1, or 2.

**rule_array**
- Direction: Input
- Type: String
Keywords that provide control information to the callable service. A keyword specifies the method for calculating the RSA digital signature. Table 152 lists the keywords. Each keyword is left-justified in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage.

Table 152. Keywords for Digital Signature Generate Control Information - Valid only for RSA key types.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO-9796</td>
<td>Calculate the digital signature on the hash according to ISO-9796-1. Any hash method is allowed. This is the default.</td>
</tr>
<tr>
<td>PKCS-1.0</td>
<td>Calculate the digital signature on the BER-encoded ASN.1 value of the type DigestInfo containing the hash according to the RSA Data Security, Inc. Public Key Cryptography Standards #1 block type 00. The text must have been hashed prior to inputting to this service.</td>
</tr>
<tr>
<td>PKCS-1.1</td>
<td>Calculate the digital signature on the BER-encoded ASN.1 value of the type DigestInfo containing the hash according to the RSA Data Security, Inc. Public Key Cryptography Standards #1 block type 01. The text must have been hashed prior to inputting to this service.</td>
</tr>
<tr>
<td>ZERO-PAD</td>
<td>Format the hash by padding it on the left with binary zeros to the length of the RSA key modulus. Any supported hash function is allowed.</td>
</tr>
<tr>
<td>X9.31</td>
<td>Format according to the ANSI X9.31 standard. The input text must have been previously hashed with one of these hash algorithms:</td>
</tr>
<tr>
<td></td>
<td><strong>Hash Method Specification: Required with X9.31</strong></td>
</tr>
<tr>
<td>RPMD-160</td>
<td>Hash the input text using the RIPEMD-160 hash method.</td>
</tr>
<tr>
<td>SHA-1</td>
<td>Hash the input text using the SHA-1 hash method.</td>
</tr>
</tbody>
</table>

**PKA_private_key_identifier_length**

*Direction: Input*  
*Type: Integer*

The length of the *PKA_private_key_identifier* field. The maximum size is 3500 bytes.

**PKA_private_key_identifier**

*Direction: Input*  
*Type: String*

An internal token or label of the PKA private key or Retained key. If the signature format is X9.31, the modulus of the RSA key must have a length of at least 1024 bits.

**hash_length**

*Direction: Input*  
*Type: Integer*
Digital Signature Generate (CSNDDSG and CSNFDSG)

The length of the hash parameter in bytes. It must be the exact length of the text to sign. The maximum size is 512 bytes. If you specify ZERO-PAD in the rule_array parameter, the length is restricted to 36 bytes unless the RSA key is a signature only key, then the maximum length is 512 bytes.

On the IBM Eserver zSeries 990 and subsequent releases, the hash length limit is controlled by a new access control point. Only RSA key management keys are affected by this access control point. The limit for RSA signature use only keys is 512 bytes. This new access control point is always disabled in the Default role. You must have a TKE workstation to enable it.

hash

Direction: Input  Type: String

The application-supplied text on which to generate the signature. The input text must have been previously hashed, and for PKCS formatting, it must be BER-encoded as previously described. For X9.31, the hash algorithms must have been either SHA-1 or RIPEMD-160. See the rule_array parameter for more information.

signature_field_length

Direction: Input/Output  Type: Integer

The length in bytes of the signature_field to contain the generated digital signature. Upon return, this field contains the actual length of the generated signature.

Note: For RSA, this must be at least the RSA modulus size (rounded up to a multiple of 32 bytes for the X9.31 signature format, or one byte for all other signature formats). For DSS, this must be at least 40 bytes. For RSA and DSS, this field is updated with the minimum byte length of the digital signature. The maximum size is 512 bytes.

signature_bit_length

Direction: Output  Type: Integer

The bit length of the digital signature generated. For ISO-9796 this is 1 less than the modulus length. For other RSA processing methods, this is the modulus length. For DSS, this is 320.

signature_field

Direction: Output  Type: String

The digital signature generated is returned in this field. The digital signature is in the low-order bits (right-justified) of a string whose length is the minimum number of bytes that can contain the digital signature. This string is left-justified within the signature_field. Any unused bytes to the right are undefined.

Restrictions

Although ISO-9796 does not require the input hash to be an integral number of bytes in length, this service requires you to specify the hash_length in bytes.

X9.31 requires the RSA token to have a modulus bit length of at least 1024 bits and the length must also be a multiple of 256 bits (or 32 bytes).
Digital Signature Generate (CSNDDSG and CSNFDSG)

The length of the hash parameter in bytes. It must be the exact length of the text to sign. The maximum size is 512 bytes. If you specify ZERO-PAD in the rule_array parameter, the length is restricted to 36 bytes unless the RSA key is a signature only key, then the maximum length is 512 bytes.

On the IBM @server zSeries 990 and subsequent releases, the hash length limit is controlled by a new access control point. If OFF (disabled), the maximum hash length limit for ZERO-PAD is the modulus length of the PKA private key. If ON (enabled), the maximum hash length limit for ZERO-PAD is 36 bytes. Only RSA key management keys are affected by this access control point. The limit for RSA signature use only keys is 512 bytes. This new access control point is always disabled in the Default role. You must have a TKE workstation to enable it.

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
Table 153. Digital signature generate required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
</table>
| IBM @server zSeries 800 | Cryptographic Coprocessor Feature | The request is processed on the CCF when:  
  - the modulus bit length of the RSA key is less than 512 bits  
  - the key specified is a DSS key  
  - the key specified is a X'02' private modulus-exponent RSA key  
  - the key specified is a X'06' private modulus-exponent RSA key and the key use bits indicate signature only  
  - the key specified is a X'06' private modulus-exponent RSA key and the key use bits indicate key-management use and the SMK is equal to the KMMK  
RSA keys with moduli greater than 1024-bit length are not supported. |
| IBM @server zSeries 900 | PCI Cryptographic Coprocessor | The request is processed on the PCICC when  
  - the key specified is a X'08' CRT RSA key  
  - the key specified is a retained key. The request will be routed to the specific coprocessor of the retained key.  
  - the key specified is a X'06' private modulus-exponent RSA key and the key use bits indicate signature only  
  - the key specified is a X'06' private modulus-exponent RSA key and the key use bits indicate key-management use and the SMK is equal to the KMMK  
  - the key specified is a X'06' private modulus-exponent RSA key and the key use bits indicate key-management use and the SMK is not equal to the KMMK  
RSA keys with moduli greater than 2048-bit length are not supported. |
| IBM @server zSeries 990 | PCI X Cryptographic Coprocessor | DSS tokens are not supported.  
ZERO-PAD hash length is controlled by an access control point. When enabled, the hash length limit is 36 bytes. When disabled, the hash length limit is the modulus byte length of the RSA key. This access control point is always disabled and can only be enabled with TKE V4.0 or higher.  
RSA keys with moduli greater than 2048-bit length are not supported. |
### Digital Signature Generate (CSNDDSG and CSNFD SG)

Table 153. Digital signature generate required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>DSS tokens are not supported.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td>ZERO-PAD hash length is controlled by an access control point. When enabled, the hash length limit is 36 bytes. When disabled, the hash length limit is the modulus byte length of the RSA key. This access control point is always disabled and can only be enabled with TKE V4.0 or higher.</td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>DSS tokens are not supported.</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td>ZERO-PAD hash length is controlled by an access control point. When enabled, the hash length limit is 36 bytes. When disabled, the hash length limit is the modulus byte length of the RSA key. This access control point is always disabled and can only be enabled with TKE V4.0 or higher. RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
</tbody>
</table>

### Digital Signature Verify (CSNDDSV and CSNFD SV)

Use the digital signature verify callable service to verify a digital signature using a PKA public key.

- The digital signature verify callable service can use the RSA or DSS public key, depending on the digital signature algorithm used to generate the signature. DSS is supported only on the IBM @server zSeries 800 and IBM @server zSeries 900.

- The digital signature verify callable service can also use the public keys that are contained in trusted blocks regardless of whether the block also contains rules to govern its use when generating or exporting keys with the RKX service. The format of the trusted block allows DSV to distinguish it from other RSA key tokens, and therefore no special rule array keyword or other parameters are required in order to indicate that the trusted block is being used. However, if the TPK-ONLY keyword is used in the rule_array, an error will occur if the PKA_public_key_identifier does not contain a trusted block.

This service supports these methods:
- ANSI X9.30 (DSS)
- ANSI X9.31 (RSA)
- ISO 9796 (RSA)
- RSA DSI PKCS 1.0 and 1.1 (RSA)
Digital Signature Verify (CSNDDSV and CSNFDSV)

- Padding on the left with zeros (RSA)

Input text should have been previously hashed. You can use either the one-way hash generate callable service or the MDC generation callable service. See also "Formatting Hashes and Keys in Public-Key Cryptography" on page 695.

**Note:** The maximum signature length is 512 bytes (4096 bits).

This callable service supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSNFDSV.

**Format**

```call
CALL CSNDDSV(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  rule_array_count,
  rule_array,
  PKA_public_key_identifier_length,
  PKA_public_key_identifier,
  hash_length,
  hash,
  signature_field_length,
  signature_field)
```

**Parameters**

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the installation exit. The length can be from `X'00000000'` to `X'7FFFFFFF'` (2 gigabytes). The data is identified in the `exit_data` parameter.

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation exit.
Digital Signature Verify (CSNDDSV and CSNFDSV)

**rule_array_count**

Direction: Input  
Type: Integer

The number of keywords you are supplying in the *rule_array* parameter. The value must be 0, 1, or 2.

**rule_array**

Direction: Input  
Type: String

Contains an array of keywords. Keywords that provide control information to the callable service. A keyword specifies the method to use to verify the RSA digital signature. Table 154 lists the keywords. Each keyword is left-justified in an 8-byte field and padded on the right with blanks. All keywords must be in contiguous storage.

Table 154. Keywords for Digital Signature Verify Control Information. Valid Only for RSA Key Types.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Digital signature hash formatting method (one, optional, for RSA)</strong></td>
<td></td>
</tr>
<tr>
<td>X9.31</td>
<td>Format the hash according to the ANSI X9.31 standard and compare to the digital signature.</td>
</tr>
<tr>
<td>ISO-9796</td>
<td>Format the hash according to the ISO 9796-1 standard and compare to the digital signature. This is the default.</td>
</tr>
<tr>
<td>PKCS-1.0</td>
<td>Format the hash as specified in the RSA Data Security, Inc., Public Key Cryptography Standards #1 block type 00 and compare to the digital signature. The text must have been hashed and BER-encoded prior to inputting to this service.</td>
</tr>
<tr>
<td>PKCS-1.1</td>
<td>Format the hash as specified in the RSA Data Security, Inc., Public Key Cryptography Standards #1 block type 01 and compare to the digital signature. The text must have been hashed and BER-encoded prior to inputting to this service.</td>
</tr>
<tr>
<td>ZERO-PAD</td>
<td>Format the hash by padding it on the left with binary zeros to the length of the PKA key modulus. Any supported hash function is allowed.</td>
</tr>
</tbody>
</table>

| **PKA public key token type (one, optional)**                                                                 |
| TPK-ONLY         | The PKA_public_key_identifier must be a trusted block that contains, at a minimum, two sections:                                          |
|                  | 1. Trusted Block Information section 0x14 which is required for all trusted blocks and                                                  |
|                  | 2. Trusted Public Key section 0x11 which contains the trusted public key and usage rules that indicate whether or not the trusted public key can be used in digital signature operations. |

PKA_public_key_identifier_length

Direction: Input  
Type: Integer

The length of the *PKA_public_key_identifier* parameter containing the public key token or label. The maximum size is 3500 bytes.
Digital Signature Verify (CSNDDSV and CSNFDSV)

**PKA_public_key_identifier**
Direction: Input Type: String

A token or label of the PKA public key or internal trusted block.

**hash_length**
Direction: Input Type: Integer

The length of the hash parameter in bytes. It must be the exact length of the text that was signed. The maximum size is 512 bytes.

**hash**
Direction: Input Type: String

The application-supplied text on which the supplied signature was generated. The text must have been previously hashed and, for PKCS formatting, BER-encoded as previously described.

**signature_field_length**
Direction: Input Type: Integer

The length in bytes of the signature_field parameter. The maximum size is 512 bytes.

**signature_field**
Direction: Input Type: String

This field contains the digital signature to verify. The digital signature is in the low-order bits (right-justified) of a string whose length is the minimum number of bytes that can contain the digital signature. This string is left-justified within the signature_field.

**Restrictions**

The ability to recover a message from a signature (which ISO-9796 allows but does not require) is not supported.

The exponent of the RSA public key must be odd.

Although ISO-9796 does not require the input hash to be an integral number of bytes in length, this service requires you to specify the hash_length in bytes.

X9.31 requires the RSA token to have a modulus bit length of at least 1024 bits and the length must also be a multiple of 256 bits (or 32 bytes).

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

For DSS if r=0 or s=0 then verification always fails. The DSS digital signature is of the form r || s, each 20 bytes.
Digital Signature Verify (CSNDDSV and CSNFDSV)

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

**Table 155. Digital signature verify required hardware**

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>Trusted key block not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td>TPK-ONLY keyword not supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA keys with moduli greater than 1024-bit length are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>DSS tokens are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td>Trusted key block not supported.</td>
</tr>
<tr>
<td></td>
<td>Crypto Express2 Accelerator</td>
<td>TPK-ONLY keyword not supported.</td>
</tr>
<tr>
<td></td>
<td>PCI Cryptographic Accelerator</td>
<td>RSA keys with moduli greater than 2048-bit length are not supported.</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>DSS tokens are not supported.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express2 Accelerator</td>
<td>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>DSS tokens are not supported.</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td></td>
<td>Crypto Express2 Accelerator</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Crypto Express3 Accelerator</td>
<td></td>
</tr>
</tbody>
</table>
Digital Signature Verify (CSNDDSV and CSNFDSV)
Chapter 10. Managing PKA Cryptographic Keys

This topic describes the callable services that generate and manage PKA keys.

- **PKA Key Generate (CSNDPKG and CSNFPKG)**
- **PKA Key Import (CSNDPKI and CSNFPKI)** on page 395
- **PKA Key Token Build (CSNDPKB and CSNFPKB)** on page 399
- **PKA Key Token Change (CSNDKTC)** on page 409
- **PKA Key Translate (CSNDPKT and CSNFPKT)** on page 412
- **PKA Public Key Extract (CSNDPKX and CSNFPKX)** on page 415
- **PKDS Record Create (CSNDKRC and CSNFKRC)** on page 418
- **PKDS Record Delete (CSNDKRD and CSNFKRD)** on page 420
- **PKDS Record Read (CSNDKRR)** on page 422
- **PKDS Record Write (CSNDKRW)** on page 424
- **Retained Key Delete (CSNDRKD and CSNFRKD)** on page 426
- **Retained Key List (CSNDRKL and CSNFRKL)** on page 429

### PKA Key Generate (CSNDPKG and CSNFPKG)

Use the PKA key generate callable service to generate these PKA keys:

- PKA internal tokens for use with the DSS algorithm in the digital signature services
- RSA keys for use on the Cryptographic Coprocessor Feature, PCI Cryptographic Coprocessor, PCI X Cryptographic Coprocessor, Crypto Express2 Coprocessor, or Crypto Express3 Coprocessor.

Input to the PKA key generate callable service is either a skeleton key token that has been built by the PKA key token build service or a valid internal token. In the case of a valid internal token, PKG will generate a key with the same modulus length and the same exponent. Internal tokens with a X'09' section are not supported.

DSS key generation requires this information in the input skeleton token:

- Size of modulus p in bits
- Prime modulus p
- Prime divisor q
- Public generator g
- Optionally, the private key name

DSS standards define restrictions on p, q, and g. (Refer to the Federal Information Processing Standard (FIPS) Publication 186 for DSS standards.) This callable service does not verify all of these restrictions. If you do not follow these restrictions, the keys you generate may not be valid DSS keys. The PKA Key Token Build service or an existing internal or external PKA DSS token can generate the input skeleton token, but all of the preceding must be provided. You can extract the DSS public key token from the internal private key token by calling the PKA public key extract callable service.

**Note:** DSS keys are not supported on a PCIXCC, CEX2C, or CEX3C.

RSA key generation requires this information in the input skeleton token:

- Size of the modulus in bits. The modulus for modulus-exponent form keys is between 512 and 1024. The CRT modulus is between 512 and 4096. The modulus for the variable-length-modulus-exponent form is between 512 and 4096.
PKA Key Generate (CSNDPKG and CSNFPKG)

RSA key generation has these restrictions: For modulus-exponent, there are restrictions on modulus, public exponent, and private exponent. For CRT, there are restrictions on dp, dq, U, and public exponent. See the Key value structure in "PKA Key Token Build (CSNDPKB and CSNFPKB)" on page 399 for a summary of restrictions.

**Note:** The Transaction Security System PKA96 PKA key generate verb supports RSA key generation only; it does not support DSS key generation.

This callable service supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSNFPKG.

**Format**

```call csndpkg(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  rule_array_count,
  rule_array,
  regeneration_data_length,
  regeneration_data,
  skeleton_key_identifier_length,
  skeleton_key_identifier,
  transport_key_identifier,
  generated_key_token_length,
  generated_key_token
)
```

**Parameters**

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. [Appendix A, ICSF and TSS Return and Reason Codes](#) lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. [Appendix A, "ICSF and TSS Return and Reason Codes"](#) lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the **exit_data** parameter.

**exit_data**

Direction: Input/Output  
Type: String
PKA Key Generate (CSNDPKG and CSNFPKG)

The data that is passed to the installation exit.

**rule_array_count**

Direction: Input  
Type: Integer

The number of keywords you supplied in the rule_array parameter. Value may be 1 or 2.

**rule_array**

Direction: Input  
Type: String

A keyword that provides control information to the callable service. See Table 156 for a list. A keyword is left-justified in an 8-byte field and padded on the right with blanks.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Private Key Encryption (required)</strong></td>
<td></td>
</tr>
<tr>
<td>CLEAR</td>
<td>Return the private key in clear text. The private key in clear text is an external token. Only valid for RSA keys.</td>
</tr>
<tr>
<td>MASTER</td>
<td>Encipher the private key under the master key. The keyword is not supported if a skeleton token with a X'09' section is provided.</td>
</tr>
<tr>
<td>RETAIN</td>
<td>Retain the private key within the PCI Cryptographic Coprocessor for additional security. Only valid for RSA signature keys. The keyword is not supported if a skeleton token with a X'09' section is provided.</td>
</tr>
<tr>
<td>XPORT</td>
<td>Encipher the private key under the transport_key_identifier. Only valid for RSA keys.</td>
</tr>
<tr>
<td><strong>Options (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>CLONE</td>
<td>Mark a generated and retained private key as usable in cryptographic engine cloning process. This keyword is supported only if RETAIN is also specified. Only valid for RSA keys. The keyword is not supported if a skeleton token with a X'09' section is provided.</td>
</tr>
</tbody>
</table>

**regeneration_data_length**

Direction: Input  
Type: Integer

The value must be 0 for DSS tokens. For RSA tokens, the regeneration_data_length can be non-zero. If it is non-zero, it must be between 8 and 512 bytes inclusive.

**regeneration_data**

Direction: Input  
Type: String

This field points to a string variable containing a string used as the basis for creating a particular public-private key pair in a repeatable manner.

**skeleton_key_identifier_length**

Direction: Input  
Type: Integer
PKA Key Generate (CSNDPKG and CSNFPKG)

The length of the skeleton_key_identifier parameter in bytes. The maximum allowed value is 3500 bytes.

**skeleton_key_identifier**

**Direction:** Input  
**Type:** String

The application-supplied skeleton key token generated by PKA key token build or label of the token that contains the required network quantities for DSS key generation, or the required modulus length and public exponent for RSA key generation. If RETAIN was specified and the skeleton_key_identifier is a label, the label must match the private key name of the key.

The skeleton_key_identifier parameter must contain a token which specifies a modulus length in the range 512 – 4096 bits.

**transport_key_identifier**

**Direction:** Input  
**Type:** String

A 64-byte field to contain a DES key identifier. This field must be binary zeros, unless the XPORT rule is specified. For XPORT rule, this is an IMPORTER or EXPORTER key or the label of an IMPORTER or EXPORTER key that is used to encrypt the generated key. If you specify a label, it must resolve uniquely to either an IMPORTER or EXPORTER key. Only valid for RSA keys.

**generated_key_token_length**

**Direction:** Input/Output  
**Type:** Integer

The length of the generated key token. The field is checked to ensure it is at least equal to the token being returned. The maximum size is 3500 bytes. On output, this field is updated with the actual token length.

**generated_key_token**

**Direction:** Input/Output  
**Type:** String

The internal token or label of the generated DSS or RSA key. The label can be that of a retained key. Checks are made to ensure that a retained key is not overlayed in PKDS. If the label is that of a retained key, the private name in the token must match the label name. If a label is specified in the generated_key_token field, the generated_key_token_length returned to the application will be the same as the input length. If RETAIN was specified, but the generated_key_token was not specified as a label, the generated key length returned to the application will be zero (the key was retained in the PCI Cryptographic Coprocessor). If the record already exists in the PKDS with the same label as the one specified as the generated_key_token, the record will be overwritten with the newly generated key token (unless the PKDS record is an existing retained private key, in which case it cannot be overwritten). If there is no existing PKDS record with this label in the case of generating a retained key, a record will be created. For generation of a non-retained key, if a label is specified in the generated_key_token field, a record must already exist in the PKDS with this same label or the service will fail.
PKA Key Generate (CSNDPKG and CSNFPKG)

Restrictions

2048-bit RSA keys may have a public exponent in the range of 1-256 bytes. 4096-bit RSA key public exponents are restricted to the values 3 and 65537.

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 157. PKA key generate required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required Cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>The service examines the skeleton token and routes the generation request to the appropriate cryptographic processor. If the skeleton is a DSS key token, processing takes place on the Cryptographic Coprocessor Feature.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>PCI Cryptographic Coprocessor</td>
<td>The service examines the skeleton token and routes the generation request to the appropriate cryptographic processor. If the skeleton is a DSS key token, processing takes place on the Cryptographic Coprocessor Feature.</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express2 Coprocessor</td>
<td>DSS tokens are not supported. RSA keys with moduli greater than 2048-bit length are not supported.</td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express2 Coprocessor</td>
<td>DSS tokens are not supported. RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
</tbody>
</table>

PKA Key Import (CSNDPKI and CSNFPKI)

Use this service to import an external PKA private key token. (The private key must consist of a PKA private key and public key.) The secret values of the key may be clear or encrypted under a limited-authority DES importer key.

This service can also import a clear PKA key. The PKA key token build service creates a clear PKA key token.

This service can also import an external trusted block token for use with the remote key export callable service.
PKA Key Import (CSNDPKI and CSNFPKI)

Output of this service is an ICSF internal token of the RSA, DSS private key or trusted block.

This callable service supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSNFPKI.

**Restriction**: DSS keys are not supported on the PCIXCC, CEX2C, or CEX3C.

**Format**

```c
CALL CSNDPKI(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    source_key_identifier_length,
    source_key_identifier,
    importer_key_identifier,
    target_key_identifier_length,
    target_key_identifier)
```

**Parameters**

**return_code**

Direction: Output  Type: Integer

The return code specifies the general result of the callable service. Appendix A, ICSF and TSS Return and Reason Codes lists the return codes.

**reason_code**

Direction: Output  Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, ICSF and TSS Return and Reason Codes lists the reason codes.

**exit_data_length**

Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X’00000000’ to X’7FFFFFFF’ (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**

Direction: Input/Output  Type: String

The data that is passed to the installation exit.

**rule_array_count**

Direction: Input  Type: Integer
PKA Key Import (CSNDPKI and CSNFPKI)

The number of keywords you supplied in the rule_array parameter. This must be 0.

**rule_array**

Direction: Input  
Type: String

Reserved field. This field is not used, but you must specify it.

**source_key_identifier_length**

Direction: Input  
Type: Integer

The length of the source_key_identifier parameter. The maximum size is 3500 bytes.

**source_key_identifier**

Direction: Input  
Type: String

Contains an external token or label of a PKA private key, without section identifier 0x14 (Trusted Block Information), or the trusted block in external form as produced by the Trusted Block Create (CSNDTBC) service with the ACTIVATE keyword.

If a PKA private key without the section identifier 0x14 is passed in:

- There are no qualifiers. A retained key can not be used.
- The key token must contain both public-key and private-key information. The private key can be in cleartext or it can be enciphered.
- This is the output of the PKA key generate (CSNDPKG) callable service or the PKA key token build (CSNDPKB) callable service.
- If encrypted, it was created on another platform.

If a PKA key token with section 0x14 is passed in:

- This service will be used to encipher the MAC key within the trusted block under the PKA master key instead of the IMP-PKA key-encrypting key.
- The importer_key_identifier must contain an IMP-PKA KEK in this case.

**importer_key_identifier**

Direction: Input/Output  
Type: String

A DES internal token or the label of an IMP-PKA key. This is a limited authority key-encrypting key. It is ignored for clear tokens.

**target_key_identifier_length**

Direction: Input/Output  
Type: Integer

The length of the target_key_identifier parameter. The maximum size is 3500 bytes. On output, and if the size is of sufficient length, the variable is updated with the actual length of the target_key_identifier field.

**target_key_identifier**

Direction: Input/Output  
Type: String
PKA Key Import (CSNDPKI and CSNFPKI)

This field contains the internal token or label of the imported PKA private key or a Trusted Block. If a label is specified on input, a PKDS record with this label must exist. The PKDS record with this label will be overwritten with imported key unless the existing record is a retained key. If the record is a retained key, the import will fail. A retained key record cannot be overwritten. If no label is specified on input, this field should be set to binary zeros on input.

Restrictions

This service imports RSA keys of up to 4096 bits. However, the hardware configuration sets the limits on the modulus size of keys for digital signatures and key management; thus, the key may be successfully imported but fail when used if the limits are exceeded.

The importer_key_identifier is a limited-authority key-encrypting key.

CRT form tokens with a private section ID of X'05' cannot be imported into ICSF.

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

An RSA modulus-exponent form token imported on the PCICC, PCIXCC, CEX2C, or CEX3C results in a X'06' format, while a token imported on a Cryptographic Coprocessor Feature will result in a X'02' format. If the modulus length is less than 512, the token will be imported on the CCF, and it will be X'02' format.

This service imports keys of any modulus size up to 4096 bits. However, the hardware configuration sets the limits on the modulus size of keys for digital signatures and key management; thus, the key may be successfully imported but fail when used if the limits are exceeded.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
PKA Key Import (CSNDPKI and CSNFPKI)

Table 158. PKA key import required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td>The request will be processed on the CCF when</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td>• the source_key_identifier contains an RSA modulus-exponent private key with a modulus length of less than 512 bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• the source_key_identifier contains a DSS private key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA keys with moduli greater than 1024-bit length are not supported.</td>
</tr>
<tr>
<td>PCI Cryptographic Coprocessor</td>
<td></td>
<td>The request will be processed on the PCICC when</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• the source_key_identifier contains an RSA modulus-exponent private key with a modulus length of a least 512 bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• the source_key_identifier contains an RSA CRT private key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA keys with moduli greater than 2048-bit length are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td>DSS tokens are not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td>RSA keys with moduli greater than 2048-bit length are not supported.</td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>DSS tokens are not supported.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>DSS tokens are not supported.</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC).</td>
</tr>
</tbody>
</table>

PKA Key Token Build (CSNDPKB and CSNFPKB)

Use this callable service to build external PKA key tokens containing unenciphered private RSA or DSS keys, or public RSA or DSA keys. This callable service is used to create the following:

- A skeleton_key_token for use with the PKA Key Generate callable service (see Table 156 on page 393).
- A key token with a public key that has been obtained from another source
- A key token with a clear private-key and the associated public key
- A key token for an RSA private key in optimized Chinese Remainder Theorem (CRT) form.
- An RSA token with X'09' section identifier using the RSAMEVAR keyword to obtain a token for a key in modulus-exponent form that is variable length.
PKA Key Token Build (CSNDPKB and CSNFPKB)

DSS key generation requires this information in the input skeleton token:
- Size of modulus p in bits
- Prime modulus p
- Prime divisor q
- Public generator g
- Optionally, the private key name

Note: DSS standards define restrictions on the prime modulus p, prime divisor q, and public generator g. (Refer to the Federal Information Processing Standard (FIPS) Publication 186 for DSS standards.) This callable service does not verify all of these restrictions. If you do not follow the restrictions, the keys you generate may not be valid DSS keys.

Restriction: DSS is not supported on a PCIXCC, CEX2C, or CEX3C. PKA key token build will still build DSS tokens, but they cannot be used in any other service on the z890, z990, z9 EC, z9 BC, z10 EC and z10 BC.

This callable service supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSNFPKB.

Format

```
CALL CSNDPKB(
    return_code,  
    reason_code,  
    exit_data_length,  
    exit_data,  
    rule_array_count,  
    rule_array,  
    key_value_structure_length,  
    key_value_structure,  
    private_key_name_length,  
    private_key_name,  
    reserved_1_length,  
    reserved_1,  
    reserved_2_length,  
    reserved_2,  
    reserved_3_length,  
    reserved_3,  
    reserved_4_length,  
    reserved_4,  
    reserved_5_length,  
    reserved_5,  
    key_token_length,  
    key_token)
```

Parameters

**return_code**
Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**
Direction: Output  
Type: Integer
PKA Key Token Build (CSNDPKB and CSNFPKB)

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**
Direction: Ignored
Type: Integer

Reserved field.

**exit_data**
Direction: Input/Output
Type: String

Reserved field.

**rule_array_count**
Direction: Input
Type: Integer

The number of keywords you supplied in the rule_array parameter. Value must be 1, 2 or 3.

**rule_array**
Direction: Input
Type: String

One or two keywords that provide control information to the callable service. Table 159 lists the keywords. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key Type (required)</strong></td>
<td></td>
</tr>
<tr>
<td>DSS-PRIV</td>
<td>This keyword indicates building a key token containing both public and private DSS key information. The parameter key_value_structure identifies the input key values, if supplied.</td>
</tr>
<tr>
<td>DSS-PUBL</td>
<td>This keyword indicates building a key token containing public DSS key information. The parameter key_value_structure identifies the input key values, if supplied.</td>
</tr>
<tr>
<td>RSA-CRT</td>
<td>This keyword indicates building a token containing an RSA private key in the optimized Chinese Remainder Theorem (CRT) form. The parameter key_value_structure identifies the input key values, if supplied.</td>
</tr>
<tr>
<td>RSA-PRIV</td>
<td>This keyword indicates building a token containing both public and private RSA key information. The parameter key_value_structure identifies the input key values, if supplied.</td>
</tr>
<tr>
<td>RSA-PUBL</td>
<td>This keyword indicates building a token containing public RSA key information. The parameter key_value_structure identifies the input values, if supplied.</td>
</tr>
</tbody>
</table>
### Table 159. Keywords for PKA Key Token Build Control Information (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSAMEVAR</td>
<td>This keyword is for creating a key token for an RSA public and private key pair in modulus-exponent form whose modulus is 512 bits or greater.</td>
</tr>
</tbody>
</table>

#### Key Usage Control (optional)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY-MGMT</td>
<td>Indicates that an RSA private key can be used in both the symmetric key import and the digital signature generate callable services.</td>
</tr>
<tr>
<td>KM-ONLY</td>
<td>Indicates that an RSA private key can be used only in symmetric key distribution.</td>
</tr>
<tr>
<td>SIG-ONLY</td>
<td>Indicates that an RSA private key cannot be used in symmetric key distribution. This is the default. Note that for DSS-PRIV the keyword is allowed but extraneous; DSS keys are defined only for digital signature.</td>
</tr>
</tbody>
</table>

#### Translate Control (optional)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>XLATE-OK</td>
<td>Specifies that the private key material can be translated. XLATE-OK is only allowed with key types RSA-PRIV, RSAMEVAR, RSA-CRT and is valid with all key usage rules.</td>
</tr>
<tr>
<td>NO-XLATE</td>
<td>Indicates key translation is not allowed. This is the default. NO-XLATE is only allowed with key types RSA-PRIV, RSAMEVAR, RSA-CRT and is valid with all key usage rules.</td>
</tr>
</tbody>
</table>

### key_value_structure_length

**Direction:** Input  
**Type:** Integer

This is a segment of contiguous storage containing a variable number of input clear key values. The length depends on the key type parameter in the rule array and on the actual values input. The length is in bytes.

#### Table 160. Key Value Structure Length Maximum Values for Key Types

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Key Value Structure Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSS-PRIV</td>
<td>436</td>
</tr>
<tr>
<td>DSS-PUBL</td>
<td>416</td>
</tr>
<tr>
<td>RSA-CRT</td>
<td>3500</td>
</tr>
<tr>
<td>RSAMEVAR</td>
<td>3500</td>
</tr>
<tr>
<td>RSA-PRIV</td>
<td>648</td>
</tr>
<tr>
<td>RSA-PUBL</td>
<td>520</td>
</tr>
</tbody>
</table>

### key_value_structure

**Direction:** Input  
**Type:** String

This is a segment of contiguous storage containing a variable number of input clear key values and the lengths of these values in bits or bytes, as specified. The structure elements are ordered, of variable length, and the input key values must be right-justified within their respective structure elements and padded on
the left with binary zeros. If the leading bits of the modulus are zero’s, don’t
count them in the length. Table 161 defines the structure and contents as a
function of key type.

Table 161. Key Value Structure Elements for PKA Key Token Build

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key Value Structure (Optimized RSA, Chinese Remainder Theorem form, RSA-CRT)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>002</td>
<td>Modulus length in bits (512 to 4096). This is required.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Modulus field length in bytes, “nnn.” This value can be zero if the key token is used as a skeleton_key_token in the PKA key generate callable service. This value must not exceed 512.</td>
</tr>
<tr>
<td>004</td>
<td>002</td>
<td>Public exponent field length in bytes, “eee.” This value can be zero if the key token is used as a skeleton_key_token in the PKA key generate callable service.</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>008</td>
<td>002</td>
<td>Length of the prime number, p, in bytes, “ppp.” This value can be zero if the key token is used as a skeleton_key_token in the PKA key generate callable service. Maximum size of p + q is 512 bytes.</td>
</tr>
<tr>
<td>010</td>
<td>002</td>
<td>Length of the prime number, q, in bytes, “qqq.” This value can be zero if the key token is used as a skeleton_key_token in the PKA key generate callable service. Maximum size of p + q is 512 bytes.</td>
</tr>
<tr>
<td>012</td>
<td>002</td>
<td>Length of d_p, in bytes, “rrr.” This value can be zero if the key token is used as a skeleton_key_token in the PKA key generate callable service. Maximum size of d_p + d_q is 512 bytes.</td>
</tr>
<tr>
<td>014</td>
<td>002</td>
<td>Length of d_q, in bytes, “sss.” This value can be zero if the key token is used as a skeleton_key_token in the PKA key generate callable service. Maximum size of d_p + d_q is 512 bytes.</td>
</tr>
<tr>
<td>016</td>
<td>002</td>
<td>Length of U, in bytes, “uuu.” This value can be zero if the key token is used as a skeleton_key_token in the PKA key generate callable service. Maximum size of U is 512 bytes.</td>
</tr>
<tr>
<td>018</td>
<td>nnn</td>
<td>Modulus, n.</td>
</tr>
</tbody>
</table>
PKA Key Token Build (CSNDPKB and CSNFPKB)

Table 161. Key Value Structure Elements for PKA Key Token Build (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>018 + nnn</td>
<td>eee</td>
<td>Public exponent, e. This is an integer such that 1 &lt; e &lt; n. e must be odd. When you are building a skeleton_key_token to control the generation of an RSA key pair, the public key exponent can be one of these values: 3, 65537 ((2^{16} + 1)), or 0 to indicate that a full random exponent should be generated. The exponent field can be a null-length field if the exponent value is 0.</td>
</tr>
<tr>
<td>018 + nnn + eee</td>
<td>ppp</td>
<td>Prime number, p.</td>
</tr>
<tr>
<td>018 + nnn + eee + ppp</td>
<td>qqq</td>
<td>Prime number, q.</td>
</tr>
<tr>
<td>018 + nnn + eee + ppp + qqq</td>
<td>rrr</td>
<td>(d_p = d \mod(p-1)).</td>
</tr>
<tr>
<td>018 + nnn + eee + ppp + qqq + rrr</td>
<td>sss</td>
<td>(d_q = d \mod(q-1)).</td>
</tr>
<tr>
<td>018 + nnn + eee + ppp + qqq + rrr + sss</td>
<td>uuu</td>
<td>(U = q^{-1} \mod(p)).</td>
</tr>
</tbody>
</table>

Key Value Structure (RSA Private, RSA Private variable or RSA Public)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>002</td>
<td>Modulus length in bits. This is required. When building a skeleton token, the modulus length in bits must be greater than or equal to 512 bits.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Modulus field length in bytes, “XXX”. This value can be zero if you are using the key token as a skeleton in the PKA key generate verb. This value must not exceed 512 when either the RSA-PUBL or RSAMEVAR keyword is used, and must not exceed 128 when the RSA-PRIV keyword is used. This service can build a key token for a public RSA key with a 4096-bit modulus length, or it can build a key token for a 1024-bit modulus length private key.</td>
</tr>
</tbody>
</table>
### PKA Key Token Build (CSNDPKB and CSNFPKB)

#### Table 161. Key Value Structure Elements for PKA Key Token Build (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>004</td>
<td>002</td>
<td>Public exponent field length in bytes, “YYY”. This value must not exceed 512 when either the RSA-PUBL or RSAMEVAR keyword is used, and must not exceed 128 when the RSA-PRIV keyword is used. This value can be zero if you are using the key token as a skeleton token in the PKA key generate verb. In this case, a random exponent is generated. To obtain a fixed, predetermined public key exponent, you can supply this field and the public exponent as input to the PKA key generate verb.</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>Private exponent field length in bytes, “ZZZ”. This field can be zero, indicating that private key information is not provided. This value must not exceed 128 bytes. This value can be zero if you are using the key token as a skeleton token in the PKA key generate verb.</td>
</tr>
<tr>
<td>008</td>
<td>XXX</td>
<td>Modulus, n. This is an integer such that $1 &lt; n &lt; 2^{2048}$. The n is the product of p and q for primes p and q.</td>
</tr>
<tr>
<td>008 + XXX</td>
<td>YYY</td>
<td>RSA public exponent, e. This is an integer such that $1 &lt; e &lt; n$. e must be odd. When you are building a skeleton_key_token to control the generation of an RSA key pair, the public key exponent can be one of these values: 3, 65537 ($2^{16} + 1$), or 0 to indicate that a full random exponent should be generated. The exponent field can be a null-length field if the exponent value is 0.</td>
</tr>
<tr>
<td>008 + XXX + YYY</td>
<td>ZZZ</td>
<td>RSA secret exponent d. This is an integer such that $1 &lt; d &lt; n$. The value of d is $e^{-1} \mod(p-1)(q-1)$. This can be a null-length field if you are using the key token as a skeleton token in the PKA key generate verb.</td>
</tr>
</tbody>
</table>

#### Key Value Structure (DSS Private or DSS Public)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>002</td>
<td>Modulus length in bits. This is required.</td>
</tr>
</tbody>
</table>
## PKA Key Token Build (CSNDPKB and CSNFPKB)

**Table 161. Key Value Structure Elements for PKA Key Token Build (continued)**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>002</td>
<td>002</td>
<td>Prime modulus field length in bytes, “XXX”. You can supply this as a network quantity to the ICSF PKA key generate callable service, which uses the quantity to generate DSS keys. The maximum allowed value is 128.</td>
</tr>
<tr>
<td>004</td>
<td>002</td>
<td>Prime divisor field length in bytes, “YYY”. You can supply this as a network quantity to the ICSF PKA key generate callable service, which uses the quantity to generate DSS keys. The allowed values are 0 or 20 bytes.</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>Public generator field length in bytes, “ZZZ”. You can supply this in a skeleton token as a network quantity to the ICSF PKA key generate callable service, which uses the quantity to generate DSS keys. The maximum allowed value is 128 bytes and is exactly the same length as the prime modulus.</td>
</tr>
<tr>
<td>008</td>
<td>002</td>
<td>Public key field length in bytes, “AAA”. This field can be zero, indicating that the ICSF PKA key generate callable service generates a value at random from supplied or generated network quantities. The maximum allowed value is 128 bytes and is exactly the same length as the prime modulus.</td>
</tr>
<tr>
<td>010</td>
<td>002</td>
<td>Secret key field length in bytes, “BBB”. This field can be zero, indicating that the ICSF PKA key generate callable service generates a value at random from supplied or generated network quantities. The allowed values are 0 or 20 bytes.</td>
</tr>
<tr>
<td>012</td>
<td>XXX</td>
<td>DSS prime modulus p. This is an integer such that (2^{i-1}&lt;p&lt;2^i). The p must be prime. You can supply this value in a skeleton token as a network quantity; it is used in the algorithm that generates DSS keys.</td>
</tr>
<tr>
<td>012 + XXX</td>
<td>YYY</td>
<td>DSS prime divisor q. This is an integer that is a prime divisor of p-1 and (2^{159}&lt;q&lt;2^{160}). You can supply this value in a skeleton token as a network quantity; it is used in the algorithm that generates DSS keys.</td>
</tr>
</tbody>
</table>
### Table 161. Key Value Structure Elements for PKA Key Token Build (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>012 + XXX+ YYY</td>
<td>ZZZ</td>
<td>DSS public generator g. This is an integer such that 1&lt;g&lt;p. You can supply this value in a skeleton token as a network quantity; it is used in the algorithm that generates DSS keys.</td>
</tr>
<tr>
<td>012 + XXX+ YYY+ ZZZ</td>
<td>AAA</td>
<td>DSS public key y. This is an integer such that y=g x mod p.</td>
</tr>
<tr>
<td>012 + XXX+ YYY+ ZZZ+ AAA</td>
<td>BBB</td>
<td>DSS secret private key x. This is an integer such that 0&lt;x&lt;q. The x is random. You need not supply this value if you specify DSS-PUBL in the rule array.</td>
</tr>
</tbody>
</table>

**Notes:**
1. All length fields are in binary.
2. All binary fields (exponent, lengths, modulus, and so on) are stored with the high-order byte field first. This integer number is right-justified within the key structure element field.
3. You must supply all values in the structure to create a token containing an RSA or DSS private key for input to the PKA key import service.

**private_key_name_length**

- **Direction:** Input
- **Type:** Integer

The length can be 0 or 64.

**private_key_name**

- **Direction:** Input
- **Type:** EBCDIC character

This field contains the name of a private key. The name must conform to ICSF label syntax rules. That is, allowed characters are alphanumeric, national (@,#,$) or period (.). The first character must be alphabetic or national. The name is folded to upper case and converted to ASCII characters. ASCII is the permanent form of the name because the name should be independent of the platform. The name is then cryptographically coupled with clear private key data prior to its encryption of the private key. Because of this coupling, the name can never change when the key token is already imported. The parameter is not valid with key types DSS-PUBL or RSA-PUBL.

**reserved_1_length**

- **Direction:** Input
- **Type:** Integer.

Length in bytes of a reserved parameter. You must set this variable to 0.

**reserved_1**

- **Direction:** Input
- **Type:** String

The `reserved_1` parameter identifies a string that is reserved. The service ignores it.
PKA Key Token Build (CSNDPKB and CSNFPKB)

reserved_2_length
Direction: Input
Type: Integer.

Length in bytes of a reserved parameter. You must set this variable to 0.

reserved_2
Direction: Input
Type: String

The reserved_2 parameter identifies a string that is reserved. The service ignores it.

reserved_3_length
Direction: Input
Type: Integer.

Length in bytes of a reserved parameter. You must set this variable to 0.

reserved_3
Direction: Input
Type: String

The reserved_3 parameter identifies a string that is reserved. The service ignores it.

reserved_4_length
Direction: Input
Type: Integer.

Length in bytes of a reserved parameter. You must set this variable to 0.

reserved_4
Direction: Input
Type: String

The reserved_4 parameter identifies a string that is reserved. The service ignores it.

reserved_5_length
Direction: Input
Type: Integer.

Length in bytes of a reserved parameter. You must set this variable to 0.

reserved_5
Direction: Input
Type: String

The reserved_5 parameter identifies a string that is reserved. The service ignores it.

key_token_length
Direction: Input/Output
Type: Integer

Length of the returned key token. The service checks the field to ensure it is at least equal to the size of the token to return. On return from this service, this
PKA Key Token Build (CSNDPKB and CSNFPKB)

The returned key token containing an unenciphered private or public key. The private key is in an external form that can be exchanged with different Common Cryptographic Architecture (CCA) PKA systems. You can use the public key token directly in appropriate ICSF signature verification or key management services.

Usage Notes

If you are building a skeleton for use in a PKA Key Generate request to generate a retained PKA private key, you must build a private key name section in the skeleton token.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 162. PKA key token build required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>None.</td>
<td></td>
</tr>
</tbody>
</table>

PKA Key Token Change (CSNDKTC)

The PKA Key Token Change callable service changes PKA key tokens (RSA and DSS) or trusted block key tokens, from encipherment under the cryptographic coprocessors old Asymmetric-Keys Master Key to encipherment under the current cryptographic coprocessors Asymmetric-Keys Master Key.

- For PKA key tokens - Key tokens must be Private Internal PKA Key Tokens to be changed by this service. PKA private keys encrypted under the Key Management Master Key (KMMK) cannot be reenciphered using this services unless the KMMK has the same value as the Signature Master Key (SMK).
- For trusted block key tokens - Trusted block key tokens must be internal.
PKA Key Token Change (CSNDKTC)

Format

```call
CALL CSNDKTC(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_identifier_length,
    key_identifier)
```

Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the `exit_data` parameter.

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**rule_array_count**

Direction: Input  
Type: Integer

The number of keywords you are supplying in the `rule_array` parameter. The value must be 1.

**rule_array**

Direction: Input  
Type: String

The process rule for the callable service. The keyword must be in 8 bytes of contiguous storage, left-justified and padded on the right with blanks.
PKA Key Token Change (CSNDKTC)

Table 163. Rule Array Keywords for PKA Key Token Change (Required)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| RTCMK   | If the key_identifier is an RSA key token, the service will change an RSA private key from encipherment with the old asymmetric master key to encipherment with the current asymmetric master key.  
If the key_identifier is a trusted block token, the service will change the trusted block’s embedded MAC key from encipherment with the old asymmetric master key to encipherment with the current asymmetric master key. |

key_identifier_length

Direction: Input  
Type: Integer

The length of the key_identifier parameter. The maximum size is 3500 bytes.

key_identifier

Direction: Input/Output  
Type: String

Contains an internal key token of an internal RSA, DSS or trusted block key.  
If the key token is an RSA key token, the private key within the token is securely reenciphered under the current asymmetric master key.  
If the key token is a Trusted Block key token, the MAC key within the token is securely reenciphered under the current asymmetric master key.

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

PKA callable services must be enabled to use the PKA Key Token Change callable service.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 164. PKA key token change required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
</table>
| IBM @server zSeries 800 | PCI Cryptographic Coprocessor | Trusted key blocks are not supported.  
RSA keys with moduli greater than 2048-bit length are not supported. |
| IBM @server zSeries 900 | PCI Cryptographic Coprocessor | Trusted key blocks are not supported.  
RSA keys with moduli greater than 2048-bit length are not supported. |
| IBM @server zSeries 990 | PCI X Cryptographic Coprocessor | Trusted key blocks are not supported.  
RSA keys with moduli greater than 2048-bit length are not supported. |
| IBM @server zSeries 890 | Crypto Express2 Coprocessor | Trusted key blocks are not supported.  
RSA keys with moduli greater than 2048-bit length are not supported. |
| IBM System z9 EC   | Crypto Express2 Coprocessor | RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (LIC). |
| IBM System z9 BC   |                                 |              |
Table 164. PKA key token change required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>RSA key support with moduli within the range 2048-bit to 4096-bit requires the Nov. 2007 or later licensed internal code (Lic).</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PKA Key Translate (CSNDPKT and CSNFPKT)

Use the PKA key translate callable service to translate a source CCA RSA key token into a target external smart card key token.

The source CCA RSA key token must be wrapped with a transport key encrypting key (KEK). The XLATE bit must also be turned on in the key usage byte of the source token. The source token is unwrapped using the specified source transport KEK. The target key token will be wrapped with the specified target transport KEK. Existing information in the target token is overwritten.

There are restrictions on which type key can be used for the source and target transport key tokens. These restrictions are enforced by access control points.

There are restrictions on which rule can be used. These restrictions are enforced by access control points.

This callable service supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSNFPKT.

Format

```
CALL CSNDPKT(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    source_key_identifier_length,
    source_key_identifier,
    source_transport_key_identifier_length,
    source_transport_key_identifier,
    target_transport_key_identifier_length,
    target_transport_key_identifier,
    target_key_token_length
    target_key_token)
```

Parameters

**return_code**

Direction: Output Type: Integer

The return code specifies the general result of the callable service. [Appendix A, "ICSF and TSS Return and Reason Codes"] lists the return codes.
PKA Key Translate (CSNDPKT and CSNFPKT)

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the **exit_data** parameter.

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**rule_array_count**

Direction: Input  
Type: Integer

The number of keywords you supplied in the **rule_array** parameter. Value must be 1.

**rule_array**

Direction: Input  
Type: String

The smartcard format rule for the callable service. A keyword that provides control information to the callable service. See **Table 165** for a list. A keyword is left-justified in an 8-byte field and padded on the right with blanks.

**Table 165. Keywords for PKA Key Generate Rule Array**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Smartcard Format (required)</strong></td>
<td></td>
</tr>
<tr>
<td>SCVISA</td>
<td>This keyword indicates translating the key into the smart card Visa proprietary format.</td>
</tr>
<tr>
<td>SCCOMME</td>
<td>This keyword indicates translating the key into the smart card Modulus-Exponent format.</td>
</tr>
<tr>
<td>SCCOMCRT</td>
<td>This keyword indicates translating the key into the smart card Chinese Remainder Theorem format.</td>
</tr>
</tbody>
</table>

**source_key_identifier_length**

Direction: Input  
Type: Integer

Length in bytes of the **source_key_identifier** variable. The maximum length is 3500 bytes.
PKA Key Translate (CSNDPKT and CSNFPKT)

**source_key_identifier**

Direction: Input  
Type: String

This field contains either a key label identifying an RSA private key or an external public-private key token. The private key must be wrapped with a key encrypting key.

**source_transport_key_identifier_length**

Direction: Input  
Type: Integer

Length in bytes of the source_transport_key_identifier parameter. This value must be 64.

**source_transport_key_identifier**

Direction: Input/Output  
Type: String

This field contains an internal token or label of a DES key-encrypting key. This key is used to unwrap the input RSA key token specified with parameter source_key_identifier. See "Usage Notes" on page 415 for details on the type of transport key that can be used.

**target_transport_key_identifier_length**

Direction: Input  
Type: Integer

Length in bytes of the target_transport_key_identifier parameter. This value must be 64.

**target_transport_key_identifier**

Direction: Input/Output  
Type: String

This field contains an internal token or label of a DES key-encrypting key. This key is used to wrap the output RSA key returned with parameter target_key_token. See "Usage Notes" on page 415 for details on the type of transport key that can be used.

**target_key_token_length**

Direction: Input/Output  
Type: Integer

Length in bytes of the target_key_token parameter. On output, the value in this variable is updated to contain the actual length of the target_key_token produced by the callable service. The maximum length is 3500 bytes.

**target_key_token**

Direction: Output  
Type: String

This field contains the RSA key in the smartcard format specified in the rule array and is protected by the key-encrypting key specified in the target_transport_key parameter. This is not a CCA token, and cannot be stored in the PKDS.
Restrictions

CCA RSA ME tokens will not be translated to the SCCOMCRT format. CCA RSA CRT tokens will not be translated to the SCCOMME format. SCVISA only supports Modulus-Exponent (ME) keys.

Usage Notes

There are access control points that control use of the format rule array keys and the type of transport keys that can be used. All of these access control points are enabled in the default role.

PKA Key Translate - from CCA RSA to SCVISA Format
PKA Key Translate - from CCA RSA to SC ME Format
PKA Key Translate - from CCA RSA to SC CRT Format
PKA Key Translate - from source EXP KEK to target EXP KEK
PKA Key Translate - from source IMP KEK to target EXP KEK
PKA Key Translate - from source IMP KEK to target IMP KEK

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 166. PKA key translate required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required Cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td></td>
<td>Not supported on this platform.</td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td></td>
<td>Not supported on this platform.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Requires the April 2009 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td>Requires the April 2009 or later licensed internal code (LIC).</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

PKA Public Key Extract (CSNDPKX and CSNFPKX)

Use the PKA public key extract callable service to extract a PKA public key token from a supplied PKA internal or external private key token. This service performs no cryptographic verification of the PKA private token. You can verify the private token by using it in a service such as digital signature generate.

This callable service supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSNFPKX.
PKA Public Key Extract (CSNDPKX and CSNFPKX)

Format

```
CALL CSNDPKX(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    source_key_identifier_length,
    source_key_identifier,
    target_public_key_token_length,
    target_public_key_token)
```

Parameters

*return_code*

Direction: Output  Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

*reason_code*

Direction: Output  Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

*exit_data_length*

Direction: Ignored  Type: Integer

Reserved field.

*exit_data*

Direction: Ignored  Type: String

Reserved field.

*rule_array_count*

Direction: Input  Type: Integer

The number of keywords you are supplying in the rule_array parameter. The value must be 0.

*rule_array*

Direction: Input  Type: String

Reserved field. This field is not used, but you must specify it.
PKA Public Key Extract (CSNDPKX and CSNFPKX)

**source_key_identifier_length**
Direction: Input
Type: integer

The length of the `source_key_identifier` parameter. The maximum size is 3500 bytes. When the `source_key_identifier` parameter is a key label, this field specifies the length of the label.

**source_key_identifier**
Direction: Input/output
Type: string

The internal or external token of a PKA private key or the label of a PKA private key. This can be the input or output from PKA key import or from PKA key generate.

This service supports the RSA private key token formats supported on the PCICC, PCIXCC, CEX2C, or CEX3C. If the `source_key_identifier` specifies a label for a private key that has been retained within a PCICC, PCIXCC, CEX2C, or CEX3C, this service extracts only the public key section of the token.

**target_public_key_token_length**
Direction: Input/Output
Type: Integer

The length of the `target_public_key_token` parameter. The maximum size is 3500 bytes. On output, this field will be updated with the actual byte length of the `target_public_key_token`.

**target_public_key_token**
Direction: Output
Type: String

This field contains the token of the extracted PKA public key.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the PKDS.

This service extracts the public key from the internal or external form of a private key. However, it does not check the cryptographic validity of the private token.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PKA Public Key Extract (CSNDPKX and CSNFPKX)

Table 167. PKA public key extract build required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 990</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

PKDS Record Create (CSNDKRC and CSNFKRC)

This callable service writes a new record to the PKDS.

This callable service supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSNFKRC.

Format

```call csndkrc(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    label,
    token_length,
    token)
```

Parameters

**return_code**

Direction: Output Type: Integer

The *return code* specifies the general result of the callable service. [Appendix A, “ICSF and TSS Return and Reason Codes”](#) lists the return codes.

**reason_code**

Direction: Output Type: Integer

The *reason code* specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. [Appendix A, “ICSF and TSS Return and Reason Codes”](#) lists the reason codes.
PKDS Record Create (CSNDKRC and CSNFKRC)

**exit_data_length**
- **Direction:** Input/Output
- **Type:** Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the `exit_data` parameter.

**exit_data**
- **Direction:** Input/Output
- **Type:** String

The data that is passed to the installation exit.

**rule_array_count**
- **Direction:** Input
- **Type:** Integer

The number of keywords you are supplying in the `rule_array` parameter. This parameter is ignored by ICSF.

**rule_array**
- **Direction:** Input
- **Type:** String

This parameter is ignored by ICSF.

**label**
- **Direction:** Input
- **Type:** String

The label of the record to be created. A 64 byte character string.

**token_length**
- **Direction:** Input
- **Type:** Integer

The length of the field containing the token to be written to the PKDS. If zero is specified, a null token will be added to the PKDS. The maximum value of `token_length` is the maximum length of a private RSA or DSS token.

**token**
- **Direction:** Input
- **Type:** String

Data to be written to the PKDS if `token_length` is non-zero. A RSA or DSS private token in either external or internal format, or a DSS or RSA public token.

**Usage Notes**
PKA callable services must be enabled for you to use this service.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
PKDS Record Create (CSNDKRC and CSNFKRC)

Table 168. PKDS record create required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>None.</td>
<td></td>
</tr>
</tbody>
</table>

PKDS Record Delete (CSNDKRD and CSNFKRD)

Use PKDS record delete to delete a record from the PKDS.

This callable service supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSNFKRD.

Format

```call csndkrd(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    label)
```

Parameters

**return_code**

Direction: Output

Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

Direction: Output

Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.
PKDS Record Delete (CSNDKRD and CSNFKRD)

exit_data_length
Direction: Input/Output   Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

exit_data
Direction: Input/Output   Type: String

The data that is passed to the installation exit.

rule_array_count
Direction: Input   Type: Integer

The number of keywords you are supplying in the rule_array parameter. This value must be 0, or 1.

rule_array
Direction: Input   Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deletion Mode (optional)</td>
<td>specifies whether the record is to be deleted entirely or whether only its contents are to be erased.</td>
</tr>
<tr>
<td>LABEL-DL</td>
<td>Specifies that the record will be deleted from the PKDS entirely. This is the default deletion mode.</td>
</tr>
<tr>
<td>TOKEN-DL</td>
<td>Specifies that the only the contents of the record are to be deleted. The record will still exist in the PKDS, but will contain only binary zeroes.</td>
</tr>
</tbody>
</table>

label
Direction: Input   Type: String

The label of the record to be deleted. A 64 byte character string.

Restrictions
This service cannot delete the PKDS record for a retained key.

Usage Notes
PKA callable services must be enabled for you to use this service.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
PKDS Record Delete (CSNDKRD and CSNFKRD)

Table 170. PKDS record delete required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>None.</td>
<td></td>
</tr>
</tbody>
</table>

PKDS Record Read (CSNDKRR)

Reads a record from the PKDS and returns the content of the record. This is true even when the record contains a null PKA token.

Format

```plaintext
CALL CSNDKRR(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    label,
    token_length,
    token)
```

Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. [Appendix A, “ICSF and TSS Return and Reason Codes”](#) lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. [Appendix A, “ICSF and TSS Return and Reason Codes”](#) lists the reason codes.
PKDS Record Read (CSNDKRR)

**exit_data_length**
Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit_data* parameter.

**exit_data**
Direction: Input/Output  Type: String

The data that is passed to the installation exit.

**rule_array_count**
Direction: Input  Type: Integer

The number of keywords you are supplying in the *rule_array* parameter. This parameter is ignored by ICSF.

**rule_array**
Direction: Input  Type: String

This parameter is ignored by ICSF.

**label**
Direction: Input  Type: String

The label of the record to be read. A 64 byte character string.

**token_length**
Direction: Input/Output  Type: Integer

The length of the area to which the record is to be returned. On successful completion of this service, *token_length* will contain the actual length of the record returned.

**token**
Direction: Output  Type: String

Area into which the returned record will be written. The area should be at least as long as the record.

**Usage Notes**

PKA callable services must be enabled for you to use this service.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
**PKDS Record Read (CSNDKRR)**

Table 171. PKDS record read required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>None.</td>
<td></td>
</tr>
</tbody>
</table>

**PKDS Record Write (CSNDKRW)**

Writes over an existing record in the PKDS.

**Format**

```c
CALL CSNDKRW(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    label,
    token_length,
    token)
```

**Parameters**

**return_code**

Direction: Output  
Type: Integer

The `return_code` specifies the general result of the callable service. [Appendix A, “ICSF and TSS Return and Reason Codes”](#) lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. [Appendix A, “ICSF and TSS Return and Reason Codes”](#) lists the reason codes.
PKDS Record Write (CSNDKRW)

**exit_data_length**
Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**
Direction: Input/Output  Type: String

The data that is passed to the installation exit.

**rule_array_count**
Direction: Input  Type: Integer

The number of keywords you are supplying in the rule_array parameter. Its value must be 0 or 1.

**rule_array**
Direction: Input  Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 172. Keywords for PKDS Record Write

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write Mode (optional)</td>
<td>specifies the circumstances under which the record is to be written.</td>
</tr>
<tr>
<td>CHECK</td>
<td>Specifies that the record will be written only if a record of type NULL with the same label exists in the PKDS. If such a record exists, ICSF overwrites it. This is the default condition.</td>
</tr>
<tr>
<td>OVERLAY</td>
<td>Specifies that the record will be overwritten regardless of the current content of the record. If a record with the same label exists in the PKDS, ICSF overwrites it.</td>
</tr>
</tbody>
</table>

**label**
Direction: Input  Type: String

The label of the record to be overwritten. A 64 byte character string.

**token_length**
Direction: Input  Type: Integer

The length of the field containing the token to be written to the PKDS.

**token**
Direction: Input  Type: String
PKDS Record Write (CSNDKRW)

The data to be written to the PKDS, which is a DSS or RSA private token in either external or internal format, or a DSS or RSA public token.

Restrictions

This service cannot update a PKDS record for a retained key.

Usage Notes

PKA callable services must be enabled for you to use this service.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 173. PKDS record write required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>None.</td>
<td></td>
</tr>
</tbody>
</table>

Retained Key Delete (CSNDRKD and CSNFRKD)

Use the retained key delete callable service to delete a key that has been retained within the PCICC, PCIXCC, CEX2C, or CEX3C. This service also deletes the record that contains the associated key token from the PKDS. It also allows the deletion of a retained key in the PCICC, PCIXCC, CEX2C, or CEX3C even if there isn't a PKDS record, or deletion of a PKDS record for a retained key even if the PCICC, PCIXCC, CEX2C, or CEX3C holding the retained key is not online. Use the rule_array parameter specifying the FORCE keyword and serial number of the PCICC, PCIXCC, CEX2C, or CEX3C that contains the retained key to be deleted. If a PKDS record exists for the same label, but the serial number doesn't match the serial number in rule_array, the service will fail. If any applications still need the public key, use public key extract to create a public key token prior to deletion of the retained key.

This callable service supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSNFRKD.
Retained Key Delete (CSNDRKD and CSNFRKD)

Format

```c
CALL CSNDRKD(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_label)
```

Parameters

**return_code**
- **Direction:** Output
- **Type:** Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**
- **Direction:** Output
- **Type:** Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**
- **Direction:** Input/Output
- **Type:** Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the **exit_data** parameter.

**exit_data**
- **Direction:** Input/Output
- **Type:** String

The data that is passed to the installation exit.

**rule_array_count**
- **Direction:** Input
- **Type:** Integer

The number of keywords supplied in the **rule_array** parameter. The value may be 0 or 2.

**rule_array**
- **Direction:** Input
- **Type:** Character String

This parameter may be FORCE and the PCICC, PCIXCC, CEX2C, or CEX3C serial number.

**key_label**
- **Direction:** Input
- **Type:** String

A 64-byte label of a key that has been retained in a PCICC, PCIXCC, CEX2C, or CEX3C.
Retained Key Delete (CSNDRKD and CSNFRKD)

Usage Notes

ICSF calls the Security Server (RACF) to check authorization to use the Retained Key Delete service and the label of the key specified in key_label.

Retained private keys are domain-specific. Only the LPAR domain that created a Retained private key can delete the key via the Retained Key Delete service.

When a Retained key is deleted using the Retained Key Delete service, ICSF records this event in a type 82 SMF record with a subtype of 15.

If the Retained key does not exist in the PCICC, PCIXCC, CEX2C, or CEX3C and the PKDS record exists and the domain that created the retained key matches the domain of the requestor, ICSF deletes the PKDS record. This situation may occur if the PCICC, PCIXCC, CEX2C, or CEX3C has been zeroized through TKE or the service processor.

If a PKDS record containing the retained key exists but the PCICC, PCIXCC, CEX2C, or CEX3C holding the retained key is not online, ICSF deletes the PKDS record if the FORCE keyword is specified. The serial number specified in the rule array must be the serial number of the coprocessor where the Retained key was created. The key token in the PKDS record contains this serial number, and the serial number is used to verify that the PKDS record can be deleted.

If the retained key exists on the specified PCICC, PCIXCC, CEX2C, or CEX3C but there is no corresponding PKDS record, ICSF deletes the retained key from the PCICC, PCIXCC, CEX2C, or CEX3C if the FORCE keyword is specified.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 174. Retained key delete required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>PCI Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>
Retained Key List (CSNDRKL and CSNFRKL)

Use the retained key list callable service to list the key labels of those keys that have been retained within all currently active PCICCs, PCIXCCs, CEX2Cs, or CEX3Cs.

This callable service supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSNFRKL.

Format

```call CSNDRKL(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  rule_array_count,
  rule_array,
  key_label_mask,
  retained_keys_count,
  key_labels_count,
  key_labels
)```

Parameters

**return_code**
- Direction: Output
- Type: Integer
  - The return code specifies the general result of the callable service. [Appendix A, "ICSF and TSS Return and Reason Codes"] lists the return codes.

**reason_code**
- Direction: Output
- Type: Integer
  - The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. [Appendix A, "ICSF and TSS Return and Reason Codes"] lists the reason codes.

**exit_data_length**
- Direction: Input/Output
- Type: Integer
  - The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**
- Direction: Input/Output
- Type: String
  - The data that is passed to the installation exit.

**rule_array_count**
- Direction: Input
- Type: Integer
Retained Key List (CSNDRKL and CSNFRKL)

The number of keywords supplied in the rule_array parameter. The value must be 0.

**rule_array**

Direction: Input  
Type: Character String

This parameter is ignored by ICSF.

**key_label_mask**

Direction: Input  
Type: String

A 64-byte key label mask that is used to filter the list of key names returned by the verb. You can use a wild card (*) to identify multiple keys retained within the PCICC, PCIXCC, CEX2C, or CEX3C.

**Note:** If an asterisk (*) is used, it must be the last character in key_label_mask. There can only be one *.

**retained_keys_count**

Direction: Output  
Type: Integer

An integer variable to receive the number of retained keys stored within all active PCICCs, PCIXCCs, CEX2Cs, and CEX3Cs.

**key_labels_count**

Direction: Input/Output  
Type: Integer

On input this variable defines the maximum number of key labels to be returned. On output this variable defines the total number of key labels returned. The maximum value for this field is 100. The value returned in the retained_keys_count variable can be larger if you have not provided for the return of a sufficiently large number of key labels in the key_labels_count field.

**key_labels**

Direction: Output  
Type: String

A string variable where the key label information will be returned. This field must be at least 64 times the key label count value. The key label information is a string of zero or more 64-byte entries. The first 64-byte entry contains a PCICC, PCIXCC, CEX2C, or CEX3C card serial number, and is followed by one or more 64-byte entries that each contain a key label of a key retained within that PCICC, PCIXCC, CEX2C, or CEX3C. The format of the first 64-byte entry is as follows:

```
/nnnnnnnnnbbb...bbb
```

where

- /* is the character /* (EBCDIC: X'61')
- "nnnnnnnn" is the 8-byte PCICC, PCIXCC, CEX2C, or CEX3C card serial number
- "bbb...bbb" is 55 bytes of blank pad characters (EBCDIC: X'40')

This information (64-byte card serial number entry followed by one or more 64-byte label entries) is repeated for each active PCICC, PCIXCC, CEX2C, or CEX3C that contains retained keys that match the key_label_mask. All data returned is EBCDIC characters. The number of bytes of information returned is
Retained Key List (CSNDRKL and CSNFRKL)

The key_labels field must be large enough to hold the number of 64-byte labels specified in the key_labels_count field plus one 64-byte entry for each active PCICC, PCIXCC, CEX2C, or CEX3C (a maximum of 64 PCICCs, PCIXCCs, CEX2Cs, or CEX3Cs).

Usage Notes

Not all CCA platforms may support multiple PCICC, PCIXCC, CEX2C, or CEX3C cards. In the case where only one card is supported, the key_labels field will contain one or more 64-byte entries that each contain a key label of a key retained within the PCICC, PCIXCC, CEX2C, or CEX3C. There will be no 64-byte entry or entries containing a PCICC, PCIXCC, CEX2C, or CEX3C card serial number.

ICSF calls RACF to check authorization to use the Retained Key List service.

ICSF caller must be authorized to the key_label_mask name including the *.

Retained private keys are domain-specific. ICSF lists only those keys that were created by the LPAR domain that issues the Retained Key List request.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 175. Retained key list required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>PCI Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>PCI X Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>
Retained Key List (CSNDRKL and CSNFRKL)
This topic describes these callable services:

- "Character/Nibble Conversion (CSNBXBC and CSNBXCB)"
- "Code Conversion (CSNBXEA and CSNBXAE)" on page 435
- "ICSF Query Algorithm (CSFIQA and CSFIQA6)" on page 437
- "ICSF Query Facility (CSFIQF and CSFIQF6)" on page 442
- "X9.9 Data Editing (CSNB9ED)" on page 459

Note: These services are not dependent on the hardware. They will run on any server.

Character/Nibble Conversion (CSNBXBC and CSNBXCB)

Use these utilities to convert a binary string to a character string (CSNBXBC) or convert a character string to a binary string (CSNBXCB).

Format

```
CALL CSNBXBC(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    text_length,
    source_text,
    target_text,
    code_table)
```

```
CALL CSNBXCB(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    text_length,
    source_text,
    target_text,
    code_table)
```

Parameters

- **return_code**
  
  Direction: Output  
  Type: Integer

  The return code specifies the general result of the callable service. [Appendix A, ICSF and TSS Return and Reason Codes](https://www.ibm.com/support/docview/cn/en/3714891) lists the return codes.

- **reason_code**
  
  Direction: Output  
  Type: Integer

  The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned...
Character/Nibble Conversion (CSNBXBC and CSNBXCB)

Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

exit_data_length
Direction: Ignored Type: Integer
Reserved field.

exit_data
Direction: Ignored Type: String
Reserved field.

text_length
Direction: Input/Output Type: Integer
On input, the text_length contains an integer that is the length of the source_text. The length must be a positive nonzero value. On output, text_length is updated with an integer that is the length of the target_text.

source_text
Direction: Input Type: String
This parameter contains the string to convert.

target_text
Direction: Output Type: String
The converted text that the callable service returns.

code_table
Direction: Input Type: String
A 16-byte conversion table. The code table for binary to EBCDIC conversion is X'F0F1F2F3F4F5F6F7F8F9C1C2C3C4C5C6'.

Usage Notes

These services are structured differently from the other services. They run in the caller's address space in the caller's key and mode.

ICSF need not be active for you to run either of these services. No pre- or post-processing exits are enabled for these services, and no calls to RACF are issued when you run these services.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.
Table 176. Character/Nibble conversion required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Code Conversion (CSNBXBC and CSNBXCB)**

Use these utilities to convert ASCII data to EBCDIC data (CSNBXAE) or EBCDIC data to ASCII data (CSNBXEA).

**Format**

```c
CALL CSNBXAE(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    text_length,
    source_text,
    target_text,
    code_table)
```

```c
CALL CSNBXEA(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    text_length,
    source_text,
    target_text,
    code_table)
```

**Parameters**

- **return_code**
  - Direction: Output
  - Type: Integer
Code Conversion (CSNBXEA and CSNBXAE)

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code
Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

exit_data_length
Direction: Ignored Type: Integer

Reserved field.

exit_data
Direction: Ignored Type: String

Reserved field.

text_length
Direction: Input Type: Integer

The text_length contains an integer that is the length of the source_text. The length must be a positive nonzero value.

source_text
Direction: Input Type: String

This parameter contains the string to convert.

target_text
Direction: Output Type: String

The converted text that the callable service returns.

code_table
Direction: Input Type: String

A 256-byte conversion table. To use the default code table, you need to pass a full word of hexadecimal zero's. See Appendix G, "EBCDIC and ASCII Default Conversion Tables," on page 701 for contents of the default table.

Note: The Transaction Security System code table has 2 additional 8-byte fields that are not used in the conversion process. ICSF accepts either a 256-byte or a 272-byte code table, but uses only the first 256 bytes in the conversion.

Usage Notes

These services are structured differently than the other services. They run in the caller's address space in the caller's key and mode. ICSF need not be active for
you to run either of these services. No pre- or post-processing exits are enabled for these services, and no calls to RACF are issued when you run these services.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>None.</td>
<td></td>
</tr>
</tbody>
</table>

**ICSF Query Algorithm (CSFIQA and CSFIQA6)**

Use this utility to retrieve information about the cryptographic and hash algorithms available. You can control the amount of data that is returned by passing in `rule_array` keywords. Keyword values describe the cryptographic algorithm or hash algorithm you are interested in.

The service returns a table of information in the `returned_data` parameter. A row of data consists of the algorithm name, the algorithm size, whether or not clear or secure keys are supported and what method ICSF will use to satisfy a request - CPU instructions, a cryptographic accelerator, a cryptographic coprocessor, or software. The service updates the `returned_data_length` field with the actual length of the output `returned_data` field.

This callable service supports invocation in AMODE (64). The callable service name for AMODE (64) invocation is CSFIQA6.
ICSF Query Algorithm (CSFIQA and CSFIQA6)

Format

```c
CALL CSFIQA(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    returned_data_length,
    returned_data,
    reserved_data_length,
    reserved_data)
```

Parameters

**return_code**

Direction: Output  
Type: Integer

The `return_code` specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The `reason_code` specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit_data_length**

Direction: Input  
Type: Integer

The `exit_data_length` is the length of the data that is passed to the installation exit. The data is identified in the `exit_data` parameter. The length must be set to zero.

**exit_data**

Direction: Ignored  
Type: String

The `exit_data` is the data that is passed to the installation exit. Reserved for future use.

**rule_array_count**

Direction: Input  
Type: Integer

The number of keywords you are supplying in `rule_array`. Value must be 0 or 1.

**rule_array**

Direction: Input  
Type: String

The keyword is left-justified in an 8-byte field and padded on the right with blanks. The keyword serves to limit the amount of data returned.
### ICSF Query Algorithm (CSFIQA and CSFIQA6)

#### Table 178. Keywords for ICSF Query Algorithm

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALGORITHM (optional)</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard - symmetric key algorithm</td>
</tr>
<tr>
<td>DES</td>
<td>Data Encryption Standard - single length symmetric key algorithm</td>
</tr>
<tr>
<td>DSS</td>
<td>Data Signature Standard - public key cryptography algorithm</td>
</tr>
<tr>
<td>RSA</td>
<td>Rivest-Shamir-Adleman - public key cryptography algorithm, all usage types</td>
</tr>
<tr>
<td>RSA-SIG</td>
<td>Rivest-Shamir-Adleman - public key cryptography algorithm, signature usage.</td>
</tr>
<tr>
<td>RSA-KM</td>
<td>Rivest-Shamir-Adleman - public key cryptography algorithm, key management usage.</td>
</tr>
<tr>
<td>RSA-GEN</td>
<td>Rivest-Shamir-Adleman - public key cryptography algorithm, key generation.</td>
</tr>
<tr>
<td>SHA-1</td>
<td>Secure Hash Algorithm 1 - A one way hash algorithm</td>
</tr>
<tr>
<td>SHA-2</td>
<td>Secure Hash Algorithm 2 - A one way hash algorithm</td>
</tr>
<tr>
<td>MDC-2</td>
<td>Modification Detection Code 2 - MDC-2 specifies two encipherments per 8 bytes of input text</td>
</tr>
<tr>
<td>MDC-4</td>
<td>Modification Detection Code 4 - MDC-4 specifies four encipherments per 8 bytes of input text</td>
</tr>
<tr>
<td>MD5</td>
<td>Message Digest 5 - A one way hash algorithm</td>
</tr>
<tr>
<td>RPMD-160</td>
<td>RIPE MD-160 - A one way hash algorithm</td>
</tr>
<tr>
<td>RNGL</td>
<td>Random number generate long callable service</td>
</tr>
<tr>
<td>TDES</td>
<td>Data Encryption Standard - double and triple length symmetric key algorithm</td>
</tr>
</tbody>
</table>

**returned_data_length**

**Direction:** Input/Output  
**Type:** Integer

The length of the `returned_data` parameter. Currently, the value must be large enough to handle the request. Allow additional space for future enhancements. On output, this field will contain the actual length of the data returned.

**returned_data**

**Direction:** Output  
**Type:** String

This field will contain the table output from the service. Depending on the contents of `rule_array`, multiple rows may be returned. One row in the table contains:

#### Table 179. Output for ICSF Query Algorithm

<table>
<thead>
<tr>
<th>Offset(hex)</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

Chapter 11. Utilities 439
### Table 179. Output for ICSF Query Algorithm (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'0')</td>
<td><strong>Algorithm</strong>&lt;br&gt;An 8-byte EBCDIC character string containing the name of the cryptographic algorithm. The character string is padded on the right with blanks. Possible values are:&lt;br&gt;AES&lt;br&gt;DES (single length DES)&lt;br&gt;DSS&lt;br&gt;MDC-2&lt;br&gt;MDC-4&lt;br&gt;MD5&lt;br&gt;RNGL&lt;br&gt;RPMD-160&lt;br&gt;RSA-GEN&lt;br&gt;RSA-KM&lt;br&gt;RSA-SIG&lt;br&gt;SHA-1&lt;br&gt;SHA-2&lt;br&gt;TDES (double and triple length DES)</td>
</tr>
<tr>
<td>8 (X'8')</td>
<td><strong>Size</strong>&lt;br&gt;An 8-byte EBCDIC string representing the maximum key, modulus, or hash size. The string is padded with blanks on the right. The size is in bits. This is true for all algorithms except RNGL. For RNGL, the size is in bytes.</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td><strong>Key Security</strong>&lt;br&gt;An 8-byte EBCDIC character string containing the string&lt;br&gt;CLEAR&lt;br&gt;SECURE&lt;br&gt;NA&lt;br&gt;The string is padded on the right with blanks.</td>
</tr>
<tr>
<td>24(X'18')</td>
<td><strong>Implementation</strong>&lt;br&gt;An 8-byte EBCDIC character string containing how the algorithm is implemented. The string is padded on the right with blanks. Possible choices are:&lt;br&gt;ACC - Cryptographic Accelerator&lt;br&gt;CCF - CCF&lt;br&gt;COP - Cryptographic Coprocessor&lt;br&gt;CPU - CPACF&lt;br&gt;SW - Software</td>
</tr>
</tbody>
</table>

The rows are sorted in the following order:<br>- Algorithm name - alphabetically A to Z<br>- Algorithm size - numerically highest to least<br>- Key security - alphabetically A to Z<br>- Implementation - alphabetically A to Z

- **reserved_data_length**
  - Direction: Input
  - Type: Integer
  - The length of the reserved data parameter. Currently, the value must be 0.

- **reserved_data**
  - Direction: Ignored
  - Type: String
Usage Notes

The rule_array keyword allows the caller to select how much information is returned. The returned data can describe all cryptographic support on the base system or it can be filtered by an algorithm.

For example, a rule_array_count of 0 will return information about all algorithms and key security. A rule_array_count of 1 and a keyword of ‘AES’ will return information about the AES algorithm support, both clear and secure AES keys.

Only cryptographic coprocessors in the active state are queried.

In general, a key security of SECURE implies that both SECURE and CLEAR key versions of the algorithm are supported by the processor or the cryptographic coprocessor. The exception is TDES support in CCF on a z800/z900. Only SECURE TDES keys are supported.

This service lists an algorithm as being supported when the cryptographic coprocessor or accelerator is capable of performing the function. It does not reflect when an algorithm is unavailable because TKE was used to disable the function.

RNGL keyword refers to the Random Number Generate Long (CSFBRNGL) callable service. The following is returned for implementation:

- COP - when RNGL is implemented using the RNGL verb in the cryptographic coprocessor.
- CCF- when RNGL is implemented using the CCF random number generate function (z 800/900 machines)
- SW - when RNGL is implemented using a loop around the RNG verb in the cryptographic coprocessor, creating the random number 8 bytes at a time.

When a row of the returned_data table contains a Key Security value of SECURE and an Implementation value of CPU, this indicates that the CSNBSYE and CSNBSYD callable services support the use of key labels for encrypted keys stored in the CKDS. In other words, the required functions in ICSF, CPACF and the cryptographic coprocessor are available.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>None.</td>
<td></td>
</tr>
</tbody>
</table>
ICSF Query Algorithm (CSFIQA and CSFIQA6)

Table 180. ICSF Query Algorithm required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 990</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

ICSF Query Facility (CSFIQF and CSFIQF6)

Use this utility to retrieve information about ICSF, the cryptographic coprocessors and the CCA code in the coprocessors. This information includes:

- general information about ICSF
- general information about CCA code in a coprocessor
- export control information from a coprocessor
- diagnostic information from a coprocessor

Coprocessor information requests may be directed to a specific ONLINE or ACTIVE coprocessor or any ACTIVE coprocessor.

This service has an interface similar to the IBM 4758 service CSUACFQ. Instead of the output being returned in the rule array, there is a separate output area. The format of the data returned remains the same. This service supports a subset of the keywords supported by CSUACFQ. For the same supported keywords, CSFIQF and CSUACFQ return the same coprocessor-specific information. The service returns information elements in the returned_data field and updates the returned_data_length with the actual length of the output returned_data field.

This callable service supports invocation in AMODE(64). The callable service name for AMODE(64) invocation is CSFIQF6.

Format

CALL CSFIQF(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    returned_data_length,
    returned_data,
    reserved_data_length,
    reserved_data)

z/OS V1R11 ICSF Application Programmer's Guide
Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation data.

**rule_array_count**

Direction: Input  
Type: Integer

The number of keywords you are supplying in rule_array. Value must be 1 or 2.

**rule_array**

Direction: Input  
Type: String

Keywords that provide control information to callable services. The keywords are left-justified in an 8-byte field and padded on the right with blanks. The keywords must be in contiguous storage. Specify one or two of the values in Table 181.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Coprocessor (optional) - parameter is ignored for ICSFSTAT.</strong></td>
<td></td>
</tr>
<tr>
<td>COPROCxX</td>
<td>Specifies the specific coprocessor to execute the request. xx may be 00 through 63 inclusive. This may be the processor number of any coprocessor. The processor number of any accelerator is not supported.</td>
</tr>
<tr>
<td>ANY</td>
<td>Process request on any ACTIVE cryptographic coprocessor. This is the default.</td>
</tr>
</tbody>
</table>
### ICSF Query Facility (CSFIQF and CSFIQF6)

**Table 181. Keywords for ICSF Query Service (continued)**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>nnnnnnnn</td>
<td>Specifies the 8-byte serial number of the coprocessor to execute the request.</td>
</tr>
</tbody>
</table>

**Information to return (required)**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICSFSTAT</td>
<td>Get ICSF related status information.</td>
</tr>
<tr>
<td>ICSFST2</td>
<td>Get additional ICSF related status information.</td>
</tr>
<tr>
<td>STAES</td>
<td>Get status information on AES enablement and the AES master key registers.</td>
</tr>
<tr>
<td>STATCCA</td>
<td>Get CCA-related status information.</td>
</tr>
<tr>
<td>STATCCAE</td>
<td>Get CCA-related extended status information.</td>
</tr>
<tr>
<td>STATCARD</td>
<td>Get coprocessor-related basic status information.</td>
</tr>
<tr>
<td>STATDIAG</td>
<td>Get coprocessor-related basic status information.</td>
</tr>
<tr>
<td>STATEID</td>
<td>Get coprocessor-related basic status information.</td>
</tr>
<tr>
<td>STATXPT</td>
<td>Get coprocessor-related basic status information.</td>
</tr>
</tbody>
</table>

**returned_data_length**

- **Direction:** Input/Output
- **Type:** Integer

The length of the `returned_data` parameter. Currently, the value must be at least eight times the number of elements returned for the `rule_array` keyword specified. Allow additional space for future enhancements. On output, this field will contain the actual length of the data returned.

**returned_data**

- **Direction:** Output
- **Type:** String

This field will contain the output from the service. It has the format of 8-byte elements of character data.

The format of the output `returned_data` depends on the value of the input `rule_array` and the information requested. Different information is returned depending on what the input keyword is.

For `returned_data` elements that contain numbers, those numbers are represented by numeric characters which are left-justified and padded on the right with space characters. For example, a `returned_data` element which contains the number two with contain the character string '2   '.

For ICSFSTAT, the coprocessor keyword is ignored. The output `returned_data` for the ICSFSTAT keyword is defined in Table 182.

**Table 182. Output for option ICSFSTAT**

<table>
<thead>
<tr>
<th>Element Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FMID</td>
<td>8-byte ICSF FMID</td>
</tr>
</tbody>
</table>
### ICSF Query Facility (CSFIQF and CSFIQF6)

**Table 182. Output for option ICSFSTAT (continued)**

<table>
<thead>
<tr>
<th>Field</th>
<th>Status Field 1</th>
<th>Status of ICSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>ICSF Status Field 1</td>
<td>Status of ICSF</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>Meaning</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>ICSF started</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>SYM-MK (DES master key) valid (CCVTMK is on)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Status Field 2</th>
<th>Status of ICSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>ICSF Status Field 2</td>
<td>Status of ICSF</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>Meaning</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>64-bit callers not supported</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>64-bit callers supported, and a TKDS has been specified for the storage of persistent PKCS #11 objects.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>CPACF</th>
<th>CPACF availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td>CPACF availability</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>Meaning</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>CPACF not available</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>SHA-224 and SHA-256 are available</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>SHA-224 and SHA-256, DES and TDES are available</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>SHA-384 and SHA-512, DES and TDES are available</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>AES</th>
<th>AES availability for clear keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td>AES availability for clear keys</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>Meaning</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>AES not available</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>AES-128</td>
</tr>
</tbody>
</table>
Table 182. Output for option ICSFSTAT (continued)

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>DSA</td>
<td>DSA algorithm availability</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>Meaning</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>DSA not available</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>DSA 1024 key size</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>DSA 2048 key size</td>
</tr>
<tr>
<td>7</td>
<td>RSA Signature</td>
<td>RSA Signature key length</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>Meaning</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>RSA not available</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>RSA 1024 key size</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>RSA 2048 key size</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>RSA 4096 key size</td>
</tr>
<tr>
<td>8</td>
<td>RSA Key Management</td>
<td>RSA Key Management key length</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>Meaning</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>RSA not available</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>RSA 1024 key size</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>RSA 2048 key size</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>RSA 4096 key size</td>
</tr>
<tr>
<td>9</td>
<td>RSA Key Generate</td>
<td>RSA Key Generate</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>Meaning</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>Service not available</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Service available - 2048 bit modulus</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Service available - 4096 bit modulus</td>
</tr>
<tr>
<td>10</td>
<td>Accelerators</td>
<td>Availability of clear RSA key accelerators (PCICAs)</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>Meaning</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>Not available</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>At least one available for application use.</td>
</tr>
<tr>
<td>11</td>
<td>Future Use</td>
<td>Currently blanks</td>
</tr>
<tr>
<td>12</td>
<td>Future Use</td>
<td>Currently blanks</td>
</tr>
</tbody>
</table>

For ICSFST2 the coprocessor rule array keyword is ignored. The output returned_data for the ICSFST2 keyword is defined in Table 183.

Table 183. Output for option ICSFST2

<table>
<thead>
<tr>
<th>Element Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Version</td>
<td>Version of the ICSFST2 returned_data. Initial value is 1. It covers elements 1 through 12.</td>
</tr>
<tr>
<td>2</td>
<td>FMID</td>
<td>8–byte ICSF FMID.</td>
</tr>
</tbody>
</table>
### ICSF Query Facility (CSFIQF and CSFIQF6)

#### Table 183. Output for option ICSFST2 (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Status Field 1</th>
<th>Status of ICSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>0</td>
<td>PKA callable services disabled</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>PKA callable services enabled (see &quot;Usage Notes&quot; on page 458)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Status Field 2</th>
<th>Status of ICSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>0</td>
<td>PKCS #11 is not available</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>PKCS #11 is available</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Status Field 3</th>
<th>Status of ICSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>0</td>
<td>ICSF started</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>ICSF initialized</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>AES master key valid</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Status Field 4</th>
<th>Status of ICSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>0</td>
<td>Secure key AES not available</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Secure key AES is available</td>
</tr>
</tbody>
</table>
Table 183. Output for option ICSFST2 (continued)

<table>
<thead>
<tr>
<th>Number</th>
<th>ICSF Status Field 5</th>
<th>An 8-character numeric character string summarizing the current Key Store Policy.</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td>The first character in this string indicates if Key Token Authorization Checking controls have been enabled for the CKDS in either warning or fail mode, and, if so, if the Default Key Label Checking control has also been enabled. The numbers that can appear in the first character of this string are:</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>Key Token Authorization Checking is not enabled for the CKDS.</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Key Token Authorization Checking for CKDS is enabled in FAIL mode. <strong>Key Store Policy is active for CKDS.</strong> Default Key Label Checking is not enabled.</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Key Token Authorization Checking for CKDS is enabled in WARN mode. <strong>Key Store Policy is active for CKDS.</strong> Default Key Label Checking is not enabled.</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Key Token Authorization Checking for CKDS is enabled in FAIL mode. <strong>Key Store Policy is active for CKDS.</strong> Default Key Label Checking is also enabled.</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Key Token Authorization Checking for CKDS is enabled in WARN mode. <strong>Key Store Policy is active for CKDS.</strong> Default Key Label Checking is also enabled.</td>
</tr>
</tbody>
</table>

The second character in this string indicates if Duplicate Key Token Checking controls have been enabled for the CKDS. The numbers that can appear in the second character of this string are:

<table>
<thead>
<tr>
<th>Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Duplicate Key Token Checking is not enabled for the CKDS.</td>
</tr>
<tr>
<td>1</td>
<td>Duplicate Key Token Checking is enabled for the CKDS. <strong>Key Store Policy is active for CKDS.</strong></td>
</tr>
</tbody>
</table>
The third character in this string indicates if Key Token Authorization Checking controls have been enabled for the PKDS in either warning or fail mode, and, if so, if the Default Key Label Checking control has also been enabled. The numbers that can appear in the third character of this string are:

<table>
<thead>
<tr>
<th>Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Key Token Authorization Checking is not enabled for the PKDS.</td>
</tr>
<tr>
<td>1</td>
<td>Key Token Authorization Checking for PKDS is enabled in FAIL mode. <strong>Key Store Policy is active for PKDS.</strong> Default Key Label Checking is not enabled.</td>
</tr>
<tr>
<td>2</td>
<td>Key Token Authorization Checking for PKDS is enabled in WARN mode. <strong>Key Store Policy is active for PKDS.</strong> Default Key Label Checking is not enabled.</td>
</tr>
<tr>
<td>3</td>
<td>Key Token Authorization Checking for PKDS is enabled in FAIL mode. <strong>Key Store Policy is active for PKDS.</strong> Default Key Label Checking is also enabled.</td>
</tr>
<tr>
<td>4</td>
<td>Key Token Authorization Checking for PKDS is enabled in WARN mode. <strong>Key Store Policy is active for PKDS.</strong> Default Key Label Checking is also enabled.</td>
</tr>
</tbody>
</table>

The fourth character in this string indicates if Duplicate Key Token Checking controls have been enabled for the PKDS. The numbers that can appear in the fourth character of this string are:

<table>
<thead>
<tr>
<th>Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Duplicate Key Token Checking is not enabled for the PKDS.</td>
</tr>
<tr>
<td>1</td>
<td>Duplicate Key Token Checking is enabled for the PKDS. <strong>Key Store Policy is active for PKDS.</strong></td>
</tr>
</tbody>
</table>
The fifth character in this string indicates if Granular Key Label Access controls have been enabled in WARN or FAIL mode. The numbers that can appear in the fifth character of this string are:

<table>
<thead>
<tr>
<th>Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Granular Key Label Access controls are not enabled.</td>
</tr>
<tr>
<td>1</td>
<td>Granular Key Label Access control is enabled in FAIL mode</td>
</tr>
<tr>
<td>2</td>
<td>Granular Key Label Access control is enabled in WARN mode</td>
</tr>
</tbody>
</table>

The sixth character in this string indicates if Symmetric Key Label Export controls have been enabled for AES and/or DES keys. The numbers that can appear in the sixth character of this string are:

<table>
<thead>
<tr>
<th>Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Symmetric Key Label Export controls are not enabled.</td>
</tr>
<tr>
<td>1</td>
<td>Symmetric Key Label Export control is enabled for DES keys only.</td>
</tr>
<tr>
<td>2</td>
<td>Symmetric Key Label Export control is enabled for AES keys only.</td>
</tr>
<tr>
<td>3</td>
<td>Symmetric Key Label Export controls are enabled for both DES and AES keys.</td>
</tr>
</tbody>
</table>

Table 183. Output for option ICSFST2 (continued)
Table 183. Output for option ICSFST2 (continued)

The seventh character in this string indicates if PKA Key Management Extensions have been enabled in either WARN or FAIL mode, and, if so, whether a SAF key ring or a PKCS #11 token is identified as the trusted certificate repository. (The trusted certificate repository is identified using the APPLDATA field of the CSF.PKAEEXTNS.ENABLE profile. If no value is specified in the APPLDATA field, a PKCS #11 token is assumed.) The numbers that can appear in the seventh character of this string are:

<table>
<thead>
<tr>
<th>Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Symmetric Key Label Export controls are not enabled.</td>
</tr>
<tr>
<td>1</td>
<td>PKA Key Management Extensions control is enabled in FAIL mode. The trusted certificate repository is a SAF key ring.</td>
</tr>
<tr>
<td>2</td>
<td>PKA Key Management Extension control is enabled in FAIL mode. The trusted certificate repository is a PKCS #11 token.</td>
</tr>
<tr>
<td>3</td>
<td>PKA Key Management Extensions control is enabled in WARN mode. The trusted certificate repository is a SAF key ring.</td>
</tr>
<tr>
<td>4</td>
<td>PKA Key Management Extension control is enabled in WARN mode. The trusted certificate repository is a PKCS #11 token.</td>
</tr>
</tbody>
</table>

8 Future use Currently blanks
9 Future use Currently blanks
10 Future use Currently blanks
11 Future use Currently blanks
12 Future use Currently blanks

Table 184. Output for option STATAES

<table>
<thead>
<tr>
<th>Element Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AES NMK Status</td>
<td>State of the AES new master key register:</td>
</tr>
<tr>
<td>Number</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Register is clear</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Register contains a partially complete key</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Register contains a complete key</td>
<td></td>
</tr>
</tbody>
</table>
### Table 184. Output for option STATAES (continued)

<table>
<thead>
<tr>
<th>Number</th>
<th>AES CMK Status</th>
<th>State of the AES current master key register:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Number</td>
<td>Meaning</td>
</tr>
<tr>
<td>2</td>
<td>Register is clear</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Register contains a key</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number</th>
<th>AES OMK Status</th>
<th>State of the AES old master key register:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Number</td>
<td>Meaning</td>
</tr>
<tr>
<td>2</td>
<td>Register is clear</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Register contains a key</td>
<td></td>
</tr>
</tbody>
</table>

| Number | AES key length enablement | The maximum AES key length that is enabled by the function control vector. The value is 0 (if no AES key length is enabled in the FCV), 128, 192, or 256. |

### Table 185. Output for option STATCCA

<table>
<thead>
<tr>
<th>Element Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NMK Status</td>
<td>State of the DES New Master Key Register:</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>Meaning</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Register is clear</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Register contains a partially complete key</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Register contains a complete key</td>
</tr>
</tbody>
</table>

| 2              | CMK Status     | State of the DES Current Master Key Register:                                                                                                                                                            |
|                | Number         | Meaning                                                                                                                                       |
|                | 1              | Register is clear                                                                                                                             |
|                | 2              | Register contains a key                                                                                                                       |

| 3              | OMK Status     | State of the DES Old Master Key Register:                                                                                                                                                                |
|                | Number         | Meaning                                                                                                                                       |
|                | 1              | Register is clear                                                                                                                             |
|                | 2              | Register contains a key                                                                                                                       |

| 4              | CCA Application Version | A character string that identifies the version of the CCA application program that is running in the coprocessor.                                                                                 |

| 5              | CCA Application Build Date | A character string containing the build date for the CCA application program that is running in the coprocessor.                                                                                       |

| 6              | User Role        | A character string containing the Role identifier which defines the host application user's current authority.                                                                                       |

### Table 186. Output for option STATCCAE

<table>
<thead>
<tr>
<th>Element Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
Table 186. Output for option STATCCAE (continued)

<table>
<thead>
<tr>
<th></th>
<th>Symmetric NMK Status</th>
<th>State of the DES Symmetric New Master Key Register:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Number</td>
<td>Meaning</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Register is clear</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Register contains a partially complete key</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Register contains a complete key</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Symmetric CMK Status</th>
<th>State of the DES Symmetric Current Master Key Register:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Number</td>
<td>Meaning</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Register is clear</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Register contains a key</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Symmetric OMK Status</th>
<th>State of the DES Symmetric Old Master Key Register:</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Number</td>
<td>Meaning</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Register is clear</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Register contains a key</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>CCA Application Version</th>
<th>A character string that identifies the version of the CCA application program that is running in the coprocessor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>CCA Application Build Date</th>
<th>A character string containing the build date for the CCA application program that is running in the coprocessor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>User Role</th>
<th>A character string containing the Role identifier which defines the host application user's current authority.</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Asymmetric NMK Status</th>
<th>State of the Asymmetric New Master Key Register:</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Number</td>
<td>Meaning</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Register is clear</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Register contains a partially complete key</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Register contains a complete key</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Asymmetric CMK Status</th>
<th>State of the Asymmetric Current Master Key Register:</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Number</td>
<td>Meaning</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Register is clear</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Register contains a key</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Asymmetric OMK Status</th>
<th>State of the Asymmetric Old Master Key Register:</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Number</td>
<td>Meaning</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Register is clear</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Register contains a key</td>
</tr>
</tbody>
</table>

Table 187. Output for option STATCARD

<table>
<thead>
<tr>
<th>Element Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Chapter 11. Utilities 453
### Table 187. Output for option STATCARD (continued)

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Number of installed adapters</td>
<td>The number of active cryptographic coprocessors installed in the machine. This only includes coprocessors that have CCA software loaded (including those with CCA UDX software).</td>
</tr>
<tr>
<td>2</td>
<td>DES hardware level</td>
<td>A numeric character string containing an integer value identifying the version of DES hardware that is on the coprocessor.</td>
</tr>
<tr>
<td>3</td>
<td>RSA hardware level</td>
<td>A numeric character string containing an integer value identifying the version of RSA hardware that is on the coprocessor.</td>
</tr>
<tr>
<td>4</td>
<td>POST Version</td>
<td>A character string identifying the version of the coprocessor’s Power-On Self Test (POST) firmware. The first four characters define the POST0 version and the last four characters define the POST1 version.</td>
</tr>
<tr>
<td>5</td>
<td>Coprocessor Operating System Name</td>
<td>A character string identifying the operating system firmware on the coprocessor. Padding characters are blanks.</td>
</tr>
<tr>
<td>6</td>
<td>Coprocessor Operating System Version</td>
<td>A character string identifying the version of the operating system firmware on the coprocessor.</td>
</tr>
<tr>
<td>7</td>
<td>Coprocessor Part Number</td>
<td>A character string containing the eight-character part number identifying the version of the coprocessor.</td>
</tr>
<tr>
<td>8</td>
<td>Coprocessor EC Level</td>
<td>A character string containing the eight-character EC (engineering change) level for this version of the coprocessor.</td>
</tr>
<tr>
<td>9</td>
<td>Miniboot Version</td>
<td>A character string identifying the version of the coprocessor’s miniboot firmware. This firmware controls the loading of programs into the coprocessor. The first four characters define the MiniBoot0 version and the last four characters define the MiniBoot1 version.</td>
</tr>
<tr>
<td>10</td>
<td>CPU Speed</td>
<td>A numeric character string containing the operating speed of the microprocessor chip, in megahertz.</td>
</tr>
<tr>
<td>11</td>
<td>Adapter ID (Also see element number 15)</td>
<td>A unique identifier manufactured into the coprocessor. The coprocessor’s Adapter ID is an eight-byte binary value.</td>
</tr>
<tr>
<td>12</td>
<td>Flash Memory Size</td>
<td>A numeric character string containing the size of the flash EPROM memory on the coprocessor, in 64-kilobyte increments.</td>
</tr>
<tr>
<td>13</td>
<td>DRAM Memory Size</td>
<td>A numeric character string containing the size of the dynamic RAM (DRAM) on the coprocessor, in kilobytes.</td>
</tr>
<tr>
<td>14</td>
<td>Battery-Backed Memory Size</td>
<td>A numeric character string containing the size of the battery-backed RAM on the coprocessor, in kilobytes.</td>
</tr>
<tr>
<td>15</td>
<td>Serial Number</td>
<td>A character string containing the unique serial number of the coprocessor. The serial number is factory installed and is also reported by the CLU utility in a coprocessor signed status message.</td>
</tr>
</tbody>
</table>
### Table 188. Output for option STATDIAG

<table>
<thead>
<tr>
<th>Element Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Battery State</td>
<td>A numeric character string containing a value which indicates whether the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>battery on the coprocessor needs to be replaced:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Number</strong> <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Battery is good</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 Battery should be replaced</td>
</tr>
<tr>
<td>2</td>
<td>Intrusion Latch State</td>
<td>A numeric character string containing a value which indicates whether the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>intrusion latch on the coprocessor is set or cleared:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Number</strong> <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Latch is cleared</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 Latch is set</td>
</tr>
<tr>
<td>3</td>
<td>Error Log Status</td>
<td>A numeric character string containing a value which indicates whether there</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is data in the coprocessor CCA error log:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Number</strong> <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Error log is empty</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 Error log contains data but is not yet full</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 Error log is full</td>
</tr>
<tr>
<td>4</td>
<td>Mesh Intrusion</td>
<td>A numeric character string containing a value to indicate whether the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>coprocessor has detected tampering with the protective mesh that surrounds</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the secure module — indicating a probable attempt to physically penetrate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the module.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Number</strong> <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 No intrusion detected</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 Intrusion attempt detected</td>
</tr>
<tr>
<td>5</td>
<td>Low Voltage Detected</td>
<td>A numeric character string containing a value to indicate whether a power</td>
</tr>
<tr>
<td></td>
<td></td>
<td>supply voltage was under the minimum acceptable level. This may indicate an</td>
</tr>
<tr>
<td></td>
<td></td>
<td>attempt to attack the security module.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Number</strong> <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Only acceptable voltages have been detected</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 A voltage has been detected under the low-voltage tamper threshold</td>
</tr>
</tbody>
</table>
### Table 188. Output for option STATDIAG (continued)

<table>
<thead>
<tr>
<th>Element Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>High Voltage Detected</td>
<td>A numeric character string containing a value to indicate whether a power supply voltage was higher than the maximum acceptable level. This may indicate an attempt to attack the security module.</td>
</tr>
<tr>
<td></td>
<td>Number Meaning</td>
<td>Description</td>
</tr>
<tr>
<td>1</td>
<td>Only acceptable voltages have been detected</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>A voltage has been detected that is higher than the high-voltage tamper threshold</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Temperature Range Exceeded</td>
<td>A numeric character string containing a value to indicate whether the temperature in the secure module was outside of the acceptable limits. This may indicate an attempt to obtain information from the module:</td>
</tr>
<tr>
<td></td>
<td>Number Meaning</td>
<td>Description</td>
</tr>
<tr>
<td>1</td>
<td>Temperature is acceptable</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Detected temperature is outside an acceptable limit</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Radiation Detected</td>
<td>A numeric character string containing a value to indicate whether radiation was detected inside the secure module. This may indicate an attempt to obtain information from the module:</td>
</tr>
<tr>
<td></td>
<td>Number Meaning</td>
<td>Description</td>
</tr>
<tr>
<td>1</td>
<td>No radiation has been detected</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Radiation has been detected</td>
<td></td>
</tr>
<tr>
<td>9, 11, 13, 15, 17</td>
<td>Last Five Commands Run</td>
<td>These five rule-array elements contain the last five commands that were executed by the coprocessor CCA application. They are in chronological order, with the most recent command in element 9. Each element contains the security API command code in the first four characters and the subcommand code in the last four characters.</td>
</tr>
<tr>
<td>10, 12, 14, 16, 18</td>
<td>Last Five Return Codes</td>
<td>These five rule-array elements contain the SAPI return codes and reason codes corresponding to the five commands in rule-array elements 9, 11, 13, 15, and 17. Each element contains the return code in the first four characters and the reason code in the last four characters.</td>
</tr>
</tbody>
</table>

### Table 189. Output for option STATEID

<table>
<thead>
<tr>
<th>Element Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EID</td>
<td>During initialization, a value of zero is set in the coprocessor.</td>
</tr>
</tbody>
</table>

### Table 190. Output for option STATEXPT

<table>
<thead>
<tr>
<th>Element Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>Feature</td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1</td>
<td>Base CCA Services</td>
<td>A numeric character string containing a value to indicate whether base CCA services are available.</td>
</tr>
<tr>
<td></td>
<td>Availability</td>
<td>Number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>CDMF Availability</td>
<td>A numeric character string containing a value to indicate whether CDMF is available.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>56-bit DES Availability</td>
<td>A numeric character string containing a value to indicate whether 56-bit DES encryption is available.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Triple-DES Availability</td>
<td>A numeric character string containing a value to indicate whether triple-DES encryption is available.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>SET Services Availability</td>
<td>A numeric character string containing a value to indicate whether SET (Secure Electronic Transaction) services are available.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Table 190. Output for option STATEXPT (continued)

<table>
<thead>
<tr>
<th>Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DSA not available</td>
</tr>
<tr>
<td>1024</td>
<td>DSA 1024 key size</td>
</tr>
<tr>
<td>2048</td>
<td>DSA 2048 key size</td>
</tr>
<tr>
<td>4096</td>
<td>RSA 4096 key size</td>
</tr>
</tbody>
</table>

reserved_data_length

Direction: Input
Type: Integer

The length of the reserved_data parameter. Currently, the value must be 0.

reserved_data

Direction: Input
Type: String

This field is currently not used.

Usage Notes

RACF will be invoked to check authorization to use this service.

PKA key generate available indicates the PKA callable services are enabled and there is at least one ACTIVE coprocessor.

The options ICSFSTAT and ICSFST2 report on the state of PKA callable services. ICSFSTAT reports it in element 2. ICSFST2 reports it in element 3. There is a subtle difference between the two options. ICSFSTAT reports PKA callable services as enabled only after the DES master key is loaded and valid. ICSFSTAT does not report PKA callable services as enabled when only the AES master key is loaded and valid. Option ICSFST2 reports PKA callable services as enabled when the DES and/or AES master key is loaded and valid.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 191. ICSF Query Service required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 191. ICSF Query Service required hardware (continued)

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 990</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 880</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

**X9.9 Data Editing (CSNB9ED)**

Use this utility to edit an ASCII text string according to the editing rules of ANSI X9.9-4. It edits the text that the `source_text` parameter supplies according to these rules. The rules are listed here in the order in which they are applied. It returns the result in the `target_text` parameter.

1. This service replaces each carriage-return (CR) character and each line-feed (LF) character with a single-space character.
2. It replaces each lowercase alphabetic character (a through z) with its equivalent uppercase character (A through Z).
3. It deletes all characters other than:
   - Alphabets A...Z
   - Numerics 0..9
   - Space
   - Comma ,
   - Period .
   - Dash -
   - Solidus /
   - Asterisk *
   - Open parenthesis (   
   - Close parenthesis )
4. It deletes all leading space characters.
5. It replaces all sequences of two or more space characters with a single-space character.

**Format**

```call
CALL CSNB9ED(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    text_length,
    source_text,
    target_text)
```
X9.9 Data Editing (CSNB9ED)

Parameters

**return_code**
- Direction: Output
- Type: Integer

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

**reason_code**
- Direction: Output
- Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit_data_length**
- Direction: Ignored
- Type: Integer

Reserved field.

**exit_data**
- Direction: Ignored
- Type: String

Reserved field.

**text_length**
- Direction: Input/Output
- Type: Integer

On input, the text_length contains an integer that is the length of the source_text. The length must be a positive, nonzero value. On output, text_length is updated with an integer that is the length of the edited text.

**source_text**
- Direction: Input
- Type: String

This parameter contains the string to edit.

**target_text**
- Direction: Output
- Type: String

The edited text that the callable service returns.

Usage Notes

This service is structured differently from the other services. It runs in the caller's address space in the caller's key and mode.

ICSF need not be active for the service to run. There are no pre-processing or post-processing exits that are enabled for this service. While running, this service does not issue any calls to RACF.
This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 192. X9.9 data editing required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>None.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>None.</td>
<td></td>
</tr>
</tbody>
</table>
The Trusted Key Entry (TKE) workstation is an optional feature. It offers an alternative to clear key entry. You can use the TKE workstation to load:

- DES master keys, PKA master keys, and operational keys in a secure way. CCF only supports Operational Transport and PIN keys. On the PCIXCC and CEX2C, all operational keys may be loaded with TKE V4.1 or higher. On the CEX3C, all operational keys may be loaded with TKE 6.0 or higher.
- DES-MK and ASYM-MK master keys on the PCICC, PCIXCC, CEX2C, and CEX3C.
- AES-MK master key and operational key are supported on the z9 and z10 systems with the Nov. 2008 or later licensed internal code (LIC).

This topic describes these callable services:
- "PCI Interface Callable Service (CSFPCI)"
- "PKSC Interface Callable Service (CSFPKSC)" on page 467

**PCI Interface Callable Service (CSFPCI)**

TKE uses this callable service to send a request to a specific PCI card queue and remove the corresponding response when complete. This service also allows the TKE workstation to query the list of access control points which may be enabled or disabled by a TKE user. This service is synchronous. The return and reason codes reflect the success or failure of the queue functions rather than the success or failure of the actual PCI request.

**Format**

```c
CALL CSFPCI(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    target_pci_coprocessor,
    target_pci_coprocessor_serial_number,
    request_block_length,
    request_block,
    request_data_block_length,
    request_data_block,
    reply_block_length,
    reply_block,
    reply_data_block_length,
    reply_data_block,
    masks_length,
    masks_data)
```

**Parameters**

- **return_code**
  
  Direction: Output
  Type: Integer
The return code specifies the general result of the callable service. See Appendix A, “ICSF and TSS Return and Reason Codes,” for a list of return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. See Appendix A, “ICSF and TSS Return and Reason Codes” for a list of reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the **exit_data** parameter.

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**rule_array_count**

Direction: Input  
Type: Integer

The number of keywords you are supplying in **rule_array**. The value must be 1.

**rule_array**

Direction: Input  
Type: String

Keyword that provides control information to callable services. The keyword is left-justified in an 8-byte field and padded on the right with blanks. The keyword must be in contiguous storage. These keywords are mutually exclusive:

**Table 193. Keywords for PCI Interface Callable Service**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACPOINTS</td>
<td>Queries the list of access control points which may be enabled or disabled by a TKE user.</td>
</tr>
<tr>
<td>ACTIVECP</td>
<td>This keyword is a request to call the PCI card initialization code to revalidate the PCI cards. When the PCI card initialization is completed, both the 64-bit mask indicating which of the PCI cards are online and 64-bit mask indicating which of the PCI cards are active will be returned. This keyword is used by the TKE workstation code after the ACTIVATE portion of the domain zeroize command. This is to ensure that the status of the PCI card is accurately reflected to the users. See the masks_data parameter description for more information.</td>
</tr>
<tr>
<td>APNUM</td>
<td>Specifies the target_pci_coprocessor field to be used.</td>
</tr>
<tr>
<td>SERIALNO</td>
<td>Specifies the target_pci_coprocessor_number field to be used.</td>
</tr>
</tbody>
</table>
Table 193. Keywords for PCI Interface Callable Service (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCIMASKS</td>
<td>This keyword is a request to return both the 64-bit mask indicating which of the PCI cards are online and 64-bit mask indicating which of the PCI cards are active. See the masks_data parameter description for more information.</td>
</tr>
<tr>
<td>XCPMASK</td>
<td>This keyword is a request to return both the 64-bit mask indicating which of the PCIXCCs, CEX2Cs, and CEX3Cs are online and the 64-bit mask indicating which of the PCIXCCs, CEX2Cs, and CEX3Cs are active. See the masks_data parameter description for more information.</td>
</tr>
<tr>
<td>CX2MASK</td>
<td>This keyword is a request to return both the 64-bit mask indicating which of the CEX2Cs are online and the 64-bit mask indicating which of the CEX2Cs are active. See the masks_data parameter description for more information.</td>
</tr>
<tr>
<td>CX3MASK</td>
<td>This keyword is a request to return both the 64-bit mask indicating which of the CEX3Cs are online and the 64-bit mask indicating which of the CEX3Cs are active. See the masks_data parameter description for more information.</td>
</tr>
</tbody>
</table>

**Note:** When the PCIMASKS, ACTIVECP, XCPMASK, CX2MASK and CX3MASK keywords are specified, the request_data_block_length, request_data_block, reply_data_block_length, and the reply_data_block parameters are ignored.

**target_pci_coprocessor**
- **Direction:** Input
- **Type:** Integer

The PCICC, PCIXCC, CEX2C, or CEX3C card to which this request is directed. Valid values are between 0 and 64.

**target_pci_coprocessor_serial_number**
- **Direction:** Input/Output
- **Type:** String

The PCICC, PCIXCC, CEX2C, or CEX3C card serial number to which the request is directed. This parameter may be used instead of the target_pci_coprocessor. The length is 8 bytes. This parameter is updated with the serial number of the card if the request was successfully processed.

**request_block_length**
- **Direction:** Input
- **Type:** Integer

Length of CPRB and the request block in the request_block field. The maximum length allowed is 5,500 bytes.

**request_block**
- **Direction:** Input
- **Type:** String

PCICC, PCIXCC, CEX2C, or CEX3C command or query request for the target PCICC, PCIXCC, CEX2C, or CEX3C. This is the complete CPRB and request block to be processed by the PCICC, PCIXCC, CEX2C, or CEX3C.
PCI Interface (CSFPCI)

request_data_block_length
Direction: Input  Type: Integer

Length of request data block in the request_data_block field. The maximum length allowed is 6,400 bytes. The length field must be a multiple of 4.

request_data_block
Direction: Input  Type: String

The data that accompanies the request_block field.

reply_block_length
Direction: Input/Output  Type: Integer

Length of CPRB and the reply block in the reply_block field. The maximum length allowed is 5,500 bytes. This field is updated on output with the actual length of the reply_block field.

reply_block
Direction: Output  Type: String

Reply from the target PCICC, PCIXCC, CEX2C, or CEX3C. This is the CPRB and reply block that has been processed by the PCICC, PCIXCC, CEX2C, or CEX3C.

reply_data_block_length
Direction: Input/Output  Type: Integer

Length of reply block in the reply_data_block field. The maximum length allowed is 6,400 bytes. This field is updated on output with the actual length of the reply_data_block field. This length field must be a multiple of 4. For the ACPOINTS keyword, the minimum length is 2572 bytes.

reply_data_block
Direction: Output  Type: String

The data that accompanies the reply_block field.

masks_length
Direction: Input  Type: Integer

Length of the reply data being returned in the masks_data field. The length must be 32 bytes. This field is only valid when the input rule_array keyword is PCIMASKS, ACTIVECP XCPMASK, CX2MASK, CX3MASK. For all other rule_array keywords, this field is ignored.

masks_data
Direction: Output  Type: String

The data being returned for all requests. The first 8 bytes indicate the count of the PCI cards online. The second 8 bytes indicate a bit mask of the actual PCI
cards brought online. The third 8 bytes indicate the count of the PCI cards active. The fourth 8 bytes indicate a bit mask of the actual PCI cards that are active. For the ACTVECP keyword, if the PCI card initialization failed, the appropriate return code and reason code is issued and the masks_data field will contain zeros.

Usage Notes

The target_pciCoprocessor, the target_pciCoprocessor_serial_number, the request_block, the reply_block, the request_block_data_block, and the reply_block_data_block, are recorded in SMF Record Type 82, subtype 16.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

**Table 194. PCI Interface required hardware**

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>PCI Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>PCI X Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>PCI X Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>Crypto Express2 Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td>Crypto Express3 Coprocessor</td>
<td></td>
</tr>
</tbody>
</table>

PKSC Interface Callable Service (CSFPKSC)

**Restriction:** This service is only supported on the IBM @server zSeries 800 and IBM @server zSeries 900.

TKE uses this callable service to send a request to a specific cryptographic module and receive a corresponding response when processing is complete. The service is synchronous. Note that the return and reason codes reflect the success or failure of CSFPKSC's interaction with the cryptographic module rather than the success or failure of the cryptographic module request. The response block contains the results of the cryptographic module request.
PKSC Interface (CSFPKSC)

Format

```c
CALL CSFPKSC(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    target_crypto_module,
    request_length,
    request,
    response)
```

Parameters

**return_code**
- Direction: Output
- Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**
- Direction: Output
- Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**
- Direction: Input/Output
- Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**
- Direction: Input/Output
- Type: String

The data that is passed to the installation exit.

**target_crypto_module**
- Direction: Input
- Type: Integer

Cryptographic module to which this request is directed. Value is 0 or 1.

**request_length**
- Direction: Input
- Type: Integer

Length of request message in the request field. The maximum length allowed is 1024 bytes.
PKSC Interface (CSFPKSC)

**request**
Direction: Input  Type: String

PKSC command or query request for the target cryptographic module. This is the complete architected command or query for the cryptographic module to process.

**response**
Direction: Output  Type: String

Area where the PKSC response from the target cryptographic module is returned to the caller. The area returned can be up to 512 bytes.

**Usage Notes**

The format and content of the PKSC request and response areas are proprietary IBM hardware information that may be licensed. Customers interested in this information may contact the IBM Director of Licensing. For the address, refer to Notices.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

*Table 195. PKSC Interface required hardware*

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>PCI Cryptographic Coprocessor</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td></td>
<td>This service is not supported.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td>This service is not supported.</td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PKSC Interface (CSFPKSC)
Chapter 13. Managing Keys According to the ANSI X9.17 Standard

This topic describes the callable services that support the ANSI X9.17 key management standard:

- **“ANSI X9.17 EDC Generate (CSNAEGN)”**
- **“ANSI X9.17 Key Export (CSNAKEX)” on page 473**
- **“ANSI X9.17 Key Import (CSNAKIM)” on page 478**
- **“ANSI X9.17 Key Translate (CSNAKTR)” on page 483**
- **“ANSI X9.17 Transport Key Partial Notarize (CSNATKN)” on page 488**

These services are only supported on an IBM @server zSeries 800 and IBM @server zSeries 900.

These callable services, that are described in other topics of this publication, also support the ANSI X9.17 key management standard:

- **“Key Generate (CSNBKGN and CSNEKGN)” on page 111**
- **“Key Part Import (CSNBKPI)” on page 127**
- **“Key Token Build (CSNBKTB)” on page 147**

### ANSI X9.17 EDC Generate (CSNAEGN)

Use the ANSI X9.17 EDC generate callable service to generate an error detection code (EDC) on a text string. The service calculates the EDC by by using a key value of X'0123456789ABCDEF' to generate a MAC on the specified text string, as defined by the ANSI X9.17 standard.

**Restriction:** This service is only supported on an IBM @server zSeries 800 and IBM @server zSeries 900.

**Format**

```call
CALL CSNAEGN( 
    return_code, 
    reason_code, 
    exit_data_length, 
    exit_data, 
    rule_array_count, 
    rule_array, 
    text_length, 
    text, 
    chaining_vector, 
    EDC)
```

**Parameters**

**return_code**

Direction: Output  
Type: Integer

The **return code** specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.
ANSI X9.17 EDC Generate (CSNAEGN)

reason_code
Direction: Output                      Type: Integer

The reason code specifies the result of the callable service that is returned to
the application program. Each return code has different reason codes that are
assigned to it that indicate specific processing problems. Appendix A, “ICSF and
TSS Return and Reason Codes” lists the reason codes.

exit_data_length
Direction: Input/Output                Type: Integer

The length of the data that is passed to the installation exit. The length can be
from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the
exit_data parameter.

exit_data
Direction: Input/Output                Type: String

The data that is passed to the installation exit.

rule_array_count
Direction: Input                       Type: Integer

The number of keywords you supplied in the rule_array parameter. The value
must be 0.

rule_array
Direction: Input                       Type: String

Keywords that provide control information to the callable service. Currently there
are no keywords that are defined for this variable, but you must declare the
variable. To do so, declare an area of blanks of any length.

text_length
Direction: Input                       Type: Integer

The length of the user-supplied text parameter for which the service should
calculate the EDC.

text
Direction: Input                       Type: String

The application-supplied text field for which the service is to generate the EDC.

chaining_vector
Direction: Input/Output                Type: String

An 18-byte string that ICSF uses as a system work area. The chaining vector
permits data to be chained from one call to another. ICSF ignores the
information in this field, but you must declare an 18-byte string.
ANSI X9.17 EDC Generate (CSNAEGN)

EDC
Direction: Output Type: String

A 9-byte field where the callable service returns the EDC generated as two groups of four ASCII-encoded hexadecimal characters that are separated by an ASCII space character.

Usage Notes

The ANSI X9.17 standard states that for EDC, prior to the service generating the MAC the caller must first edit the input text according to topic 4.3 of ANSI X9.9-1982. It is the caller’s responsibility to do the editing prior to calling the ANSI X9.17 EDC generate service. If the supplied text is not a multiple of 8, the service pads the text with X’00’ up to a multiple of 8, as specified in ANSI X9.9-1.

To use this service you must have the ANSI system keys installed in the CKDS.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 196. ANSI X9.17 EDC generate required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ANSI X9.17 Key Export (CSNAKEX)

Use the ANSI X9.17 key export callable service to export a DATA key or a pair of DATA keys, along with an ANSI key-encrypting key (AKEK), using the ANSI X9.17 protocol. This service converts a single DATA key, or combines two DATA keys, into a single MAC key. You can use the MAC key in either, or both, the MAC generation, or MAC verification service to authenticate the service message. In addition, this service also supports the export of a CCA IMPORTER or EXPORTER KEK.

If you export only DATA keys, the DATA keys are exported encrypted under the specified transport AKEK. You have the option of applying the ANSI X9.17 key offset or key notarization process to the transport AKEK.
ANSI X9.17 Key Export (CSNAKEX)

If you export both DATA keys and an AKEK, the DATA keys are exported encrypted under the key-encrypting key that is also being exported. The AKEK is exported encrypted under the specified transport AKEK. You have the option of applying the ANSI X9.17 key offset or key notarization process to the transport AKEK. The ANSI X9.17 key offset process is applied to the source AKEK. Use the CKT keyword to specify whether to use an offset of 0 or 1. Use an offset of 0 when sending the DATA key to a key translation center along with a transport AKEK.

**Note:** You must create the cryptographic service message and maintain the offset counter value that is associated with the AKEK.

**Restriction:** This service is only supported on an IBM @server zSeries 800 and IBM @server zSeries 900.

### Format

```call csnakex(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    origin_identifier,
    destination_identifier,
    source_data_key_1_identifier,
    source_data_key_2_identifier,
    source_key_encrypting_key_identifier,
    transport_key_identifier,
    outbound_KEK_count,
    target_data_key_1,
    target_data_key_2,
    target_key_encrypting_key,
    MAC_key_token)
```

### Parameters

**return_code**

Direction: Output Type: Integer

The return code specifies the general result of the callable service. [Appendix A, "ICSF and TSS Return and Reason Codes"] lists the return codes.

**reason_code**

Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicates specific processing problems. [Appendix A, "ICSF and TSS Return and Reason Codes"] lists the reason codes.

**exit_data_length**

Direction: Input/Output Type: Integer

...
ANSI X9.17 Key Export (CSNAKEX)

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**rule_array_count**

Direction: Input  
Type: Integer

The number of keywords you supplied in the rule_array parameter. The value can be 0 to 4. If you specify 0, the callable service does not perform either notarization or offset.

**rule_array**

Direction: Input  
Type: String

Zero to four keywords that provide control information to the callable service. See the list of keywords in Table 197. The keywords must be in 8 to 32 bytes of contiguous storage. Left-justify each keyword in its own 8-byte location and pad on the right with blanks. You must specify this parameter even if you specify no keyword.

**Table 197. Keywords for ANSI X9.17 Key Export Rule Array**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Notarization and Offset Rule (optional with no defaults)</strong></td>
<td></td>
</tr>
<tr>
<td>CPLT-NOT</td>
<td>Complete ANSI X9.17 notarization using the value obtained from the outbound KEK_count parameter. The transport key that the transport_key_identifier specifies must be partially notarized.</td>
</tr>
<tr>
<td>NOTARIZE</td>
<td>Perform notarization processing using the values obtained from the origin_identifier, destination_identifier, and outbound KEK_count parameters.</td>
</tr>
<tr>
<td>OFFSET</td>
<td>Perform ANSI X9.17 key offset processing using the origin counter value obtained from the outbound KEK_count parameter.</td>
</tr>
<tr>
<td><strong>Parity Rule (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>ENFORCE</td>
<td>Stop processing if any source keys do not have odd parity. This is the default value.</td>
</tr>
<tr>
<td>IGNORE</td>
<td>Ignore the parity of the source key.</td>
</tr>
<tr>
<td><strong>Source Key Rule (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>CCA-EXP</td>
<td>Export a CCA EXPORTER KEK. Requires NOCV keys to be enabled.</td>
</tr>
<tr>
<td>CCA-IMP</td>
<td>Export a CCA IMPORTER KEK. Requires NOCV keys to be enabled.</td>
</tr>
<tr>
<td>1-KD</td>
<td>Export one DATA key. This is the default parameter.</td>
</tr>
<tr>
<td>1-KD+KK</td>
<td>Export one DATA key and a single-length AKEK.</td>
</tr>
<tr>
<td>1-KD+*KK</td>
<td>Export one DATA key and a double-length AKEK.</td>
</tr>
</tbody>
</table>
ANSI X9.17 Key Export (CSNAKEX)

Table 197. Keywords for ANSI X9.17 Key Export Rule Array (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-KD</td>
<td>Export two DATA keys.</td>
</tr>
<tr>
<td>2-KD+KK</td>
<td>Export two DATA keys and a single-length AKEK.</td>
</tr>
<tr>
<td>2-KD+KK</td>
<td>Export two DATA keys and a double-length AKEK.</td>
</tr>
</tbody>
</table>

Data Key Offset Value (optional)

| CKT          | Valid only when a key-encrypting key is being exported along with a DATA key. If this keyword is specified, any DATA keys being exported are encrypted under the key-encrypting key using an offset value of 0. If this keyword is not specified (this is the default), any DATA keys being exported are encrypted under the key-encrypting key using an offset value of 1. The CKT keyword is not valid with CCA-IMP or CCA-EXP keywords. |

origin_identifier

Direction: Input  Type: String

This parameter is valid if the NOTARIZE keyword is specified. It specifies an area that contains a 16-byte string that contains the origin identifier that is defined in the ANSI X9.17 standard. The string must be ASCII characters, left-justified, and padded on the right by space characters. This parameter must be a minimum of four, non-space characters. ICSF ignores this parameter if you specify the OFFSET or CPLT-NOT keyword in the rule_array parameter.

destination_identifier

Direction: Input  Type: String

This parameter is valid if the NOTARIZE keyword is specified. It specifies an area that contains a 16-byte string. The 16-byte string contains the destination identifier that is defined in the ANSI X9.17 standard. The string must be ASCII characters, left-justified, and padded on the right by space characters. This parameter must be a minimum of four, non-space characters. ICSF ignores this parameter if you specify the OFFSET or CPLT-NOT keyword in the rule_array parameter.

source_data_key_1_identifier

Direction: Input/Output  Type: String

A 64-byte area that contains an internal token, or the label of a CKDS entry that contains a DATA key. ICSF ignores this field if you specify CCA-EXP or CCA-IMP in the rule_array parameter.

source_data_key_2_identifier

Direction: Input/Output  Type: String

A 64-byte area that contains an internal token, or the label of a CKDS entry that contains a DATA key. This parameter is valid only if you specify 2-KD, 2-KD+KK, or 2-KD+KK as the source key rule keyword on the rule_array parameter. ICSF ignores this parameter if you specify other source key rule keywords, or if you specify CCA-EXP or CCA-IMP in the rule_array parameter.
source_key_encrypting_key_identifier
Direction: Input/Output  Type: String

A 64-byte area that contains an internal token, or the label of a CKDS entry that contains either an AKEK, a CCA IMPOR Ter, or a CCA EXPORTER key. If this parameter contains an AKEK, you must specify 1-KD+KK, 2-KD+KK, 1-KD+KK, or 2-KD+*KK for the source key rule on the rule_array parameter. If this parameter contains a CCA IMPORTER or CCA EXPORTER key, you must specify CCA-IMP or CCA-EXP, respectively, for the source key rule on the rule_array parameter. ICSF ignores this field if you specify any other source key rule keywords.

transport_key_identifier
Direction: Input/Output  Type: String

A 64-byte area that contains either an internal token or a label that refers to an internal token for an AKEK.

outbound KEK_count
Direction: Input  Type: String

An 8-byte area that contains an ASCII count that is used in the notarization process. The count is an ASCII character string, left-justified, and padded on the right by ASCII space characters. ICSF interprets a single ASCII space character as a zero counter. The maximum value is 99999999.

target_data_key_1
Direction: Output  Type: String

A 16-byte area where the exported data key 1 is returned. The enciphered key is an ASCII-encoded hexadecimal string.

target_data_key_2
Direction: Output  Type: String

A 16-byte area where the exported data key 2 is returned. The enciphered key is an ASCII-encoded hexadecimal string. This key is returned if 2-KD, 2-KD+KK, or 2-KD+*KK is specified in the rule_array parameter.

target_key_encrypting_key
Direction: Output  Type: String

If the rule_array parameter specifies 1-KD+KK, 2-KD+KK, 1-KD+KK, or 2-KD+*KK, this parameter specifies a 32-byte area that contains the exported AKEK. If the rule_array parameter specifies CCA-IMP or CCA-EXP, this parameter specifies a 32-byte area that contains the exported key-encrypting key (KEK). The enciphered key is an ASCII-encoded hexadecimal string. If the rule_array parameter specifies 1-KD+KK or 2-KD+KK, the 16-byte ASCII-encoded output is left-justified in the field and the rest of the field remains unchanged.
ANSI X9.17 Key Export (CSNAKEX)

**MAC_key_token**

*Direction: Output  Type: String*

A 64-byte area that contains an internal token for a MAC key that is intended for use in the MAC generation or MAC verification process. This field is the EXCLUSIVE OR of the two supplied DATA keys when the source key rule in the rule_array parameter specifies 2-KD, 2-KD+KK, or 2-KD+*KK. When the source key rule specifies 1-KD, the DATA key is converted to a MAC key and returned as an internal token in this field.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

You must install the ANSI system keys in the CKDS to use this service.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

**Table 198. ANSI X9.17 key export required hardware**

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ANSI X9.17 Key Import (CSNAKIM)

Use the ANSI X9.17 key import callable service to import a DATA key or a pair of DATA keys, along with an ANSI key-encrypting key (AKEK), using the ANSI X9.17 protocol. This service converts a single DATA key, or combines two DATA keys, into a single MAC key. The MAC key can be used in either, or both, the MAC generation or the MAC verification service to authenticate the service message. In addition, this service also supports the import of the KEK to a CCA IMPORTER or EXPORTER KEK, as well as an AKEK.
If you are importing only DATA keys, this service assumes that the DATA keys are encrypted under the specified transport AKEK. You have the option of applying the ANSI X9.17 key offset or key notarization process to the transport AKEK.

If you are importing both DATA keys and an AKEK, this service assumes that the AKEK is encrypted under the specified transport AKEK. This service also assumes that the DATA keys are encrypted under the source AKEK that is also being imported. You have the option of applying the ANSI X9.17 key offset or key notarization process to the transport AKEK. ICSF applies the ANSI X9.17 key offset process to the source AKEK with an offset of 1.

**Note:** You must create the cryptographic service message and maintain the offset counter value that is associated with the AKEK.

**Restriction:** This service is only supported on an IBM @server zSeries 800 and IBM @server zSeries 900.

### Format

```python
CALL CSNAKIM( 
   return_code, 
   reason_code, 
   exit_data_length, 
   exit_data, 
   rule_array_count, 
   rule_array, 
   origin_identifier, 
   destination_identifier, 
   source_data_key_1, 
   source_data_key_2, 
   source_key_encrypting_key, 
   inbound_KEK_count, 
   transport_key_identifier, 
   target_data_key_1, 
   target_data_key_2, 
   target_key_encrypting_key, 
   MAC_key_token)
```

### Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. [Appendix A, "ICSF and TSS Return and Reason Codes"] lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. [Appendix A, "ICSF and TSS Return and Reason Codes"] lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer
ANSI X9.17 Key Import (CSNAKIM)

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**rule_array_count**

Direction: Input  
Type: Integer

The number of keywords you supplied in the rule_array parameter. The value can be 0 to 3. If you specify 0, ICSF does not perform either notarization or offset.

**rule_array**

Direction: Input  
Type: String

Zero to three keywords that provide control information to the callable service. See the list of keywords in Table 199. The keywords must be in 8 to 24 bytes of contiguous storage. Each of the keywords must be left-justified in its own 8-byte location and padded on the right with blanks. You must specify this parameter even if you do not specify a keyword.

**Table 199. Keywords for ANSI X9.17 Key Import Rule Array**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Notarization and Offset Rule (optional with no defaults)</strong></td>
<td></td>
</tr>
<tr>
<td>CPLT-NOT</td>
<td>Complete ANSI X9.17 notarization using the value obtained from the inbound KEK_count parameter. The transport key that the transport_key_identifier specifies must be partially notarized.</td>
</tr>
<tr>
<td>NOTARIZE</td>
<td>Perform notarization processing using the values obtained from the origin_identifier, destination_identifier, and inbound KEK_count parameters.</td>
</tr>
<tr>
<td>OFFSET</td>
<td>Perform ANSI X9.17 key offset processing using the origin counter value obtained from the inbound KEK_count parameter.</td>
</tr>
<tr>
<td><strong>Parity Rule (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>ENFORCE</td>
<td>Stop processing if any source keys do not have odd parity. This is the default value.</td>
</tr>
<tr>
<td>IGNORE</td>
<td>Ignore the parity of the source key.</td>
</tr>
<tr>
<td><strong>Source Key Rule (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>CCA-EXP</td>
<td>Import a key-encrypting key as a CCA EXPORTER. Requires NOCV keys to be enabled.</td>
</tr>
<tr>
<td>CCA-IMP</td>
<td>Import a key-encrypting key as a CCA IMPORTER. Requires NOCV keys to be enabled.</td>
</tr>
<tr>
<td>1-KD</td>
<td>Import one DATA key. This is the default parameter.</td>
</tr>
<tr>
<td>1-KD+KK</td>
<td>Import one DATA key and a single-length AKEK.</td>
</tr>
</tbody>
</table>
ANSI X9.17 Key Import (CSNAKIM)

Table 199. Keywords for ANSI X9.17 Key Import Rule Array (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-KD+*KK</td>
<td>Import one DATA key and a double-length AKEK.</td>
</tr>
<tr>
<td>2-KD</td>
<td>Import two DATA keys.</td>
</tr>
<tr>
<td>2-KD+KK</td>
<td>Import two DATA keys and a single-length AKEK.</td>
</tr>
<tr>
<td>2-KD+*KK</td>
<td>Import two DATA keys and a double-length AKEK.</td>
</tr>
</tbody>
</table>

**origin_identifier**

Direction: Input  
Type: String

This parameter is valid if you specify the NOTARIZE keyword in the `rule_array` parameter. It specifies an area that contains a 16-byte string that contains the origin identifier that is defined in the ANSI X9.17 standard. The string must be ASCII characters, left-justified, and padded on the right by space characters. The string must be a minimum of four, non-space characters. This parameter is ignored if the OFFSET or CPLT-NOT keyword is specified.

**destination_identifier**

Direction: Input  
Type: String

This parameter is valid if you specify the NOTARIZE keyword in the `rule_array` parameter. It specifies an area that contains a 16-byte string that contains the destination identifier that is defined in the ANSI X9.17 standard. The string must be ASCII characters, left-justified, and padded on the right by space characters. It must be a minimum of four non-space characters. This parameter is ignored if the OFFSET or CPLT-NOT keyword is specified.

**source_data_key_1**

Direction: Input  
Type: String

A 16-byte area that contains the enciphered DATA key to be imported. You must supply the DATA key as an ASCII-encoded hexadecimal string. The field is ignored if the `rule_array` parameter specifies CCA-IMP or CCA-EXP.

**source_data_key_2**

Direction: Input  
Type: String

A 16-byte area that contains the second enciphered DATA key to be imported. This parameter is valid only if the `rule_array` parameter specifies KK, or 2-KD+KK. You must supply the key as an ASCII-encoded hexadecimal string. This field is ignored if the `rule_array` parameter specifies other source key rules.

**source_key_encrypting_key**

Direction: Input  
Type: String

A 16- or 32-byte area that contains an enciphered AKEK, if the `rule_array` parameter specifies either 1-KD+KK, 2-KD+KK, 1-KD+KK, or 2-KD+KK. This parameter specifies a KEK, if the `rule_array` parameter specifies either CCA-IMP or CCA-EXP. The area is 16 bytes if the `rule_array` parameter specifies a single-length AKEK (1-KD+KK or 2-KD+KK). The area is 32 bytes if
the `rule_array` parameter specifies a double-length AKEK (1-KD+"KK or 2-KD+"KK). You must supply the key as an ASCII-encoded hexadecimal string. This field is ignored if the `rule_array` parameter specifies 1-KD or 2-KD.

**inbound_KEK_count**

Direction: Input
Type: String

An 8-byte area that contains an ASCII count for use in the notarization process. The count is an ASCII character string, left-justified, and padded on the right by space characters. ICSF interprets a single space character as a zero counter. The maximum value is 99999999.

**transport_key_identifier**

Direction: Input/Output
Type: String

A 64-byte area that contains an internal token or a label that refers to an internal token for an AKEK.

**target_data_key_1**

Direction: Output
Type: String

A 64-byte area where the imported data key 1 is returned as an ICSF internal key token. ICSF does not support the direct import by label.

**target_data_key_2**

Direction: Output
Type: String

A 64-byte area where the imported data key 2 is returned as an ICSF internal key token. ICSF does not support the direct import by label. This key is returned if 2-KD, 2-KD+KK, or 2-KD+*KK is specified in the `rule_array` parameter.

**target_key_encrypting_key**

Direction: Output
Type: String

A 64-byte area where the imported key-encrypting key is returned as an ICSF internal key token. If the `rule_array` parameter specifies 1-KD+KK, 1-KD+*KK, 2-KD+KK, or 2-KD+*KK, the internal key token contains an AKEK. If the `rule_array` parameter specifies either CCA-IMP or CCA-EXP, the internal token contains a CCA IMPORTER or a CCA EXPORTER, respectively.

**MAC_key_token**

Direction: Output
Type: String

A 64-byte area that contains an internal token for a MAC key that is intended for use in the MAC generation or MAC verification function. This field is the EXCLUSIVE OR of the two imported DATA keys if the source key rule in the `rule_array` parameter specifies 2-KD, 2-KD+KK, or 2-KD+*KK. If the source key rule in the `rule_array` parameter specifies 1-KD, ICSF converts the DATA key to a MAC key and returns it as an internal token in this field.
ANSI X9.17 Key Import (CSNAKIM)

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

You must install the ANSI system keys in the CKDS to use this service.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 200. ANSI X9.17 key import required hardware

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ANSI X9.17 Key Translate (CSNAKTR)

Use the ANSI X9.17 key translate callable service to translate a key from encryption under one AKEK to encryption under another AKEK. In a single service call you can translate either one or two encrypted DATA keys, or a single encrypted key-encrypting key. In addition, this service also imports the supplied DATA keys. If the rule_array parameter specifies 2-KD, this service exclusive-ORs the two imported DATA keys and converts the result into a MAC key, which it returns in the MAC_key_token field. The MAC key is used to perform MAC processing on the service message. If the rule_array specifies keywords 1-KD and 2-KD, ICSF translates only DATA keys. The service uses the inbound transport key-encrypting key to decrypt the DATA keys, and uses the outbound transport key-encrypting key to reencrypt the DATA keys. The service uses the ANSI X9.17 key offset process during decryption or importing. The service can use the ANSI X9.17 notarization process during reencryption or exporting of the DATA keys.

If the rule_array parameter specifies 1-KD+KK or 1-KD+*KK, the service translates only the AKEK. The service uses the inbound transport key-encrypting key to decrypt or import the input AKEK, applying the ANSI X9.17 offset process. The service uses the outbound transport key-encrypting key to reencipher or export the AKEK, with or without applying the optional ANSI X9.17 notarization process. ICSF uses the inbound key-encrypting key that is being translated to import the supplied
ANSI X9.17 Key Translate (CSNAKTR)

DATA key, applying the ANSI X9.17 offset processing only with an offset of 0. The
DATA key is imported as previously discussed then converted to a MAC key token
and returned in the MAC_key_token field.

Restriction: This service is only supported on an IBM zSeries 800 and
IBM zSeries 900.

Format

CALL CSNAKTR(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    inbound KEK_count,
    inbound_transport_key_identifier,
    inbound_data_key_1,
    inbound_data_key_2,
    inbound_key_encrypting_key,
    outbound_origin_identifier,
    outbound_destination_identifier,
    outbound KEK_count,
    outbound_transport_key_identifier,
    outbound_data_key_1,
    outbound_data_key_2,
    outbound_key_encrypting_key,
    MAC_key_token)

Parameters

return_code
Direction: Output Type: Integer

The return code specifies the general result of the callable service. Appendix A,
"ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code
Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to
the application program. Each return code has different reason codes that are
assigned to it that indicate specific processing problems. Appendix A, "ICSF and
TSS Return and Reason Codes" lists the reason codes.

exit_data_length
Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be
from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the
exit_data parameter.

exit_data
Direction: Input/Output Type: String
The data that is passed to the installation exit.

**rule_array_count**

Direction: Input  
Type: Integer

The number of keywords you supplied in the *rule_array* parameter. The value can be 0 to 3. If you specify 0, the service does not perform notarization or offset.

**rule_array**

Direction: Input  
Type: String

Zero to three keywords that provide control information to the callable service. See the list of keywords in Table 201. The keywords must be in 8 to 24 bytes of contiguous storage. Each of the keywords must be left-justified in its own 8-byte location and padded on the right with blanks. You must specify this parameter even if do not specify any keywords.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Notarization Rule (optional with no defaults)</strong></td>
<td></td>
</tr>
<tr>
<td>CPLT-NOT</td>
<td>Complete ANSI X9.17 notarization using the value obtained from the <em>outbound_KEK_count</em> parameter. The outbound transport key specified must be partially notarized.</td>
</tr>
<tr>
<td>NOTARIZE</td>
<td>Perform notarization processing using the values obtained from the <em>outbound_origin_identifier</em>, the <em>outbound_destination_identifier</em>, and the <em>outbound_KEK_count</em>.</td>
</tr>
<tr>
<td><strong>Parity Rule (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>ENFORCE</td>
<td>Stop processing if any source keys do not have odd parity. This is the default value.</td>
</tr>
<tr>
<td>IGNORE</td>
<td>Ignore the parity of the source key.</td>
</tr>
<tr>
<td><strong>Source Key Rule (optional)</strong></td>
<td></td>
</tr>
<tr>
<td>1-KD</td>
<td>Import and translate one DATA key. This is the default parameter.</td>
</tr>
<tr>
<td>1-KD+KK</td>
<td>Import and translate one DATA key and a single-length AKEK.</td>
</tr>
<tr>
<td>1-KD+*KK</td>
<td>Import and translate one DATA key and a double-length AKEK.</td>
</tr>
<tr>
<td>2-KD</td>
<td>Import and translate two DATA keys.</td>
</tr>
</tbody>
</table>

**inbound_KEK_count**

Direction: Input  
Type: String

An 8-byte area that contains an ASCII count for use in the offset process. The count is an ASCII character string, left-justified, and padded on the right by space characters. ICSF interprets a single space character as a zero counter. The maximum value is 99999999.

**inbound_transport_key_identifier**

Direction: Input/Output  
Type: String
ANSI X9.17 Key Translate (CSNAKTR)

A 64-byte area that contains either an internal token, or a label that refers to an internal token for an AKEK.

**inbound_data_key_1**

Direction: Input  Type: String

A 16-byte area that contains the enciphered DATA key that the service is importing and translating. You must specify the DATA key as an ASCII-encoded hexadecimal string.

**inbound_data_key_2**

Direction: Input  Type: String

A 16-byte area that contains the second enciphered DATA key that the service is importing and translating. This field is valid if the *rule_array* parameter specifies 2-KD. You must supply the key as an ASCII-encoded hexadecimal string. This field is ignored if the *rule_array* parameter specifies other source key rules.

**inbound_key_encrypting_key**

Direction: Input  Type: String

A 16- or 32-byte area that contains an enciphered AKEK that the service is to translate. The area is 16 bytes if the *rule_array* parameter specifies a source key rule of single-length AKEK. The area is 32 bytes if the source key rule specifies a double-length AKEK (1-KD+*KK). You must supply the key as an ASCII-encoded hexadecimal string. ICSF ignores this field if the *rule_array* specifies either 1-KD or 2-KD.

**outbound_origin_identifier**

Direction: Input  Type: String

This parameter is valid if the *rule_array* parameter specifies a keyword of NOTARIZE. It specifies an area that contains a 16-byte string that contains the origin identifier that is defined in the ANSI X9.17 standard. The string must be ASCII characters, left-justified, and padded on the right by space characters. The string must be a minimum of four non-space characters. ICSF ignores this field if the *rule_array* parameter specifies a keyword of CPLT-NOT.

**outbound_destination_identifier**

Direction: Input  Type: String

This parameter is valid if the *rule_array* parameter specifies a keyword of NOTARIZE. It specifies an area that contains a 16-byte string that contains the destination identifier that is defined in the ANSI X9.17 standard. The string must be ASCII characters, left-justified, and padded on the right by space characters. The string must be a minimum of four non-space characters. This parameter is ignored if the *rule_array* parameter specifies a keyword of CPLT-NOT.

**outbound_KEK_count**

Direction: Input  Type: String
ANSI X9.17 Key Translate (CSNAKTR)

An 8-byte area that contains an ASCII count for use in the notarization process. The count is an ASCII character string, left-justified, and padded on the right by space characters. ICSF interprets a single space character as a zero counter. The maximum value is 99999999.

outbound_transport_key_identifier
Direction: Input/Output Type: String

A 64-byte area that contains either an internal token, or a label that refers to an internal token for an AKEK.

outbound_data_key_1
Direction: Output Type: String

A 16-byte area where the service returns the translated data key 1 as an ASCII-encoded hexadecimal string. The service returns the key only if the rule_array parameter specifies 1-KD or 2-KD. ICSF ignores this field if the rule_array parameter specifies either 1-KD+KK or 1-KD+*KK.

outbound_data_key_2
Direction: Output Type: String

A 16-byte area where the service returns the translated data key 2 as an ASCII-encoded hexadecimal string. The service returns the key only if the rule_array parameter specifies 2-KD. ICSF ignores this field if the rule_array parameter specifies 1-KD, 1-KD+KK, or 1-KD+*KK.

outbound_key_encrypting_key
Direction: Output Type: String

A 16- or 32-byte area that contains the enciphered, translated AKEK. The area is 16 bytes if the rule_array parameter specifies a single-length AKEK (1-KD+KK). The area is 32 bytes if the rule_array parameter specifies a double-length AKEK (1-KD+*KK). The service returns the key as an ASCII-encoded hexadecimal string. ICSF ignores this field if the rule_array parameter specifies either 1-KD or 2-KD.

MAC_key_token
Direction: Output Type: String

A 64-byte area that contains an internal token for a MAC key that is intended for use in the MAC generation or MAC verification process. This field is the EXCLUSIVE OR of the two imported DATA keys when the rule_array parameter specifies 2-KD for the source key rule. If the rule_array parameter specifies 1-KD, the service returns the imported key in this field as an ICSF internal key token.

Usage Notes

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

You must install the ANSI system keys in the CKDS to use this service.
ANSI X9.17 Key Translate (CSNAKTR)

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td>This callable service is not supported.</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 890</td>
<td>This callable service is not supported.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td>This callable service is not supported.</td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td>This callable service is not supported.</td>
<td></td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ANSI X9.17 Transport Key Partial Notarize (CSNATKN)

Use the ANSI X9.17 transport key partial notarize callable service to preprocess an ANSI X9.17 transport key-encrypting key with origin and destination identifiers. ICSF completes the notarization process when you use the partially notarized key in the ANSI X9.17 key export, ANSI X9.17 key import, or ANSI X9.17 key translate services and specify the CPLT-NOT rule_array keyword.

**Note:** You cannot reverse the partial notarization process. If you want to keep the original value of the AKEK, you must record the value.

**Restriction:** This service is only supported on an IBM @server zSeries 800 and IBM @server zSeries 900.

**Format**

```plaintext
CALL CSNATKN(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    origin_identifier,
    destination_identifier,
    source_transport_key_identifier,
    target_transport_key_identifier)
```
Parameters

return_code
Direction: Output Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code
Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that are assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

exit_data_length
Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

exit_data
Direction: Input/Output Type: String

The data that is passed to the installation exit.

rule_array_count
Direction: Input Type: Integer

The number of keywords you supplied in the rule_array parameter. Currently no rule_array keywords are defined; thus, this field must be set to 0.

rule_array
Direction: Input Type: String

Currently, no rule_array keywords are defined for this service. You must still specify this parameter for possible future use.

origin_identifier
Direction: Input Type: String

A 16-byte string that contains the origin identifier that is defined in the ANSI X9.17 standard. The string must be ASCII characters, left-justified, and padded on the right by space characters. The string must be a minimum of four non-space characters.

destination_identifier
Direction: Input Type: String
ANSI X9.17 Transport Key Partial Notarize (CSNATKN)

A 16-byte string that contains the destination identifier that is defined in the ANSI X9.17 standard. The string must be ASCII characters, left-justified, and padded on the right by space characters. The string must be a minimum of four non-space characters.

**source_transport_key_identifier**

Direction: Input/Output  Type: String

A 64-byte area that contains either an internal token, or a label of an internal token for an AKEK that permits notarization.

**target_transport_key_identifier**

Direction: Output  Type: String

A 64-byte area where the internal token of a partially notarized AKEK will be returned. This AKEK cannot be used directly as a notarizing KEK until the notarization process has been completed. To do this, specify CPLT-NOT as the rule_array keyword in any service in which you intend to use this key as a notarizing KEK.

**Usage Notes**

SAF may be invoked to verify the caller is authorized to use this callable service, the key label, or internal secure key tokens that are stored in the CKDS or PKDS.

You must install the ANSI system keys in the CKDS to use this service.

This table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

<table>
<thead>
<tr>
<th>Server</th>
<th>Required cryptographic hardware</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM @server zSeries 800</td>
<td>Cryptographic Coprocessor Feature</td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM @server zSeries 990</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z9 EC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM System z9 BC</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM System z10 EC</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
<tr>
<td>IBM System z10 BC</td>
<td></td>
<td>This callable service is not supported.</td>
</tr>
</tbody>
</table>
Part 3. PKCS #11 Callable Services
Chapter 14. Using PKCS #11 Tokens and Objects

This topic describes the callable services for creating and maintaining PKCS #11 tokens and objects. ICSF provides a number of callable services to assist you in managing PKCS #11 tokens and maintaining the token data set (TKDS). Services are also provided for generating, using, and managing key objects.

The following callable services are described:

- "PKCS #11 Derive multiple keys (CSFPDMK)"
- "PKCS #11 Derive key (CSFPDVK)" on page 496
- "PKCS #11 Get attribute value (CSFPGAV)" on page 500
- "PKCS #11 Generate key pair (CSFPGBK)" on page 503
- "PKCS #11 Generate secret key (CSFPGSK)" on page 505
- "PKCS #11 Generate HMAC (CSFPHMG)" on page 507
- "PKCS #11 Verify HMAC (CSFPHMV)" on page 511
- "PKCS #11 One-way hash generate (CSFPOWH)" on page 514
- "PKCS #11 Private key sign (CSFPKPS)" on page 518
- "PKCS #11 Public key verify (CSFPKVP)" on page 520
- "PKCS #11 Pseudo-random function (CSFPPRF)" on page 523
- "PKCS #11 Set attribute value (CSFPSAV)" on page 526
- "PKCS #11 Secret key decrypt (CSFPSKD)" on page 528
- "PKCS #11 Secret key encrypt (CSFPSKE)" on page 532
- "PKCS #11 Token record create (CSFPTRC)" on page 537
- "PKCS #11 Token record delete (CSFPTRD)" on page 541
- "PKCS #11 Token record list (CSFPTRL)" on page 543
- "PKCS #11 Unwrap key (CSFPUWK)" on page 547
- "PKCS #11 Wrap key (CSFPWPK)" on page 550

As of ICSF FMID HCR7770, a TKDS is no longer required to use the PKCS #11 services. If ICSF is started without a TKDS, however, only the omnipresent token will be available. The omnipresent token supports session objects only. Session objects are objects that do not persist beyond the life of a PKCS #11 session.

PKCS #11 Derive multiple keys (CSFPDMK)

Use the PKCS #11 Derive Multiple Keys callable service to generate multiple secret key objects and protocol dependent keying material from an existing secret key object. This service does not support any recovery methods.

The key handle must be a handle of a PKCS #11 secret key object. The CKA_DERIVE attribute for the secret key object must be true. The mechanism keyword specified in the rule array indicates what derivation protocol to use. The derive parms list provides additional input/output data. The format of this list is dependent on the protocol being used.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPDMK6.
PKCS #11 Derive multiple keys (CSFPDMK)

Format

```
CALL CSFPDMK(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    attribute_list_length,
    attribute_list,
    key_handle,
    parms_list_length,
    parms_list)
```

Parameters

**return_code**

Direction: Output  Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

Direction: Output  Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**

Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes-1). The data is defined in the `exit_data` parameter.

**exit_data**

Direction: Input/Output  Type: String

The data that is passed to the installation exit.

**rule_array_count**

Direction: Input  Type: Integer

The number of keywords you supplied in the `rule_array` parameter. This value must be 1.

**rule_array**

Direction: Input  Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.
PKCS #11 Derive multiple keys (CSFPDMK)

Table 204. Keywords for derive multiple keys

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mechanism (required)</td>
<td></td>
</tr>
<tr>
<td>SSL-KM</td>
<td>Use the SSL 3.0 Key and MAC derivation protocol as defined in the PKCS #11 standard as mechanism CKM_SSL3_KEY_AND_MAC_DERIVE.</td>
</tr>
<tr>
<td>TLS-KM</td>
<td>Use the TLS 1.0/1.1 Key and MAC derivation protocol as defined in the PKCS #11 standard as mechanism CKM_TLS_KEY_AND_MAC_DERIVE.</td>
</tr>
</tbody>
</table>

attribute_list_length

Direction: Input  Type: Integer

The length of the attributes supplied in the attribute_list parameter in bytes. The maximum value for this field is 32752.

attribute_list

Direction: Input  Type: String

List of attributes for the generated secret key object(s). See "Attribute List" on page 82 for the format of an attribute_list.

key_handle

Direction: Input  Type: String

The 44-byte handle of the base key object. See "Handles" on page 83 for the format of a key_handle.

parms_list_length

Direction: Input  Type: Integer

The length of the parameters supplied in the parms_list parameter in bytes.

parms_list

Direction: Input/Output  Type: String

Parameters for the SSL-KM and TLS-KM mechanisms.

Table 205. parms_list parameter format for SSL-KM and TLS-KM mechanisms

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length in bytes</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Input</td>
<td>Boolean indicating if “export” processing is required. Any value other than x’00’ means yes</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>Not applicable</td>
<td>reserved</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Input</td>
<td>length in bytes of the client’s random data (x), where 1 &lt;= length &lt;= 32</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>Input</td>
<td>length in bytes of the server’s random data (y), where 1 &lt;= length &lt;= 32</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>Input</td>
<td>size of MAC to be generated in bits, where 8 &lt;= size &lt;= 384, in multiples of 8</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>Input</td>
<td>size of key to be generated in bits, Must match a supported size for the key type specified in the attribute list</td>
</tr>
</tbody>
</table>
Table 205. parms_list parameter format for SSL-KM and TLS-KM mechanisms (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length in bytes</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>4</td>
<td>Input</td>
<td>size of IV to be generated in bits (v), where 0 &lt;= size &lt;= 128, in multiples of 8</td>
</tr>
<tr>
<td>24</td>
<td>44</td>
<td>Output</td>
<td>handle of client MAC secret object created</td>
</tr>
<tr>
<td>68</td>
<td>44</td>
<td>Output</td>
<td>handle of server MAC secret object created</td>
</tr>
<tr>
<td>112</td>
<td>44</td>
<td>Output</td>
<td>handle of client key object created</td>
</tr>
<tr>
<td>156</td>
<td>44</td>
<td>Output</td>
<td>handle of server key object created</td>
</tr>
<tr>
<td>200</td>
<td>x</td>
<td>Input</td>
<td>client's random data</td>
</tr>
<tr>
<td>200+x</td>
<td>y</td>
<td>Input</td>
<td>server's random data</td>
</tr>
<tr>
<td>200+x+y</td>
<td>v/8</td>
<td>Output</td>
<td>client's IV</td>
</tr>
<tr>
<td>200+x+y+v/8</td>
<td>v/8</td>
<td>Output</td>
<td>server's IV</td>
</tr>
</tbody>
</table>

**Authorization**

There are multiple keys involved in this service — the base key and the target keys (the new keys created from the base key).

- To use a base key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).
- To use a base key that is a private object, the caller must have USER (READ) authority (user access).
- To derive a target key that is a public object, the caller must have SO (READ) authority or USER (UPDATE) authority.
- To derive a target key that is a private object, the caller must have SO (CONTROL) authority or USER (UPDATE) authority.

**Usage Notes**

Key derivation operations are performed in software.

**PKCS #11 Derive key (CSFPDVK)**

Use the PKCS #11 Derive Key callable service to generate a new secret key object from an existing key object. This service does not support any recovery methods.

The deriving key handle must be a handle of an existing PKCS #11 key object. The CKA_DERIVE attribute for this object must be true. The mechanism keyword specified in the rule array indicates what derivation protocol to use. The derive parms list provides additional input data. The format of this list is dependent on the protocol being used.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPDVK6.
PKCS #11 Derive key (CSFPDVK)

Format

```c
CALL CSFPDVK(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    attribute_list_length,
    attribute_list,
    base_key_handle,
    parms_list_length,
    parms_list,
    target_key_handle)
```

Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes-1). The data is defined in the `exit_data` parameter.

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**rule_array_count**

Direction: Input  
Type: Integer

The number of keywords you supplied in the `rule_array` parameter. This value must be 1.

**rule array**

Direction: Input  
Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords
must be in contiguous storage.

Table 206. Keywords for derive key

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mechanism (required)</td>
<td></td>
</tr>
<tr>
<td>PKCS-DH</td>
<td>Use the Diffie-Hellman PKCS derivation protocol as defined in the PKCS #11 standard as mechanism CKM_DH_PKCS_DERIVE.</td>
</tr>
<tr>
<td>SSL-MS</td>
<td>Use the SSL 3.0 Master Secret derivation protocol as defined in the PKCS #11 standard as mechanism CKM_SSL3_MASTER_KEY_DERIVE. The SSL protocol version is also returned. The base key must have been generated according to the rules for SSL 3.0</td>
</tr>
<tr>
<td>SSL-MSDH</td>
<td>Use the SSL 3.0 Master Secret for Diffie-Hellman derivation protocol as defined in the PKCS #11 standard as mechanism CKM_SSL3_MASTER_KEY_DERIVE_DH.</td>
</tr>
<tr>
<td>TLS-MS</td>
<td>Use the TLS Master Secret derivation protocol as defined in the PKCS #11 standard as mechanism CKM_TLS_MASTER_KEY_DERIVE. The base key must have been generated according to the rules for TLS 1.0 or TLS 1.1</td>
</tr>
<tr>
<td>TLS-MSDH</td>
<td>Use the TLS Master Secret for Diffie-Hellman derivation protocol as defined in the PKCS #11 standard as mechanism CKM_TLS_MASTER_KEY_DERIVE_DH.</td>
</tr>
<tr>
<td>EC-DH</td>
<td>Use the Elliptic Curve Diffie-Hellman derivation protocol as defined in the PKCS #11 standard as mechanism CKM_ECDH1_DERIVE.</td>
</tr>
</tbody>
</table>

attribute_list_length

Direction: Input Type: Integer

The length of the attributes supplied in the attribute_list parameter in bytes. The maximum value for this field is 32750.

attribute_list

Direction: Input Type: String

List of attributes for the generated secret key object(s). See “Attribute List” on page 82 for the format of an attribute_list.

base_key_handle

Direction: Input Type: String

The 44-byte handle of the source key object. See “Handles” on page 83 for the format of a base_key_handle.

parms_list_length

Direction: Input Type: Integer

The length of the parameters supplied in the parms_list parameter in bytes.

parms_list

Direction: Input/Output Type: String

The protocol specific parameters. This field has a varying format depending on the mechanism specified:
### Table 207. \textit{parms\_list} parameter format for PKCS-DH mechanism

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length in bytes</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>Input</td>
<td>length in bytes of the other party’s public value, where 64 &lt;= length &lt;= 256</td>
</tr>
<tr>
<td>4</td>
<td>&lt;=256</td>
<td>Input</td>
<td>binary value representing the other party’s public value.</td>
</tr>
</tbody>
</table>

### Table 208. \textit{parms\_list} parameter format for SSL-MS, SSL-MSDH, TLS-MS, and TLS-MSDH mechanisms

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length in bytes</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>Output</td>
<td>SSL protocol version returned for SSL-MS and TLS-MS only. For the other protocols, this field is left unchanged.</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>not applicable</td>
<td>reserved</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Input</td>
<td>length in bytes of the client’s random data (x), where 1 &lt;= length &lt;= 32</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>Input</td>
<td>length in bytes of the server’s random data (y), where 1 &lt;= length &lt;= 32</td>
</tr>
<tr>
<td>12</td>
<td>x</td>
<td>Input</td>
<td>client’s random data</td>
</tr>
<tr>
<td>12+x</td>
<td>y</td>
<td>Input</td>
<td>server’s random data</td>
</tr>
</tbody>
</table>

### Table 209. \textit{parms\_list} parameter format for EC-DH mechanism

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length in bytes</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Input</td>
<td>KDF function code, x’01’ = NULL; x’02’ = SHA1, x’05’ = SHA224, x’06’ = SHA256, x’07’ = SHA384, and x’08’ = SHA512</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>not applicable</td>
<td>reserved</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Input</td>
<td>length in bytes of the optional data shared between the two parties. A zero length means no shared data. For the NULL KDF the length must be zero. Otherwise, the maximum shared data length 2147483647.</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>Input</td>
<td>64-bit address of the data shared between the two parties. The data must reside in the caller’s address space. High order word must be set to all zeros by AMODE31 callers. This field is ignored if the length is zero.</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>Input</td>
<td>length in bytes of the other party’s public value (x). This length is dependent on the curve type/size of the base key and on whether the value is DER encoded or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>secp192r1 – 49 (51 w/DER)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>secp224r1 – 57 (59 w/DER)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>secp256r1 – 65 (67 w/DER)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>secp384r1 – 97 (99 w/DER)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>secp521r1 – 133 (136 w/DER)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>brainpoolP160r1 – 41 (43 w/DER)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>brainpoolP192r1 – 49 (51 w/DER)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>brainpoolP224r1 – 57 (59 w/DER)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>brainpoolP256r1 – 65 (67 w/DER)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>brainpoolP320r1 – 81 (83 w/DER)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>brainpoolP384r1 – 97 (99 w/DER)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>brainpoolP512r1 – 129 (132 w/DER)</td>
</tr>
<tr>
<td>20</td>
<td>x&lt;=136</td>
<td>Input</td>
<td>binary value representing the other party’s public value with or without DER encoding.</td>
</tr>
</tbody>
</table>
PKCS #11 Derive key (CSFPDVK)

target_key_handle

Direction: Output       Type: String

Upon successful completion, the 44-byte handle of the secret key object that was derived.

Authorization

There are two keys involved in this service — the base key and the target key (the new key created from the base key).

- To use a base key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).
- To use a base key that is a private object, the caller must have USER (READ) authority (user access).
- To derive a target key that is a public object, the caller must have SO (READ) authority or USER (UPDATE) authority.
- To derive a target key that is a private object, the caller must have SO (CONTROL) authority or USER (UPDATE) authority.

Usage Notes

Key derivation operations are performed in software.

PKCS #11 Get attribute value (CSFPGAV)

Use the get attribute value callable service (CSFPGAV) to retrieve the attributes of an object.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPGAV6.

Format

CALL CSFPGAV(
   return_code,
   reason_code,
   exit_data_length,
   exit_data,
   handle,
   rule_array_count,
   rule_array,
   attribute_list_length,
   attribute_list)

Parameters

return_code

Direction: Output
Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code
PKCS #11 Get attribute value (CSFPAGAV)

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

exit_data_length

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the callable service. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is defined in the exit_data parameter.

exit_data

Direction: Input/Output  
Type: String

Reserved

handle

Direction: Input  
Type: String

The 44-byte handle of the object. See "Handles" on page 83 for the format of a handle.

rule_array_count

Direction: Input  
Type: Integer

The number of keywords supplied in the rule_array parameter. This value must be 0.

rule_array

Direction: Input  
Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

attribute_list_length

Direction: Input/Output  
Type: Integer

On input, the length of the attribute_list parameter in bytes.

On output, the length of the attribute_list parameter in bytes. If the length supplied on input is insufficient to hold all attributes, the length on output is set to the minimum length required.
PKCS #11 Get attribute value (CSFPGAV)

attribute_list

Direction: Output
Type: String

A list of object attributes.

See [Attribute List on page 82] for the format of an attribute_list.

Authorization

The token authorization required and the amount of attribute information returned is dependent on the values of the attributes the object possesses.

The authority to retrieve the non-sensitive attributes is as follows:
- For a public object - any authority to the token (USER (READ) or SO (READ))
- For a private object - USER (READ) or SO (CONTROL)

If the caller is not authorized to retrieve the non-sensitive attributes, the service fails.

If the caller is authorized to retrieve the non-sensitive attributes and the object does not possess any sensitive attributes, the service returns all the object's attributes.

If the caller is authorized to retrieve the non-sensitive attributes and the object does possess sensitive attributes, processing is as defined in this table:

Table 210. Get attribute value processing for objects possessing sensitive attributes

<table>
<thead>
<tr>
<th>Object</th>
<th>PKCS #11 role authority</th>
<th>CKA_SENSITIVE</th>
<th>CKA_EXTRACTABLE</th>
<th>Attributes returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>USER (READ) or SO (READ)</td>
<td>True</td>
<td>True or False</td>
<td>Non-sensitive only</td>
</tr>
<tr>
<td>Private</td>
<td>USER (READ) or SO (CONTROL)</td>
<td>True</td>
<td>True or False</td>
<td>Non-sensitive only</td>
</tr>
<tr>
<td>Public</td>
<td>USER (READ) or SO (READ)</td>
<td>False</td>
<td>False</td>
<td>Non-sensitive only</td>
</tr>
<tr>
<td>Private</td>
<td>USER (READ) or SO (CONTROL)</td>
<td>False</td>
<td>False</td>
<td>Non-sensitive only</td>
</tr>
<tr>
<td>Public</td>
<td>USER (READ) or SO (READ)</td>
<td>False</td>
<td>True</td>
<td>Sensitive and non-sensitive</td>
</tr>
<tr>
<td>Private</td>
<td>SO (CONTROL)</td>
<td>False</td>
<td>True</td>
<td>Non-sensitive only</td>
</tr>
<tr>
<td>Private</td>
<td>USER (READ)</td>
<td>False</td>
<td>True</td>
<td>Sensitive and non-sensitive</td>
</tr>
</tbody>
</table>

Note:
- Session and token objects require the same authority.
- The sensitive attributes are as follows:
  - CKA_VALUE for a secret key, Elliptic Curve private key, DSA private key, or Diffie-Hellman private key object.
  - CKA_PRIVATE_EXPONENT, CKA_PRIME_1, CKA_PRIME_2, CKA_EXPONENT_1, CKA_EXPONENT_2, and CKA_COEFFICIENT for a private key object.
PKCS #11 Get attribute value (CSFPGAV)

- See z/OS Cryptographic Services ICSF Writing PKCS #11 Applications for more information on the SO and User PKCS #11 roles.

Usage Notes

1. If the object is marked sensitive or not extractable, the sensitive attributes are not returned.
2. If the caller is authorized to list the non-sensitive attributes of an object, but not the sensitive ones, the sensitive attributes are not returned.
3. If the caller is not authorized to list the non-sensitive attributes of the object, the service fails.

PKCS #11 Generate key pair (CSFPGKP)

Use the generate key pair callable service to generate an RSA, DSA, Elliptic Curve, or Diffie-Hellman key pair. New token or session objects are created to hold the key pair.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPGKP6.

Format

```plaintext
CALL CSFPGKP(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    token_handle,
    rule_array_count,
    rule_array,
    public_key_attribute_list_length,
    public_key_attribute_list,
    public_key_object_handle,
    private_key_attribute_list_length,
    private_key_attribute_list,
    private_key_object_handle
)
```

Parameters

**return_code**

- Direction: Output
- Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

- Direction: Output
- Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.
PKCS #11 Generate key pair (CSFPGBK)

**exit_data_length**
Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes-1). The data is defined in the exit_data parameter.

**exit_data**
Direction: Input/Output  Type: String

The data that is passed to the installation exit.

**token_handle**
Direction: Input  Type: String

The 44-byte handle of the token of the key objects. See "Handles" on page 83 for the format of a token_handle.

**rule_array_count**
Direction: Input  Type: Integer

The number of keywords you supplied in the rule_array parameter. This value must be 0.

**rule_array**
Direction: Input  Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

**public_key_attribute_list_length**
Direction: Input  Type: Integer

The length of the attributes supplied in the public_key_attribute list parameter in bytes.

**public_key_attribute_list**
Direction: Input  Type: String

List of attributes for the public key object. The maximum value for this field is 32750. See "Attribute List" on page 82 for the format of a public_key_attribute_list.

**public_key_object_handle**
Direction: Output  Type: String

The 44-byte handle of the new public key object.

**private_key_attribute_list_length**
Direction: Input  Type: Integer

The length of the attributes supplied in the private_key_attribute_list parameter in bytes.
PKCS #11 Generate key pair (CSFPGKP)

private_key_attribute_list
Direction: Input Type: String
List of attributes for the private key object. The maximum value for this field is 32750. See "Attribute List" on page 82 for the format of a private_key_attribute_list.

private_key_object_handle
Direction: Output Type: String
The 44-byte handle of the new private key object.

Authorization
To generate a public object, the caller must have SO (READ) authority or USER (UPDATE) authority.

To generate a private object, the caller must have SO (CONTROL) authority or USER (UPDATE) authority.

Usage Notes
The type of key pair generated is determined by the key type attributes in the public_key_attributes_list and private_key_attributes_list parameters.

DSA, Elliptic Curve, and Diffie-Hellman key pairs are generated in software. RSA key pair generation may be done in hardware or software.

PKCS #11 Generate secret key (CSFPGSK)

Use the generate secret key callable service to generate a secret key or set of domain parameters. A new token or session object is created to hold the information.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPGSK6.

Format

```
CALL CSFPGSK(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    handle,
    rule_array_count,
    rule_array,
    attribute_list_length,
    attribute_list,
    parms_list_length,
    parms_list
)
```

Parameters

return_code
Direction: Output Type: Integer
PKCS #11 Generate secret key (CSFPGSK)

The return code specifies the general result of the callable service. Appendix A, “ICSF and TSS Return and Reason Codes” lists the return codes.

reason_code

Direction: Output
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

exit_data_length

Direction: Input/Output
Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the exit_data parameter.

exit_data

Direction: Input/Output
Type: String

The data that is passed to the installation exit.

handle

Direction: Input/Output
Type: String

On input, the 44-byte handle of the token. On output, the 44-byte handle of new secret key object. See “Handles” on page 83 for the format of a handle.

rule_array_count

The number of keywords you supplied in the rule_array parameter. This value must be 1.

rule_array

Direction: Input
Type: String

Keywords that provide control information to the callable service.

Table 211. Keywords for generate secret key

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mechanism (One of the following must be specified)</td>
<td></td>
</tr>
<tr>
<td>SSL</td>
<td>Generate a generic secret key object where the client is using SSL (for CKM_SSL3_PRE_MASTER_KEY_GEN)</td>
</tr>
<tr>
<td>TLS</td>
<td>Generate a generic secret key object where the client is using TLS (for CKM_TLS_PRE_MASTER_KEY_GEN)</td>
</tr>
<tr>
<td>KEY</td>
<td>Generate a secret key object according to the key type attribute in the attributes_list parameter (for CKM_GENERIC_SECRET_KEY_GEN, CKM_DES_KEY_GEN, CKM_DES2_KEY_GEN, CKM_DES3_KEY_GEN, CKM_AES_KEY_GEN, CKM_RC4_KEY_GEN, and CKM_BLOWFISH_KEY_GEN)</td>
</tr>
</tbody>
</table>
PKCS #11 Generate secret key (CSFPGSK)

Table 211. Keywords for generate secret key (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARMS</td>
<td>Generate a domain parameters object according to the key type attribute in the attributes_list parameter (for CKM_DSA_PARAMETER_GEN and CKM_DH_PKCS_PARAMETER_GEN)</td>
</tr>
</tbody>
</table>

attribute_list_length

Direction: Input  
Type: Integer

The length of the attributes supplied in the attribute_list parameter in bytes. The maximum value for this field is 32750.

attribute_list

Direction: Input  
Type: String

List of attributes for the secret key object. See "Attribute List" on page 82 for the format of an attribute_list.

parms_list_length

Direction: Input  
Type: Integer

The length of the parameters supplied in the parms_list parameter in bytes.

parms_list

Direction: Input/Output  
Type: String

The protocol specific parameters. This field has a varying format depending on the mechanism specified:

Table 212. parms_list parameter format for SSL and TLS mechanism

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length in bytes</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>input</td>
<td>SSL or TLS version number in binary, e.g., for version 3.01 this would be x'0301'</td>
</tr>
</tbody>
</table>

For the KEY and PARMS mechanisms, there are no parameters. The parms_list_length parameter must be set to zero for these mechanisms.

Authorization

To generate a public object, the caller must have SO (READ) authority or USER (UPDATE) authority.

To generate a private object, the caller must have SO (CONTROL) authority or USER (UPDATE) authority.

Usage Notes

Domain parameters are generated in software. Secret key generation may be done in hardware or software.

PKCS #11 Generate HMAC (CSFPHMG)

Use the PKCS #11 Generate HMAC callable service to generate a hashed message authentication code (MAC). This service does not support any recovery methods.
The key handle must be a handle of a PKCS #11 generic secret key object. The mechanism keyword specified in the rule array indicates the hash algorithm to use. The CKA_SIGN attribute for the secret key object must be true.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPHMG6.

Format

CALL CSFPHMG(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    text_length,
    text,
    text_id,
    chain_data_length,
    chain_data,
    key_handle,
    hmac_length,
    hmac)

Parameters

return_code

Direction: Output  Type: Integer

The return code specifies the general result of the callable service.  Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code

Direction: Output  Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

exit_data_length

Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes-1). The data is defined in the exit_data parameter.

exit_data

Direction: Input/Output  Type: String

The data that is passed to the installation exit.

rule_array_count

Direction: Input  Type: Integer
PKCS #11 Generate HMAC (CSFPHMG)

The number of keywords you supplied in the rule_array parameter. This value must be 1 or 2.

rule array

Direction: Input

Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 213. Keywords for generate HMAC

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mechanism (required)</td>
<td></td>
</tr>
<tr>
<td>MD5</td>
<td>Generate an HMAC. Use MD5 hashing. Output returned in the hmac parameter is 16 bytes in length.</td>
</tr>
<tr>
<td>SHA-1</td>
<td>Generate an HMAC. Use SHA-1 hashing. Output returned in the hmac parameter is 20 bytes in length.</td>
</tr>
<tr>
<td>SHA-224</td>
<td>Generate an HMAC. Use SHA-224 hashing. Output returned in the hmac parameter is 28 bytes in length.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>Generate an HMAC. Use SHA-256 hashing. Output returned in the hmac parameter is 32 bytes in length.</td>
</tr>
<tr>
<td>SHA-384</td>
<td>Generate an HMAC. Use SHA-384 hashing. Output returned in the hmac parameter is 48 bytes in length.</td>
</tr>
<tr>
<td>SHA-512</td>
<td>Generate an HMAC. Use SHA-512 hashing. Output returned in the hmac parameter is 64 bytes in length.</td>
</tr>
<tr>
<td>SSL3-MD5</td>
<td>Generate a MAC according to the SSL v3 protocol. Use MD5 hashing. Output returned in the hmac parameter is 16 bytes in length.</td>
</tr>
<tr>
<td>SSL3-SHA</td>
<td>Generate a MAC according to the SSL v3 protocol. Use SHA1 hashing. Output returned in the hmac parameter is 20 bytes in length.</td>
</tr>
<tr>
<td>Chaining Selection (Optional)</td>
<td></td>
</tr>
<tr>
<td>FIRST</td>
<td>Specifies this is the first call in a series of chained calls. Intermediate results are stored in the hash field.</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>Specifies this is a middle call in a series of chained calls. Intermediate results are stored in the hash field.</td>
</tr>
<tr>
<td>LAST</td>
<td>Specifies this is the last call in a series of chained calls.</td>
</tr>
<tr>
<td>ONLY</td>
<td>Specifies this is the only call and the call is not chained. This is the default.</td>
</tr>
</tbody>
</table>

text_length

Direction: Input

Type: Integer

Length of the text parameter in bytes. The length can be from 0 to 2147483647.

text

Direction: Input

Type: String

Value for which an HMAC will be generated.
PKCS #11 Generate HMAC (CSFPHMGM)

**text_id**

Direction: Input  
Type: Integer

The ALET identifying the space where the text resides.

**chain_data_length**

Direction: Input/Output  
Type: Integer

The byte length of the `chain_data` parameter. This must be 128 bytes.

**chain_data**

Direction: Input/Output  
Type: String

This field is a 128-byte work area. The chain data permits chaining data from one call to another. ICSF initializes the chain data on a FIRST call and may change it on subsequent MIDDLE and LAST calls. Your application must not change the data in this field between the sequence of FIRST, MIDDLE, and LAST calls for a specific message. The chain data has the following format:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>Flag word</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Meaning when set on</td>
</tr>
<tr>
<td>1-31</td>
<td>32</td>
<td>Reserved for IBM's use</td>
</tr>
<tr>
<td>4</td>
<td>44</td>
<td>Cryptographic state object handle</td>
</tr>
<tr>
<td>48</td>
<td>80</td>
<td>Reserved for IBM's use</td>
</tr>
</tbody>
</table>

**key_handle**

Direction: Input  
Type: String

The 44-byte handle of a generic secret key object. This parameter is ignored for MIDDLE and LAST chaining requests. See "Handles" on page 83 for the format of a `key_handle`.

**hmac_length**

Direction: Ignored  
Type: Integer

Reserved field

**hmac**

Direction: Output  
Type: String

Upon successful completion of an ONLY or LAST request, this field contains the generated HMAC value, left justified. The caller must provide an area large enough to hold the generated HMAC as defined by the mechanism specified. This field is ignored for FIRST and MIDDLE requests.

**Authorization**

To use this service with a public object, the caller must have at least SO (READ) authority or USER (READ) authority (any access).
PKCS #11 Generate HMAC (CSFPHMG)

To use this service with a private object, the caller must have at least USER (READ) authority (user access).

Usage Notes

HMAC operations are performed in software.

If the FIRST rule is used to start a series of chained calls:

- The key used to initiate the chained calls must not be deleted until the chained calls are complete.
- The application should make a LAST call to free ICSF resources allocated. If processing is to be aborted without making a LAST call and the chain_data parameter indicates that a cryptographic state object has been allocated, the caller must free the object by calling CSFPTRD (or CSFPTRD6 for 64-bit callers) passing the state object's handle.

PKCS #11 Verify HMAC (CSFPHMV)

Use the PKCS #11 Verify HMAC callable service to verify a hash message authentication code (MAC). This service does not support any recovery methods.

The key handle must be a handle of a PKCS #11 generic secret key object. The mechanism keyword specified in the rule array indicates the hash algorithm to use. The CKA_VERIFY attribute for the secret key object must be true.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPHMV6.

Format

```
CALL CSFPHMV(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    text_length,
    text,
    text_id,
    chain_data_length,
    chain_data,
    key_handle,
    hmac_length,
    hmac )
```

Parameters

return_code

Direction: Output Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code

Direction: Output Type: Integer
PKCS #11 Verify HMAC (CSFPHMV)

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**

Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes-1). The data is defined in the exit_data parameter.

**exit_data**

Direction: Input/Output  Type: String

The data that is passed to the installation exit.

**rule_array_count**

Direction: Input  Type: Integer

The number of keywords you supplied in the rule_array parameter. This value must be 1 or 2.

**rule array**

Direction: Input  Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

*Table 215. Keywords for verify HMAC*

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mechanism (required)</td>
<td></td>
</tr>
<tr>
<td>MD5</td>
<td>Verify an HMAC. Use MD5 hashing. Data supplied in the hmac parameter must be 16 bytes in length.</td>
</tr>
<tr>
<td>SHA-1</td>
<td>Verify an HMAC. Use SHA-1 hashing. Data supplied in the hmac parameter must be 20 bytes in length.</td>
</tr>
<tr>
<td>SHA-224</td>
<td>Verify an HMAC. Use SHA-224 hashing. Data supplied in the hmac parameter must be 28 bytes in length.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>Verify an HMAC. Use SHA-256 hashing. Data supplied in the hmac parameter must be 32 bytes in length.</td>
</tr>
<tr>
<td>SHA-384</td>
<td>Verify an HMAC. Use SHA-384 hashing. Data supplied in the hmac parameter must be 48 bytes in length.</td>
</tr>
<tr>
<td>SHA-512</td>
<td>Verify an HMAC. Use SHA-512 hashing. Data supplied in the hmac parameter must be 64 bytes in length.</td>
</tr>
<tr>
<td>SSL3-MD5</td>
<td>Verify a MAC according to the SSL v3 protocol. Use MD5 hashing. Data supplied in the hmac parameter must be 16 bytes in length.</td>
</tr>
<tr>
<td>SSL3-SHA</td>
<td>Verify a MAC according to the SSL v3 protocol. Use SHA1 hashing. Data supplied in the hmac parameter must be 20 bytes in length.</td>
</tr>
</tbody>
</table>

Chaining Selection (Optional)
**Table 215. Keywords for verify HMAC (continued)**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIRST</td>
<td>Specifies this is the first call in a series of chained calls. Intermediate results are stored in the hash field.</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>Specifies this is a middle call in a series of chained calls. Intermediate results are stored in the hash field.</td>
</tr>
<tr>
<td>LAST</td>
<td>Specifies this is the last call in a series of chained calls.</td>
</tr>
<tr>
<td>ONLY</td>
<td>Specifies this is the only call and the call is not chained. This is the default.</td>
</tr>
</tbody>
</table>

**text_length**

Direction: Input  
Type: Integer

Length of the text parameter in bytes. The length can be from 0 to 2147483647.

**text**

Direction: Input  
Type: String

Value for which an HMAC will be generated.

**text_id**

Direction: Input  
Type: Integer

The ALET identifying the space where the text resides.

**chain_data_length**

Direction: Input/Output  
Type: Integer

The byte length of the chain_data parameter. This must be 128 bytes.

**chain_data**

Direction: Input/Output  
Type: String

This field is a 128-byte work area. The chain data permits chaining data from one call to another. ICSF initializes the chain data on a FIRST call and may change it on subsequent MIDDLE and LAST calls. Your application must not change the data in this field between the sequence of FIRST, MIDDLE, and LAST calls for a specific message. The chain data has the following format:

**Table 216. chain_data parameter format**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>Flag word</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit Meaning when set on</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 Cryptographic state object has been allocated</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1-31 Reserved for IBM's use</td>
</tr>
<tr>
<td>4</td>
<td>44</td>
<td>Cryptographic state object handle</td>
</tr>
<tr>
<td>48</td>
<td>80</td>
<td>Reserved for IBM's use</td>
</tr>
</tbody>
</table>

**key_handle**

Direction: Input  
Type: String
PKCS #11 Verify HMAC (CSFPHMV)

- The 44-byte handle of a generic secret key object. This parameter is ignored for MIDDLE and LAST chaining requests. See "Handles" on page 83 for the format of a key_handle.

**hmac_length**
- Direction: Ignored
- Type: Integer

**hmac**
- Direction: Input
- Type: String
- This field contains the HMAC value to be verified on ONLY and LAST requests, left justified. The caller must provide an HMAC value of the required length as determined by the mechanism specified. This field is ignored for FIRST and MIDDLE requests.

**Authorization**
- To use this service with a public object, the caller must have at least SO (READ) authority or USER (READ) authority (any access).

- To use this service with a private object, the caller must have at least USER (READ) authority (user access).

**Usage Notes**
- HMAC operations are performed in software.

- Return code 4, reason code 8000 indicates the HMAC didn't verify.

- If the FIRST rule is used to start a series of chained calls:
  - The key used to initiate the chained calls must not be deleted until the chained calls are complete.
  - The application should make a LAST call to free ICSF resources allocated. If processing is to be aborted without making a LAST call and the chain_data parameter indicates that a cryptographic state object has been allocated, the caller must free the object by calling CSFPTRD (or CSFPTRD6 for 64-bit callers) passing the state object's handle.

PKCS #11 One-way hash generate (CSFPOWH)

- Use the one-way hash generate callable service to generate a one-way hash on specified text. This service supports the following methods:
  - MD2 - software only
  - MD5 - software only
  - SHA-1
  - RIPEMD-160 - software only
  - SHA-224
  - SHA-256
  - SHA-384
  - SHA-512

- The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64).
- 64-bit callers must use CSFPOWH6.
PKCS #11 One-way hash generate (CSFPOWH)

Format

CALL CSFPOWH(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    text_length,
    text,
    text_id,
    chain_data_length,
    chain_data,
    handle,
    hash_length,
    hash )

Parameters

return_code
Direction: Output Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code
Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

exit_data_length
Direction: Input/Output Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes-1). The data is defined in the exit_data parameter.

exit_data
Direction: Input/Output Type: String

The data that is passed to the installation exit.

rule_array_count
Direction: Input Type: Integer

The number of keywords you supplied in the rule_array parameter. This value must be 1 or 2.

rule_array
Direction: Input Type: String
Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

<p>| Table 217. Keywords for one-way hash generate |
|-----------------|-----------------|</p>
<table>
<thead>
<tr>
<th><strong>Keyword</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash Method (required)</td>
<td></td>
</tr>
<tr>
<td>MD2</td>
<td>Hash algorithm is MD2 algorithm. Length of hash generated is 16 bytes.</td>
</tr>
<tr>
<td>MD5</td>
<td>Hash algorithm is MD5 algorithm. Length of hash generated is 16 bytes.</td>
</tr>
<tr>
<td>RPMD-160</td>
<td>Hash algorithm is RIPEMD-160. Length of hash generated is 20 bytes.</td>
</tr>
<tr>
<td>SHA-1</td>
<td>Hash algorithm is SHA-1. Length of hash generated is 20 bytes.</td>
</tr>
<tr>
<td>SHA-224</td>
<td>Hash algorithm is SHA-224. Length of hash generated is 28 bytes.</td>
</tr>
<tr>
<td>SHA-256</td>
<td>Hash algorithm is SHA-256. Length of hash generated is 32 bytes.</td>
</tr>
<tr>
<td>SHA-384</td>
<td>Hash algorithm is SHA-384. Length of hash generated is 48 bytes.</td>
</tr>
<tr>
<td>SHA-512</td>
<td>Hash algorithm is SHA-512. Length of hash generated is 64 bytes.</td>
</tr>
<tr>
<td>Chaining Flag (optional)</td>
<td></td>
</tr>
<tr>
<td>FIRST</td>
<td>Specifies this is the first call in a series of chained calls. Intermediate results are stored in the hash field.</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>Specifies this is a middle call in a series of chained calls. Intermediate results are stored in the hash field.</td>
</tr>
<tr>
<td>LAST</td>
<td>Specifies this is the last call in a series of chained calls.</td>
</tr>
<tr>
<td>ONLY</td>
<td>Specifies this is the only call and the call is not chained. This is the default.</td>
</tr>
</tbody>
</table>

**text_length**

Direction: Input  
Type: Integer

The length of the text parameter in bytes.

If you specify the FIRST or MIDDLE keyword, then the text length must be a multiple of the block size of the hash method. For MD2, this is a multiple of 16 bytes. For MD5, RPMD-160, SHA-1, SHA-224, and SHA-256, this is a multiple of 64 bytes. For SHA-384 and SHA-512, this is a multiple of 128 bytes. For ONLY and LAST, this service performs the required padding according to the algorithm specified. The length can be from 0 to 2147483647.

**text**

Direction: Input  
Type: String

Value to be hashed

**text_id**

Direction: Input  
Type: Integer

The ALET identifying the space where the text resides.

**chain_data_length**

Direction: Input/Output  
Type: Integer
PKCS #11 One-way hash generate (CSFPOWH)

The byte length of the chain_data parameter. This must be 128 bytes.

chain_data
Direction: Input/Output  Type: String

This field is a 128-byte work area. The chain data permits chaining data from one call to another. ICSF initializes the chain data on a FIRST call and may change it on subsequent MIDDLE calls. Your application must not change the data in this field between the sequence of FIRST, MIDDLE, and LAST calls for a specific message. The chain data has the following format:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>Flag word</td>
</tr>
<tr>
<td>Bit</td>
<td>Meaning when set on</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Cryptographic state object has been allocated</td>
<td></td>
</tr>
<tr>
<td>1-31</td>
<td>Reserved for IBM's use</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>44</td>
<td>Cryptographic state object handle</td>
</tr>
<tr>
<td>48</td>
<td>80</td>
<td>Reserved for IBM's use</td>
</tr>
</tbody>
</table>

handle
Direction: Input  Type: String

This is the 44-byte name of the token to which this hash operation is related. The first 32 bytes of the handle are meaningful. The remaining 12 bytes are reserved. See "Handles" on page 83 for the format of a handle.

hash_length
Direction: Input  Type: Integer

The length of the supplied hash field in bytes.

For SHA-1 and RPMD-160 this must be at least 20 bytes; for MD2 and MD5 this must be at least 16 bytes. For SHA-224 and SHA-256, this must be at least 32 bytes. Even though the length of the SHA-224 hash is less than SHA-256, the extra bytes are used as a work area during the generation of the hash value. The SHA-224 value is left-justified and padded with 4 bytes of binary zeroes. For SHA-384 and SHA-512, thus must be at least 64 bytes. Even though the length of the SHA-384 hash is less than SHA-512, the extra bytes are used as a work area during the generation of the hash value. The SHA-384 value is left-justified and padded with 16 bytes of binary zeroes.

hash
Direction: Input/Output  Type: String

This field contains the hash, left-justified. The processing of the rest of the field depends on the implementation. If you specify the FIRST or MIDDLE keyword, this field contains the intermediate hash value. Your application must not change the data in this field between the sequence of FIRST, MIDDLE, and LAST calls for a specific message.

Authorization
None
PKCS #11 One-way hash generate (CSFPOWH)

Usage Notes

If the FIRST rule is used to start a series of chained calls, the application should make a LAST call to free ICSF resources allocated. If processing is to be aborted without making a LAST call and the chain_data parameter indicates that a cryptographic state object has been allocated, the caller must free the object by calling CSFPTRD (or CSFPTRD6 for 64-bit callers) passing the state object’s handle.

The CSFSERV resource name that protects this service is CSFOWH, the same resource name used to protect the non-PKCS #11 One Way Hash service.

PKCS #11 Private key sign (CSFPPKS)

Use the PKCS #11 private key sign callable service to:

- Decrypt or sign data using an RSA private key using zero-pad or PKCS #1 v1.5 formatting
- Sign data using a DSA private key
- Sign data using an Elliptic Curve private key in combination with DSA

The key handle must be a handle of a PKCS #11 private key object. When the request type keyword DECRYPT is specified in the rule array, CKA_DECRYPT attribute must be true. When no request type is specified, the CKA_SIGN attribute must be true.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPPKS6.

Format

```
CALL CSFPPKS(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    cipher_value_length,
    cipher_value,
    key_handle,
    clear_value_length,
    clear_value)
```

Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

Direction: Output  
Type: Integer
PKCS #11 Private key sign (CSFPPKS)

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit_data_length**

Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes-1). The data is defined in the exit_data parameter.

**exit_data**

Direction: Input/Output  Type: String

The data that is passed to the installation exit.

**rule_array_count**

Direction: Input  Type: Integer

The number of keywords you supplied in the rule_array parameter. This value may be 1 or 2.

**rule array**

Direction: Input  Type: String

Keywords that provide control information to the callable service.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mechanism</td>
<td>(One of the following must be specified)</td>
</tr>
<tr>
<td>RSA-ZERO</td>
<td>Mechanism is RSA decryption or signature generation using zero-pad formatting</td>
</tr>
<tr>
<td>RSA-PKCS</td>
<td>Mechanism is RSA decryption or signature generation using PKCS #1 v1.5 formatting</td>
</tr>
<tr>
<td>DSA</td>
<td>Mechanism is DSA signature generation</td>
</tr>
<tr>
<td>ECDSA</td>
<td>Mechanism is Elliptic Curve with DSA signature generation</td>
</tr>
<tr>
<td>Request type (optional)</td>
<td></td>
</tr>
<tr>
<td>DECRYPT</td>
<td>The request is to decrypt data. This type of request requires the CKA_DECRYPT attribute to be true. If DECRYPT is not specified, the CKA_SIGN attribute must be true. Valid with RSA only.</td>
</tr>
</tbody>
</table>

**cipher_value_length**

Direction: Input  Type: Integer

Length of the cipher_value parameter in bytes.

**cipher_value**

Direction: Input  Type: String

For decrypt, this is the value to be decrypted. Otherwise this is the value to be signed. For RSA-PKCS signature requests, the data to be signed is expected to be a DER encoded DigestInfo structure. For DSA signature requests, the data
PKCS #11 Private key sign (CSFPKPS)

to be signed is expected to be a SHA1 digest. For ECDSA signature requests, the data to be signed is expected to be a SHA1, SHA224, SHA256, SHA384 or SHA512 digest.

key_handle

Direction: Input
Type: String

The 44-byte handle of a private key object. See "Handles" on page 83 for the format of a key_handle.

clear_value_length

Direction: Input/Output
Type: Integer

Length of the clear_value parameter in bytes. On output, this is updated to be the actual length of the decrypted value or the generated signature.

clear_value

Direction: Output
Type: String

For decrypt, this field will contain the decrypted value. Otherwise this field will contain the generated signature.

Authorization

To use this service with a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object, the caller must have USER (READ) authority (user access).

Usage Notes

DSA operations are performed in software. RSA operations may be done in hardware or software.

Request type DECRYPT is not supported for an Elliptic Curve or DSA public key.

PKCS #11 Public key verify (CSFPKPV)

Use the PKCS #11 public key verify callable service to:
- Encrypt or verify data using an RSA public key using zero-pad or PKCS #1 v1.5 formatting. For encryption, the encrypted data is returned
- Verify a signature using a DSA public key. No data is returned
- Verify a signature using an Elliptic Curve public key in combination with DSA. No data is returned

The key handle must be a handle of a PKCS #11 public key object. When the request type keyword ENCRYPT is specified in the rule array, CKA_ENCRYPT attribute must be true. When no request type is specified, the CKA_VERIFY attribute must be true.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPKPV6.
PKCS #11 Public key verify (CSFPPKV)

Format

```c
CALL CSFPPKV(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    clear_value_length,
    clear_value,
    key_handle,
    cipher_value_length,
    cipher_value
)
```

Parameters

- **return_code**
  - Direction: Output
  - Type: Integer
  - The return code specifies the general result of the callable service. See Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

- **reason_code**
  - Direction: Output
  - Type: Integer
  - The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. See Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

- **exit_data_length**
  - Direction: Input/Output
  - Type: Integer
  - The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes-1). The data is defined in the `exit_data` parameter.

- **exit_data**
  - Direction: Input/Output
  - Type: String
  - The data that is passed to the installation exit.

- **rule_array_count**
  - Direction: Input
  - Type: Integer
  - The number of keywords you supplied in the `rule_array` parameter. This value must be 1 or 2.
PKCS #11 Public key verify (CSFPPKV)

Keywords that provide control information to the callable service.

Table 220. Keywords for public key verify

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mechanism (One of the following must be specified)</td>
<td></td>
</tr>
<tr>
<td>RSA-ZERO</td>
<td>Mechanism is RSA encryption or signature verification using zero-pad formatting</td>
</tr>
<tr>
<td>RSA-PKCS</td>
<td>Mechanism is RSA encryption or signature verification using PKCS #1 v1.5 formatting</td>
</tr>
<tr>
<td>DSA</td>
<td>Mechanism is DSA signature verification</td>
</tr>
<tr>
<td>ECDSA</td>
<td>Mechanism is Elliptic Curve with DSA signature verification</td>
</tr>
</tbody>
</table>

Request type (optional)

| ENCRYPT       | The request is to encrypt data. This type of request requires the CKA_ENCRYPT attribute to be true. If ENCRYPT is not specified, the CKA_VERIFY attribute must be true. Valid with RSA only. |

clear_value_length

Direction: Input  Type: Integer

The length of the clear_value parameter

clear_value

Direction: Input  Type: String

For encrypt, this is the value to be encrypted. Otherwise this is the signature is be verified.

key_handle

Direction: Input  Type: String

The 44-byte handle of public key object. See "Handles" on page 83 for the format of a key_handle.

cipher_value_length

Direction: Input/Output  Type: Integer

For encrypt, on input, this is the length of the cipher_value parameter in bytes. On output, this is updated to be the actual length of the text encrypted into the cipher_value parameter. For signature verification, this is the length of the data to be verified (input only).

cipher_value

Direction: Input/Output  Type: String

For encrypt, this is the encrypted value (output only). For signature verification, this is the data to be verified (input only). For RSA-PKCS signature verification requests, the data to be verified is expected to be a DER encoded DigestInfo structure. For DSA signature verification requests, the data to be verified is expected to be a SHA1 digest. For ECDSA signature verification requests, the data to be verified is expected to be a SHA1, SHA224, SHA256, SHA384 or SHA512 digest.
PKCS #11 Public key verify (CSFPPKV)

Authorization
To use this service with a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object, the caller must have USER (READ) authority (user access).

Usage Notes
DSA operations are performed in software. RSA operations may be done in hardware or software.

Request type ENCRYPT is not supported for an Elliptic Curve or DSA public key.

PKCS #11 Pseudo-random function (CSFPPRF)
Use the PKCS #11 Pseudo-random callable service to generate pseudo-random output of arbitrary length. This service does not support any recovery methods.

The mechanism keyword specified in the rule array indicates what derivation protocol to use. The derive parms list provides additional input/output data. The format of this list is dependent on the protocol being used.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPPRF6.

Format

CALL CSFPPRF(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    handle,
    parms_list_length,
    parms_list,
    prf_output_length,
    prf_output)

Parameters

return_code
Direction: Output Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code
Direction: Output Type: Integer
PKCS #11 Pseudo-random function (CSFPREF)

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

exit_data_length

Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes-1). The data is defined in the exit_data parameter.

exit_data

Direction: Input/Output  Type: String

The data that is passed to the installation exit.

rule_array_count

Direction: Input  Type: Integer

The number of keywords you supplied in the rule_array parameter. This value must be 1.

rule_array

Direction: Input  Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

Table 221. Keywords for derive multiple keys

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mechanism (required)</td>
<td></td>
</tr>
<tr>
<td>TLS-PRF</td>
<td>Use the TLS Pseudo-Random Function derivation protocol as defined in the PKCS #11 standard as mechanism CKM_TLS_PRF. This mechanism derives deterministic random bytes from a caller supplied secret key object and other parameters.</td>
</tr>
<tr>
<td>PRNG</td>
<td>Generate pseudo-random bytes using the best source available. If a secure cryptographic coprocessor that supports RNGL is installed and configured, it will be used to produce true (non-deterministic) random data. Otherwise, a pseudo (deterministic) random algorithm, consistent with ANSI X9.31, will be utilized. If a secure cryptographic coprocessor is installed and configured, it will be used to provide entropy in producing the pseudo-random data. Otherwise, an IBM proprietary entropy algorithm will be used in producing the pseudo-random data</td>
</tr>
</tbody>
</table>

handle

Direction: Input  Type: String

For mechanism TLS-PRF, this is the 44-byte handle of the source secret key object. The CKA_DERIVE attribute for the secret key object must be true. If no key is to be used, set the handle to all blanks.

For mechanism PRNG, this is the 44-byte name of the token to which this operation is related. The first 32 bytes of the handle are meaningful. The remaining 12 bytes are reserved and must be blanks.

See “Handles” on page 83 for the format of a handle.
PKCS #11 Pseudo-random function (CSFPPRF)

**parms_list_length**

Direction: Input  
Type: Integer  

The length of the parameters supplied in the `parms_list` parameter in bytes.

**parms_list**

Direction: Input/Output  
Type: String  

The protocol specific parameters. This field has a varying format depending on the mechanism specified:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length in bytes</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>input</td>
<td>PRF function code – x'00', use combined MD5/SHA1 digest algorithm as defined in TLS 1.0/1.1, otherwise use the following single digest algorithm as defined in TLS 1.2: x'01' = SHA256, x'02' = SHA384, and x'03' = SHA512</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>not applicable</td>
<td>reserved</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>input</td>
<td>length in bytes of the label (x), where 1 &lt;= length &lt;= 256</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>input</td>
<td>length in bytes of the seed (y), where 1 &lt;= length &lt;= 256</td>
</tr>
<tr>
<td>12</td>
<td>x</td>
<td>input</td>
<td>label</td>
</tr>
<tr>
<td>12+x</td>
<td>y</td>
<td>input</td>
<td>seed</td>
</tr>
</tbody>
</table>

For the PRNG mechanism, there are no parameters. The `parms_list_length` parameter must be set to zero for this mechanism.

**prf_output_length**

Direction: Input  
Type: Integer  

The length in bytes of pseudo-random data to be generated and returned in the `prf_output` parameter. The maximum length is 2147483647 bytes.

**prf_output**

Direction: Output  
Type: String  

The pre-allocated area in which the pseudo-random data is returned.

### Authorization

To use this service with a public object for mechanism TLS-PRF, the caller must have at least SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object for mechanism TLS-PRF, the caller must have at least USER (READ) authority (user access).

### Usage Notes

Pseudo-random functions operations are performed in software.

The CSFSERV resource name that protects this service is CSFRNG, the same resource name used to protect the non-PKCS #11 Random Number Generation service.
Use the set attribute value callable service (CSFPSAV) to update the attributes of an object.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPSAV6.

Format

```
CALL CSFPSAV(
   return_code,
   reason_code,
   exit_data_length,
   exit_data,
   handle,
   rule_array_count,
   rule_array,
   attribute_list_length,
   attribute_list)
```

Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. [Appendix A, ICSF and TSS Return and Reason Codes](https://www.ibm.com/support/knowledgecenter/en/SSLTBK_22.5.1/com.ibm.zos.v2r5.la000ke00/ksa2g00/ksa2g00_job_title.html) lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. [Appendix A, ICSF and TSS Return and Reason Codes](https://www.ibm.com/support/knowledgecenter/en/SSLTBK_22.5.1/com.ibm.zos.v2r5.la000ke00/ksa2g00/ksa2g00_job_title.html) lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the callable service. The length can be from X’00000000’ to X’7FFFFFFF’ (2 gigabytes). The data is defined in the exit_data parameter.

**exit_data**

Direction: Input/Output  
Type: String

Reserved

**handle**

PKCS #11 Set attribute value (CSFPSAV)
PKCS #11 Set attribute value (CSFPSAV)

Direction: Input
Type: String

The 44-byte handle of the object. See "Handles" on page 83 for the format of a handle.

rule_array_count

Direction: Input
Type: Integer

The number of keywords supplied in the rule_array parameter. This value must be 0.

rule_array

Direction: Input
Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

attribute_list_length

Direction: Input
Type: Integer

The length of the attribute_list parameter in bytes.

The maximum size in bytes is 32752.

attribute_list

Direction: Input
Type: String

A list of object attributes.

Note: Lengths in the attribute list and attribute structures are unsigned integers.
See "Attribute List" on page 82 for the format of an attribute_list.

Authorization

Table 223. Authorization requirements for the set attribute value callable service

<table>
<thead>
<tr>
<th>Action</th>
<th>Object</th>
<th>Authority required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>Public object, except a CA certificate</td>
<td>USER (UPDATE) or SO (READ)</td>
</tr>
<tr>
<td>Set</td>
<td>Private object, except a CA certificate</td>
<td>USER (UPDATE) or SO (CONTROL)</td>
</tr>
<tr>
<td>Set</td>
<td>Public CA certificate object</td>
<td>USER (CONTROL) or SO (READ)</td>
</tr>
<tr>
<td>Set</td>
<td>Private CA certificate object</td>
<td>USER (CONTROL) or SO (CONTROL)</td>
</tr>
</tbody>
</table>
PKCS #11 Set attribute value (CSFPSAV)

Note:
- Session and token objects require the same authority.
- See z/OS Cryptographic Services ICSF Writing PKCS #11 Applications for more information on the SO and User PKCS #11 roles and how ICSF determines that a certificate is a CA certificate.

Usage Notes

When updating the attributes of an object, all attributes in the template will be processed and the value used is that of the last instance processed.

PKCS #11 Secret key decrypt (CSFPSKD)

Use the PKCS #11 secret key decrypt callable service to decipher data using a clear symmetric key. AES, DES, BLOWFISH, and RC4 are supported. This service supports CBC, ECB, Galois/Counter, and stream modes and PKCS #7 padding.

The key handle must be a handle of a PKCS #11 secret key object. The CKA_DECRYPT attribute must be true.

If the length of output field is too short to hold the output, the service will fail and return the required length of the output field in the clear_text_length parameter.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPSKD6.

Format

```call cfspskd(
  return_code,
  reason_code,
  exit_data_length,
  exit_data,
  rule_array_count,
  rule_array,
  key_handle,
  initialization_vector_length,
  initialization_vector,
  chain_data_length,
  chain_data,
  cipher_text_length,
  cipher_text,
  cipher_text_id,
  clear_text_length,
  clear_text,
  clear_text_id )
```

Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. Appendix A, ICSF and TSS Return and Reason Codes lists the return codes.

**reason_code**

Direction: Output  
Type: Integer
PKCS #11 Secret key decrypt (CSFPSKD)

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

exit_data_length

Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X’00000000’ to X’7FFFFFFF’ (2 gigabytes-1). The data is defined in the exit_data parameter.

exit_data

Direction: Input/Output  Type: String

The data that is passed to the installation exit.

rule_array_count

Direction: Input  Type: Integer

The number of keywords you supplied in the rule_array parameter. This value must be 0, 1, 2, or 3.

rule array

Direction: Input  Type: String

Keywords that provide control information to the callable service.

Table 224. Keywords for secret key decrypt

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encryption Mechanism (Optional. No default. If not specified, mechanism will be taken from key type of secret key. If specified, must match key type)</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>AES algorithm will be used.</td>
</tr>
<tr>
<td>DES</td>
<td>DES algorithm will be used. This is only single-key encryption.</td>
</tr>
<tr>
<td>DES3</td>
<td>DES3 algorithm will be used. This includes double- and triple-key encryption.</td>
</tr>
<tr>
<td>BLOWFISH</td>
<td>BLOWFISH algorithm will be used.</td>
</tr>
<tr>
<td>RC4</td>
<td>RC4 algorithm will be used. This is a stream cipher.</td>
</tr>
<tr>
<td>Processing Rule (optional)</td>
<td></td>
</tr>
<tr>
<td>CBC</td>
<td>Performs cipher block chaining. The cipher text length must be a multiple of the block size for the specified algorithm (8 bytes for DES, DES3, and BLOWFISH, 16 bytes for AES). CBC is the default value for DES, DES3, AES, and BLOWFISH. CBC cannot be specified for RC4.</td>
</tr>
<tr>
<td>CBC-PAD</td>
<td>Performs cipher block chaining. The cipher text length must be greater than zero and a multiple of the block size for the specified algorithm. For FINAL and ONLY calls, PKCS #7 padding is performed. For this reason, the clear text will always be shorter than the cipher text and may even be zero length. CBC-PAD cannot be specified for BLOWFISH or RC4.</td>
</tr>
<tr>
<td>ECB</td>
<td>Performs electronic code book encryption. The cipher text length must be a multiple of the block size for the specified algorithm. ECB cannot be specified for BLOWFISH or RC4.</td>
</tr>
</tbody>
</table>
PKCS #11 Secret key decrypt (CSFPSKD)

Table 224. Keywords for secret key decrypt (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCM</td>
<td>Performs Galois/Counter mode encryption. The cipher text length must be greater than zero. The clear text will be shorter than the cipher text and may even be zero length due to the truncation of the authentication tag. GCM may only be specified with AES. GMAC is a specialized form of GCM where no plain text is returned.</td>
</tr>
<tr>
<td>STREAM</td>
<td>Performs a stream cipher. STREAM cannot be specified for BLOWFISH, DES, DES3, or AES. STREAM is the default value for RC4.</td>
</tr>
</tbody>
</table>

Chaining Selection (optional)

| INITIAL  | Specifies this is the first call in a series of chained calls. For cipher block chaining, the initialization vector is taken from the initialization_vector parameter. Cannot be specified with processing rule ECB or GCM. |
| CONTINUE | Specifies this is a middle call in a series of chained calls. Intermediate results are read from and stored in the chain_data field. Cannot be specified with processing rule ECB or GCM. |
| FINAL    | Specifies this is the last call in a series of chained calls. Intermediate results are read from the chain_data field. Cannot be specified with processing rule ECB or GCM. |
| ONLY     | Specifies this is the only call and the call is not chained. For cipher block chaining, the initialization vector is taken from the initialization_vector parameter. For Galois Counter mode, the initialization parameters are taken from the initialization_vector parameter. ONLY is the default chaining. |

**key_handle**

Direction: Input  
Type: String

The 44-byte handle of secret key object. See "Handles" on page 83 for the format of a key_handle.

**initialization_vector_length**

Direction: Input  
Type: Integer

Length of the initialization_vector in bytes. For CBC and CBC-PAD, this must be 8 bytes for DES and BLOWFISH and 16 bytes for AES. For GCM, this must be the size of the initialization_vector field (28 bytes).

**initialization_vector**

Direction: Input  
Type: String

This field has a varying format depending on the mechanism specified. For CBC and CBC-PAD this is the 8 or 16 byte initial chaining value. The format for GCM is shown in the following table.

Table 225. initialization_vector parameter format for GCM mechanism

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length in bytes</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>Input</td>
<td>length in bytes of the initialization vector. The minimum value is 1. The maximum value is 128. 12 is recommended.</td>
</tr>
</tbody>
</table>
Table 225. initialization_vector parameter format for GCM mechanism (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length in bytes</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8</td>
<td>Input</td>
<td>64-bit address of the initialization vector. The data must reside in the caller’s address space. High order word must be set to all zeros by AMODE31 callers.</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>Input</td>
<td>length in bytes of the additional authentication data. The minimum value is 0. The maximum value is 1048576.</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>Input</td>
<td>64-bit address of the additional authentication data. The data must reside in the caller’s address space. High order word must be set to all zeros by AMODE31 callers. This field is ignored if the length of the additional authentication data is zero.</td>
</tr>
<tr>
<td>24</td>
<td>4</td>
<td>Input</td>
<td>Length in bytes of the desired authentication tag. This value must be one of 4, 8, 12, 13, 14, 15, or 16.</td>
</tr>
</tbody>
</table>

**chain_data_length**

Direction: Input/Output  Type: Integer

The byte length of the chain_data parameter. This must be 128 bytes.

**chain_data**

Direction: Input/Output  Type: String

This field is a 128-byte work area. The chain data permits chaining data from one call to another. ICSF initializes the chain data on an INITIAL call, and may change it on subsequent CONTINUE calls. Your application must not change the data in this field between the sequence of INITIAL, CONTINUE, and FINAL calls for a specific message. The chain data has the following format:

Table 226. chain_data parameter format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>Flag word</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning when set on</td>
</tr>
<tr>
<td>0</td>
<td>44</td>
<td>Cryptographic state object has been allocated</td>
</tr>
<tr>
<td>1-31</td>
<td></td>
<td>Reserved for IBM’s use</td>
</tr>
<tr>
<td>48</td>
<td>80</td>
<td>Cryptographic state object handle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reserved for IBM’s use</td>
</tr>
</tbody>
</table>

**cipher_text_length**

Direction: Input  Type: Integer

Length of the cipher_text parameter in bytes. Except for processing rule GCM, the length can be up to 2147483647. For processing rule GCM, the length cannot exceed 1048576 plus the length of the tag.

**cipher_text**

Direction: Input  Type: String

Text to be decrypted.

**cipher_text_id**

Direction: Input  Type: Integer
PKCS #11 Secret key decrypt (CSFPSKD)

The ALET identifying the space where the cipher text resides.

**clear_text_length**

Direction: Input/Output  
Type: Integer

On input, the length in bytes of the clear_text parameter. On output, the length of the text decrypted into the clear_text parameter.

**clear_text**

Direction: Output  
Type: String

Decrypted text

**clear_text_id**

Direction: Input  
Type: Integer

The ALET identifying the space where the clear_text resides.

**Authorization**

To use this service with a public object, the caller must have at least SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object, the caller must have at least USER (READ) authority (user access).

**Usage Notes**

If the INITIAL rule is used to start a series of chained calls:

- The key used to initiate the chained calls must not be deleted until the chained calls are complete.
- The application should make a FINAL call to free ICSF resources allocated. If processing is to be aborted without making a FINAL call and the chain_data parameter indicates that a cryptographic state object has been allocated, the caller must free the object by calling CSFPTRD (or CSFPTRD6 for 64-bit callers) passing the state object's handle.

GCM decryption may be used to verify a GMAC on some authentication data. To do this request AES decryption with processing rule. The cipher_text_length and cipher_text fields must be set to the length and value of the GMAC to be verified. A return_code of zero and no clear_text data returned means the GMAC verification was successful.

PKCS #11 Secret key encrypt (CSFPSKE)

Use the PKCS #11 secret key encrypt callable service to encipher data using a clear symmetric key. AES, DES, BLOWFISH, and RC4 are supported. This service supports CBC, ECB, Galois/Counter, and stream modes and PKCS #7 padding. The key handle must be a handle of a PKCS #11 secret key object. The CKA_ENCRYPT attribute must be true.

If the length of output field is too short to hold the output, the service will fail and return the required length of the output field in the cipher_text_length parameter.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPSKE6.
PKCS #11 Secret key encrypt (CSFPSKE)

Format

```c
CALL CSFPSKE(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    key_handle,
    initialization_vector_length,
    initialization_vector,
    chain_data_length,
    chain_data,
    clear_text_length,
    clear_text,
    clear_text_id,
    cipher_text_length,
    cipher_text,
    cipher_text_id)
```

Parameters

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. [Appendix A, "ICSF and TSS Return and Reason Codes"] lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. [Appendix A, "ICSF and TSS Return and Reason Codes"] lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes-1). The data is defined in the `exit_data` parameter.

**exit_data**

Direction: Input/Output  
Type: String

The data that is passed to the installation exit.

**rule_array_count**

Direction: Input  
Type: Integer

The number of keywords you supplied in the `rule_array` parameter. This value must be 0, 1, 2, or 3.
## PKCS #11 Secret key encrypt (CSFPSKE)

### rule array

**Direction:** Input  
**Type:** String

Keywords that provide control information to the callable service.

*Table 227. Keywords for secret key encrypt*

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encryption Mechanism (Optional. No default. If not specified, mechanism will be taken from key type of secret key. If specified, must match key type)</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>AES algorithm will be used.</td>
</tr>
<tr>
<td>DES</td>
<td>DES algorithm will be used. This is only single-key encryption.</td>
</tr>
<tr>
<td>DES3</td>
<td>DES3 algorithm will be used, This includes double- and triple-key encryption.</td>
</tr>
<tr>
<td>BLOWFISH</td>
<td>BLOWFISH algorithm will be used.</td>
</tr>
<tr>
<td>RC4</td>
<td>RC4 algorithm will be used. This is a stream cipher.</td>
</tr>
</tbody>
</table>

### Processing Rule (optional)

<table>
<thead>
<tr>
<th>CBC</th>
<th>Performs cipher block chaining. The text length must be a multiple of the block size for the specified algorithm (8 bytes for DES, DES3, and BLOWFISH, 16 bytes for AES). CBC is the default value for DES, DES3, AES, and BLOWFISH. CBC cannot be specified for RC4.</th>
</tr>
</thead>
</table>
| CBC-PAD       | Performs cipher block chaining. Except for FINAL and ONLY chaining calls, the clear text length must be a multiple of the block size for the specified algorithm. For FINAL and ONLY calls:  
- The clear text length may be shorter than the block size and may even be zero.  
- PKCS #7 padding is performed. Thus, the cipher text will always be longer than the clear text.  
CBC-PAD cannot be specified for BLOWFISH or RC4. |
| ECB           | Performs electronic code book encryption. The text length must be a multiple of the block size for the specified algorithm. ECB cannot be specified for BLOWFISH or RC4. |
| GCM           | Performs Galois/Counter mode encryption. The clear text length may be shorter than the block size and may even be zero. The authentication tag is returned appended to the cipher text. GCM may only be specified with AES. GMAC is a specialized form of GCM where no plain text is specified. |
| STREAM        | Performs a stream cipher. STREAM cannot be specified for BLOWFISH, DES, DES3, or AES. STREAM is the default value for RC4. |

### Chaining Selection (optional)

<table>
<thead>
<tr>
<th>INITIAL</th>
<th>Specifies this is the first call in a series of chained calls. For cipher block chaining, the initialization vector is taken from the initialization_vector parameter. Intermediate results are stored in the chain_data field. Cannot be specified with processing rule ECB or GCM.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTINUE</td>
<td>Specifies this is a middle call in a series of chained calls. Intermediate results are read from and stored in the chain_data field. Cannot be specified with processing rule ECB or GCM.</td>
</tr>
</tbody>
</table>
PKCS #11 Secret key encrypt (CSFPSKE)

Table 227. Keywords for secret key encrypt (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FINAL</td>
<td>Specifies this is the last call in a series of chained calls. Intermediate results are read from the chain_data field. Cannot be specified with processing rule ECB or GCM.</td>
</tr>
<tr>
<td>ONLY</td>
<td>Specifies this is the only call and the call is not chained. For cipher block chaining, the initialization vector is taken from the initialization_vector parameter. For Galois Counter mode, the initialization parameters are taken from the initialization_vector parameter. ONLY is the default chaining.</td>
</tr>
</tbody>
</table>

**key_handle**

Direction: Input  
Type: String

The 44-byte handle of secret key object. See "Handles" on page 83 for the format of a key_handle.

**Initialization_vector_length**

Direction: Input  
Type: Integer

Length of the initialization_vector in bytes. For CBC and CBC-PAD, this must be 8 bytes for DES and BLOWFISH and 16 bytes for AES. For GCM, this must be the size of the initialization_vector field (28 bytes).

**initialization_vector**

Direction: Input  
Type: String

This field has a varying format depending on the mechanism specified. For CBC and CBC-PAD this is the 8 or 16 byte initial chaining value. The format for GCM is shown in the following table.

Table 228. initialization_vector parameter format for GCM mechanism

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length in bytes</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>Input</td>
<td>length in bytes of the initialization vector. The minimum value is 1. The maximum value is 128. 12 is recommended.</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>Input</td>
<td>64-bit address of the initialization vector. The data must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers.</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>Input</td>
<td>length in bytes of the additional authentication data. The minimum value is 0. The maximum value is 1048576.</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>Input</td>
<td>64-bit address of the additional authentication data. The data must reside in the caller's address space. High order word must be set to all zeros by AMODE31 callers. This field is ignored if the length of the additional authentication data is zero.</td>
</tr>
<tr>
<td>24</td>
<td>4</td>
<td>Input</td>
<td>Length in bytes of the desired authentication tag. This value must be one of 4, 8, 12, 13, 14, 15, or 16.</td>
</tr>
</tbody>
</table>

**chain_data_length**

Direction: Input/Output  
Type: Integer

The byte length of the chain_data parameter. This must be 128 bytes.
PKCS #11 Secret key encrypt (CSFPSKE)

### chain_data

**Direction:** Input/Output  
**Type:** String

This field is a 128-byte work area. The chain data permits chaining data from one call to another. ICSF initializes the chain data on an INITIAL call, and may change it on subsequent CONTINUE calls. Your application must not change the data in this field between the sequence of INITIAL, CONTINUE, and FINAL calls for a specific message. The chain data has the following format:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>Flag word</td>
</tr>
<tr>
<td>1-31</td>
<td>44</td>
<td>Cryptographic state object handle</td>
</tr>
<tr>
<td>4</td>
<td>80</td>
<td>Reserved for IBM's use</td>
</tr>
</tbody>
</table>

### clear_text_length

**Direction:** Input  
**Type:** Integer

Length of the clear_text parameter in bytes. Except for processing rule GCM, the length can be up to 2147483647. For processing rule GCM, the length cannot exceed 1048576.

### clear_text

**Direction:** Input  
**Type:** String

Text to be encrypted

### clear_text_id

**Direction:** Input  
**Type:** Integer

The ALET identifying the space where the clear_text resides.

### cipher_text_length

**Direction:** Input/Output  
**Type:** Integer

On input, the length in bytes of the clear_text parameter. On output, the length of the text encrypted into the cipher_text parameter.

### cipher_text

**Direction:** Output  
**Type:** String

Encrypted text

### cipher_text_id

**Direction:** Output  
**Type:** Integer

The ALET identifying the space where the cipher_text resides.
PKCS #11 Secret key encrypt (CSFPSKE)

Authorization
To use this service with a public object, the caller must have at least SO (READ) authority or USER (READ) authority (any access).

To use this service with a private object, the caller must have at least USER (READ) authority (user access).

Usage Notes
If the INITIAL rule is used to start a series of chained calls:

- The key used to initiate the chained calls must not be deleted until the chained calls are complete.
- The application should make a FINAL call to free ICSF resources allocated. If processing is to be aborted without making a FINAL call and the chain_data parameter indicates that a cryptographic state object has been allocated, the caller must free the object by calling CSFPTRD (or CSFPTRD6 for 64-bit callers) passing the state object's handle.

GCM encryption may be used to produce a GMAC on some authentication data. To do this request AES encryption with processing rule. The clear_text_length field must be set to zero. The authentication tag (the GMAC) is returned in the cipher_text field.

PKCS #11 Token record create (CSFPTRC)
Use the token record create callable service (CSFPTRC) to do these tasks:

- Initialize or re-initialize a z/OS PKCS #11 token
- Create or copy a token object in the token data set
- Create or copy a session object for the current PKCS #11 session

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPTRC6.

Format

```
CALL CSFPTRC(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    handle,
    rule_array_count,
    rule_array,
    attribute_list_length,
    attribute_list)
```

Parameters

**return_code**

Direction: Output
Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.
PKCS #11 Token record create (CSFPTRC)

**reason_code**

- **Direction:** Output
- **Type:** Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, “ICSF and TSS Return and Reason Codes” lists the reason codes.

**exit_data_length**

- **Direction:** Input/Output
- **Type:** Integer

The length of the data that is passed to the callable service. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is defined in the exit_data parameter.

**exit_data**

- **Direction:** Input/Output
- **Type:** String

Reserved

**handle**

- **Direction:** Input/Output
- **Type:** String

On input, the 44-byte name of the z/OS PKCS #11 token to be initialized, or the token handle of the object to be created or copied. For the create or re-create functions, the first 32 bytes of the handle are meaningful on input. The remaining 12 bytes are filled in by the token record create service. For the copy function, all 44 bytes of the handle are significant on input.

On output, the 44-byte handle of the z/OS PKCS #11 token or object created.

See "Handles on page 83" for the format of a handle.

**rule_array_count**

- **Direction:** Input
- **Type:** Integer

The number of keywords supplied in the rule_array parameter. The value must be 1 or 2.

**rule_array**

- **Direction:** Input
- **Type:** String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.
PKCS #11 Token record create (CSFPTRC)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>One of these two keywords must be specified:</td>
<td></td>
</tr>
<tr>
<td>TOKEN</td>
<td>Specifies that a token is to be initialized. If the token exists in the token data set, the RECREATE keyword must be specified.</td>
</tr>
<tr>
<td>OBJECT</td>
<td>Specifies that an object (token object or session object) is to be created. If the object is to be a copy of an existing object, the COPY keyword must be specified.</td>
</tr>
<tr>
<td>This keyword is optional, and valid only with TOKEN:</td>
<td></td>
</tr>
<tr>
<td>RECREATE</td>
<td>Specifies that the token exists and is to be re-initialized. All objects of the existing token will be deleted.</td>
</tr>
<tr>
<td>This keyword is optional, and valid only with OBJECT:</td>
<td></td>
</tr>
<tr>
<td>COPY</td>
<td>Specifies that the object specified by the handle is to be copied into a new object.</td>
</tr>
</tbody>
</table>

attribute_list_length

Direction: Input
Type: Integer

Length of the attribute_list parameter in bytes.
The maximum size in bytes is 32752.

attribute_list

Direction: Input
Type: String

List of token or object attributes.

When creating or re-creating a token, the attribute_list parameter has this format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 31</td>
<td>Manufacturer ID</td>
</tr>
<tr>
<td>32 - 47</td>
<td>Model</td>
</tr>
<tr>
<td>48 - 63</td>
<td>Serial number</td>
</tr>
<tr>
<td>64 - 67</td>
<td>Reserved for IBM's use. Must be hexadecimal zeros.</td>
</tr>
</tbody>
</table>

Note: The strings supplied for Manufacturer ID, Model, and Serial number are assumed to be from code page IBM1047.

For objects, see "Attribute List" on page 82 for the format of an attribute_list.
PKCS #11 Token record create (CSFPTRC)

Authorization

Note: Session and token objects require the same SAF authority.

Table 230. Authorization requirements for the token record create callable service

<table>
<thead>
<tr>
<th>Action</th>
<th>Source object (Copy only)</th>
<th>Token / Object being created</th>
<th>PKCS #11 role Authority required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create or recreate token</td>
<td>N/A</td>
<td>Token</td>
<td>SO (UPDATE)</td>
</tr>
<tr>
<td>Create object</td>
<td>N/A</td>
<td>Public object, except a CA certificate</td>
<td>USER (UPDATE) or SO (READ)</td>
</tr>
<tr>
<td>Create object</td>
<td>N/A</td>
<td>Private object, except a CA certificate</td>
<td>USER (UPDATE) or SO (CONTROL)</td>
</tr>
<tr>
<td>Create object</td>
<td>N/A</td>
<td>Public CA certificate object</td>
<td>USER (CONTROL) or SO (READ)</td>
</tr>
<tr>
<td>Create object</td>
<td>N/A</td>
<td>Private CA certificate object</td>
<td>USER (CONTROL) or SO (CONTROL)</td>
</tr>
<tr>
<td>Copy object</td>
<td>Public object, except a CA certificate</td>
<td>Public object, except a CA certificate</td>
<td>USER (UPDATE) or SO (READ)</td>
</tr>
<tr>
<td>Copy object</td>
<td>Public object or private object, except a CA certificate</td>
<td>Private object, except a CA certificate</td>
<td>USER (UPDATE) or SO (CONTROL)</td>
</tr>
<tr>
<td>Copy object</td>
<td>Private object, except a CA certificate</td>
<td>Public object, except a CA certificate</td>
<td>USER (UPDATE)</td>
</tr>
<tr>
<td>Copy object</td>
<td>Public object, where source or target or both are CA certificate objects</td>
<td>Public object, where source or target or both are CA certificate objects</td>
<td>USER (CONTROL) or SO (READ)</td>
</tr>
<tr>
<td>Copy object</td>
<td>Public object or private object, where source or target or both are CA certificate objects</td>
<td>Private object, where source or target or both are CA certificate objects</td>
<td>USER (CONTROL) or both USER (UPDATE) and SO (READ)</td>
</tr>
<tr>
<td>Copy object</td>
<td>Private object, where source or target or both are CA certificate objects</td>
<td>Public object, where source or target or both are CA certificate objects</td>
<td>USER (CONTROL) or both USER (UPDATE) and SO (READ)</td>
</tr>
</tbody>
</table>

Note:
- Session and token objects require the same authority.
- See z/OS Cryptographic Services ICSF Writing PKCS #11 Applications for more information on the SO and User PKCS #11 roles and on how ICSF determines that a certificate is a CA certificate.

Usage Notes

When creating an object, these attribute processing rules will be in effect:
- All attributes will be processed and the value of the last instance of an attribute in the template will be saved.

When copying an object, these attribute processing rules will be in effect:
PKCS #11 Token record create (CSFPTRC)

- All attributes will be processed and the value of the last instance of an attribute in the template will be saved except for CKA_EXTRACTABLE and CKA_SENSITIVE. CKA_EXTRACTABLE will be copied from the source object and may be set to False if the value in the source object is True. CKA_SENSITIVE will be copied from the source object and may be set to True if the value in the source object is False.

PKCS #11 Token record delete (CSFPTRD)

Use the token record delete callable service (CSFPTRD) to delete a z/OS PKCS #11 token, token object, session object, or state object. When a token is deleted, all associated objects are deleted as well. The deletions occur in the token data set (TKDS), and all session memory areas in the ICSF address space.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPTRD6.

**Format**

```plaintext
CALL CSFPTRD(
   return_code,
   reason_code,
   exit_data_length,
   exit_data,
   handle,
   rule_array_count,
   rule_array)
```

**Parameters**

**return_code**

Direction: Output  
Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

Direction: Output  
Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer

The length of the data that is passed to the callable service. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is defined in the exit_data parameter.

**exit_data**
PKCS #11 Token record delete (CSFPTRD)

Direction: Input/Output
Type: String

Reserved

**handle**

Direction: Input
Type: String

44-byte name of the token or object to be deleted. See "Handles" on page 83 for the format of a *handle*.

**rule_array_count**

Direction: Input
Type: Integer

The number of keywords supplied in the *rule_array* parameter. This value must be 1.

**rule_array**

Direction: Input
Type: String

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>One of these two keywords must be specified:</td>
<td></td>
</tr>
<tr>
<td>TOKEN</td>
<td>Specifies that a token and all associated objects are to be deleted.</td>
</tr>
<tr>
<td>OBJECT</td>
<td>Specifies that an object is to be deleted.</td>
</tr>
</tbody>
</table>

**Authorization**

*Table 231. Authorization requirements for the token record delete callable service*

<table>
<thead>
<tr>
<th>Token / Object Type</th>
<th>PKCS #11 Role Authority Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token</td>
<td>SO (UPDATE)</td>
</tr>
<tr>
<td>Public object, except CA certificate</td>
<td>USER (UPDATE) or SO (READ)</td>
</tr>
<tr>
<td>Private object, except CA certificate</td>
<td>USER (UPDATE) or SO (CONTROL)</td>
</tr>
<tr>
<td>Public CA certificate object</td>
<td>USER (CONTROL) or SO (READ)</td>
</tr>
<tr>
<td>Private CA certificate object</td>
<td>USER (CONTROL) or SO (CONTROL)</td>
</tr>
<tr>
<td>State object</td>
<td>None</td>
</tr>
</tbody>
</table>

**Note:**

- Session and token objects require the same authority.
- See [z/OS Cryptographic Services ICSF Writing PKCS #11 Applications](/zos) for more information on the SO and User PKCS #11 roles and how ICSF determines that a certificate is a CA certificate.
PKCS #11 Token record list (CSFPTRL)

Use the token record list callable service (CSFPTRL) to:

- Obtain a list of z/OS PKCS #11 tokens. The caller must have SAF authority to
  the token for a particular token to be listed.
- Obtain a list of token and session objects for a token. Use a search template to
  restrict the search for specific attributes. The caller must have SAF authority to
  the token.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64).
64-bit callers must use CSFPTRL6.

Format

```call csfptrl(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    handle,
    rule_array_count,
    rule_array,
    search_template_length,
    search_template,
    list_length,
    handle_count,
    output_list)
```

Parameters

**return_code**

- Direction: Output
- Type: Integer

The *return code* specifies the general result of the callable service. Appendix A,
"ICSF and TSS Return and Reason Codes" lists the return codes.

**reason_code**

- Direction: Output
- Type: Integer
PKCS #11 Token record list (CSFPTRL)

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

**exit_data_length**

Direction: Input/Output  
Type: Integer  

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is defined in the exit_data parameter.

**exit_data**

Direction: Input/Output  
Type: String  

The data that is passed to the installation exit.

**handle**

Direction: Input  
Type: String  

For tokens, an empty string (blanks) for the first call, or the 44-byte handle of the last token found for subsequent calls.

For objects, the 44-byte handle of the token for the first call, or the 44-byte handle of the last object found for subsequent calls.

See Usage Notes for more information. See "Handles" on page 83 for the format of a handle.

**rule_array_count**

Direction: Input  
Type: Integer  

The number of keywords supplied in the rule_array parameter. This value must be 1 or 2.

**rule_array**

Direction: Input  
Type: String  

Keywords that provide control information to the callable service. Each keyword is left-justified in 8-byte fields and padded on the right with blanks. All keywords must be in contiguous storage.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN (required)</td>
<td>Specifies that the list will contain all tokens to which the caller has SAF access. The search_template parameter is ignored.</td>
</tr>
</tbody>
</table>
Keyword | Meaning
---|---
OBJECT | Specifies that the list will contain the handles of all objects that match the attributes specified in the search_template parameter and to which the caller has SAF access.

List options (optional, valid only with OBJECT)

| ALL | Specifies that when listing objects, both public and private objects that meet the search criteria should be listed if the caller has SAF authority for the token. There may be no sensitive attributes in the search template. See the Authorization topic for details.

search_template_length

Direction: Input
Type: Integer
The length of the search_template parameter in bytes. The value must be 0 when the TOKEN keyword is specified.

The maximum size in bytes is 32752.

search_template

Direction: Input
Type: String
A list of criteria (attribute values) that an object must meet to be added to the list. If the search_template_length parameter is 0, no criteria are checked.

See "Attribute List" on page 82 for the format of an attribute_list.

list_length

Direction: Input/Output
Type: Integer
On input, the length in bytes of the output_list parameter. On output, the number of bytes used for the output_list parameter. If the supplied length is insufficient to hold one record, the list_length parameter is set to the minimum length required for a record.

handle_count

Direction: Input/Output
Type: Integer
On input, the maximum number of tokens or object handles to return in the list. On output, the actual number of tokens or object handles in the list.

output_list

Direction: Output
Type: String
A list of token names and descriptions or a list of object handles meeting the search criteria.

PKCS #11 Token record list (CSFPTRL)

Chapter 14. Using PKCS #11 Tokens and Objects  545
PKCS #11 Token record list (CSFPTRL)

Authorization

To list tokens, the caller must have at least USER (READ) or SO (READ) authority.

Authority to list objects depends on the object’s attributes and the search criteria as follows:

- To list secret key or private key objects where sensitive key attributes are specified in the search template, this must be true:
  - The object must be marked CKA_SENSITIVE=F and CKA_EXTRACTABLE=T and
  - The caller must have USER (READ) authority
- Otherwise (no sensitive attributes in the search criteria)
  - To list public objects, the caller must have at least USER (READ) or SO (READ) authority
  - To list private objects when the ALL rule array keyword is specified, the caller must have at least USER (READ) or SO (READ) authority
  - To list private objects when the ALL rule array keyword is not specified, the caller must have USER (READ) or SO (CONTROL) authority

<table>
<thead>
<tr>
<th>Token / Object Type</th>
<th>Sensitive Attributes in search criteria</th>
<th>ALL Rule Specified</th>
<th>PKCS #11 Role Authority Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token</td>
<td>N/A</td>
<td>N/A</td>
<td>USER (READ) or SO (READ)</td>
</tr>
<tr>
<td>Public object</td>
<td>No</td>
<td>N/A</td>
<td>USER (READ) or SO (READ)</td>
</tr>
<tr>
<td>Private object</td>
<td>No</td>
<td>No</td>
<td>USER (READ) or SO (CONTROL)</td>
</tr>
<tr>
<td>Private object</td>
<td>No</td>
<td>Yes</td>
<td>USER (READ) or SO (READ)</td>
</tr>
<tr>
<td>Secret key or Private key object (public or private object class) CKA_SENSITIVE=F and CKA_EXTRACTABLE=T</td>
<td>Yes</td>
<td>N/A</td>
<td>USER (READ)</td>
</tr>
<tr>
<td>Secret key or Private key object (public or private object class) CKA_SENSITIVE=T or CKA_EXTRACTABLE=F</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A (object is not listed)</td>
</tr>
</tbody>
</table>

Note:

- Session and token objects require the same authority.
- When the caller does not possess sufficient authority to list a given token or object, that record is skipped. (No information for the token or object is returned.) Processing continues with the next token or object.
- The sensitive attributes are as follows:
  - CKA_VALUE for a secret key object, Elliptic Curve private key, DSA private key, or Diffie-Hellman private key object.
  - CKA_PRIVATE_EXPONENT, CKA_PRIME_1, CKA_PRIME_2, CKA_EXPONENT_1, CKA_EXPONENT_2, and CKA_COEFFICIENT for an RSA private key object.
- See z/OS Cryptographic Services ICSF Writing PKCS #11 Applications for more information on the SO and USER PKCS #11 roles.
PKCS #11 Token record list (CSFPTRL)

Usage Notes

For tokens:

On the initial call to get a list of tokens, the handle parameter should be all blanks. On subsequent calls, the handle parameter should be the last token handle from the output_list returned in the previous call.

The output records are in this format:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 31</td>
<td>Token name</td>
</tr>
<tr>
<td>32 - 63</td>
<td>Manufacturer ID</td>
</tr>
<tr>
<td>64 - 79</td>
<td>Model</td>
</tr>
<tr>
<td>80 - 95</td>
<td>Serial number</td>
</tr>
<tr>
<td>96 - 103</td>
<td>Date that the token information or any token object was last updated, expressed as Coordinated Universal Time (UCT) in the format yyyymmdd</td>
</tr>
<tr>
<td>104 - 111</td>
<td>Time that the token information or any token object was last updated, expressed as Coordinated Universal Time (UCT) in the format hhmmsssth</td>
</tr>
<tr>
<td>112 - 115</td>
<td>Flags</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning when set on</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Token is write protected.</td>
</tr>
</tbody>
</table>

For objects:

On the initial call to get a list of object handles matching the search template, the handle parameter contains the token handle. On subsequent calls, the handle parameter should contain the last object handle from the output_list returned in the previous call. The output records are the 44-byte handles of the objects.

PKCS #11 Unwrap key (CSFPWUK)

Use unwrap key callable service to unwrap and create a key object using another key. The following formatting is supported:

- PKCS 1.2 formatting is supported for a DES, DES3, AES, BLOWFISH, RC4, or GENERIC secret wrapped by an RSA public key.
  - A new secret key object is created with the decrypted key value
  - The unwrapping key must be a private key object
  - The CKA_UNWRAP attribute must be true
- PKCS 8 formatting (CBC mode with padding) is supported for an RSA, DSA, Elliptic Curve, and Diffie-Hellman private key wrapped by a secret key.
  - A new private key object is created with the decrypted key values
  - The unwrapping key must be a secret key object
  - The CKA_UNWRAP attribute must be true
  - The encryption mechanism must be specified in the rule array and must match the key type of the secret key object

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPWUK6.
PKCS #11 Unwrap key (CSFPUWK)

Format

```c
CALL CSFPUWK(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    wrapped_key_length,
    wrapped_key,
    initialization_vector_length,
    initialization_vector,
    unwrapping_key_handle,
    attribute_list_length,
    attribute_list,
    target_key_handle )
```

Parameters

return_code

Direction: Output  Type: Integer

The return code specifies the general result of the callable service. [Appendix A, ICSF and TSS Return and Reason Codes](#) lists the return codes.

reason_code

Direction: Output  Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. [Appendix A, ICSF and TSS Return and Reason Codes](#) lists the reason codes.

exit_data_length

Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes-1). The data is defined in the `exit_data` parameter.

exit_data

Direction: Input/Output  Type: String

The data that is passed to the installation exit.

rule_array_count

Direction: Input  Type: Integer

The number of keywords you supplied in the `rule_array` parameter. This value must be 1 or 2.

rule_array

Direction: Input  Type: String
PKCS #11 Unwrap key (CSFPUWK)

Keywords that provide control information to the callable service.

Table 232. Keywords for unwrap key

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formatting Method (required)</td>
<td>RSA PKCS #1 block type 02 will be used to recover the key value.</td>
</tr>
<tr>
<td>PKCS-1.2</td>
<td>The private key values are DER encoded as specified by PKCS-8. The encryption mechanism rule array keyword must be specified.</td>
</tr>
<tr>
<td>PKCS-8</td>
<td>For PKCS-8 processing, the unwrapping key must be a DES secret key object.</td>
</tr>
<tr>
<td>Encryption Mechanism (required when PKCS-8 specified, ignored otherwise)</td>
<td>For PKCS-8 processing, the unwrapping key must be a DES secret key object.</td>
</tr>
</tbody>
</table>

wrapped_key_length

Direction: Input  Type: Integer

Length of the wrapped key in the wrapped_key parameter.

wrapped_key

Direction: Input  Type: String

The key to be unwrapped.

initialization_vector_length

Direction: Input  Type: Integer

The length of the initialization_vector parameter. The initial value can only be used with PKCS-8. This parameter is ignored for PKCS-1.2. The length must match the key type of the wrapping key (8 for DES, DES2, DES3 and 16 for AES). If the length is zero, the initialization_vector parameter is ignored and an initial value of zero is used.

initialization_vector

Direction: Input  Type: String

The initial chaining value for symmetric encryption. The length must match the key type of the wrapping key. The initial value can only be used with PKCS-8. This parameter is ignored for PKCS-1.2.

unwrapping_key_handle

Direction: Input  Type: String

The 44-byte handle of the private key or secret key object to unwrap the key. See "Handles" on page 83 for the format of a unwrapping_key_handle.

attribute_list_length

Direction: Input  Type: Integer

Length of the attribute_list parameter in bytes. The maximum value for this field is 32750.
PKCS #11 Unwrap key (CSFPUWK)

**attribute_list**

- **Direction:** Input
- **Type:** String

List of token or object attributes for the target key. The attributes must be consistent with the class of the object. See “Attribute List” on page 82 for the format of an attribute_list.

**target_key_handle**

- **Direction:** Output
- **Type:** String

The 44-byte handle of the secret key or private key object created for the unwrapped key. The object will use to token name of the unwrapping key object.

**Authorization**

There are two keys involved in this service: the unwrapping key and the target key (the new key created from the wrapped key).

- To use an unwrapping key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).
- To use an unwrapping key that is a private object, the caller must have USER (READ) authority (user access).
- To unwrap a target key that is a public object, the caller must have SO (READ) authority or USER (UPDATE) authority.
- To unwrap a target key that is a private object, the caller must have SO (CONTROL) authority or USER (UPDATE) authority.

PKCS #11 Wrap key (CSFPWPK)

Use wrap key callable service to wrap a key with another key. The following formatting is supported:

- PKCS 1.2 is supported for wrapping a DES, DES3, AES, BLOWFISH, RC4, or GENERIC secret key with an RSA public key.
  - The wrapping key must be a public key object.
  - The CKA_WRAP attribute must be true.
- PKCS 8 formatting (CBC mode with padding) is supported for wrapping an RSA, DSA, Elliptic Curve, or Diffie-Hellman private key with a secret key.
  - The wrapping key must be a secret key object.
  - The CKA_WRAP attribute must be true
  - The encryption mechanism must be specified in the rule array and must match the key type of the secret key object.

If the length of output field is too short to hold the output, the service will fail and return the required length of the output field in the wrapped_key_length parameter.

The callable service can be invoked in AMODE(24), AMODE(31), or AMODE(64). 64-bit callers must use CSFPWPK6.
PKCS #11 wrap key (CSFPWPK)

Format

```
CALL CSFPWPK(
    return_code,
    reason_code,
    exit_data_length,
    exit_data,
    rule_array_count,
    rule_array,
    source_key_handle,
    wrapping_key_handle,
    initialization_vector_length,
    initialization_vector,
    wrapped_key_length,
    wrapped_key)
```

Parameters

return_code

Direction: Output  Type: Integer

The return code specifies the general result of the callable service. Appendix A, "ICSF and TSS Return and Reason Codes" lists the return codes.

reason_code

Direction: Output  Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes that indicate specific processing problems. Appendix A, "ICSF and TSS Return and Reason Codes" lists the reason codes.

exit_data_length

Direction: Input/Output  Type: Integer

The length of the data that is passed to the installation exit. The length can be from X’00000000’ to X’7FFFFFFF’ (2 gigabytes-1). The data is defined in the exit_data parameter.

exit_data

Direction: Input/Output  Type: String

The data that is passed to the installation exit.

rule_array_count

Direction: Input  Type: Integer

The number of keywords you supplied in the rule_array parameter. This value must be 1 or 2.

rule array

Direction: Input  Type: String
PKCS #11 wrap key (CSFPWPK)

Keywords that provide control information to the callable service.

Table 233. Keywords for wrap key

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formatting Method (required)</td>
<td></td>
</tr>
<tr>
<td>PKCS-1.2</td>
<td>RSA PKCS #1 block type 02 will be used to format the key value.</td>
</tr>
<tr>
<td>PKCS-8</td>
<td>The private key values are DER encoded as specified by PKCS-8. The encryption mechanism rule array keyword must be specified.</td>
</tr>
</tbody>
</table>

Encryption Mechanism (required when PKCS-8 specified, ignored otherwise)

| AES       | For PKCS-8 processing, the wrapping key must be an AES secret key object. |
| DES       | For PKCS-8 processing, the wrapping key must be a DES secret key object. |
| DES3      | For PKCS-8 processing, the wrapping key must be a DES2 or DES3 secret key object. |

source_key_handle

Direction: Input Type: String

The 44-byte handle of the secret key or private key object to be wrapped.

wrapping_key_handle

Direction: Input Type: String

The 44-byte handle of the public key or secret key object to wrap the secret key. See "Handles" on page 83 for the format of a wrapping_key_handle.

Initialization_vector_length

Direction: Input Type: Integer

The length of the initialization_vector parameter. The initialization vector can only be used with PKCS-8. This parameter is ignored for PKCS-1.2. The length must match the key type of the wrapping key (8 for DES, DES2, DES3 and 16 for AES). If the length is zero, the initialization vector parameter is ignored and a value of zero is used.

Initialization_vector

Direction: Input Type: String

The initial chaining value for symmetric encryption. The length must match the key type of the wrapping key. The initial value can only be used with PKCS-8. This parameter is ignored for PKCS-1.2.

wrapped_key_length

Direction: Input/Output Type: Integer

On input, the length of the wrapped_key parameter. On output, the actual length of the wrapped key returned in the wrapped_key parameter.

wrapped_key

Direction: Output Type: String

The wrapped key
PKCS #11 wrap key (CSFPWPK)

Authorization

There are two key objects used by this service, the source key (the key to be wrapped) and the wrapping key.

- To wrap a source key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).
- To wrap a source key that is a private object, the caller must have USER (READ) authority (user access).
- To use a wrapping key that is a public object, the caller must have SO (READ) authority or USER (READ) authority (any access).
- To use a wrapping key that is a private object, the caller must have USER (READ) authority (user access).
PKCS #11 wrap key (CSFPWPK)
Part 4. Appendixes
Appendix A. ICSF and TSS Return and Reason Codes

This topic includes this information:

- Return codes and reason codes issued on the completion of a call to an ICSF callable service
- Return codes and reason codes issued on the completion of a process on a PCI Cryptographic Accelerator, PCI Cryptographic Coprocessor or PCI X Cryptographic Coprocessor/Crypto Express2 Coprocessor /Crypto Express3 Coprocessor (referred to as cryptographic accelerators or coprocessors).
- ICSF return and reason codes can be specified in the installation options data set on the REASONCODES parameter. If the REASONCODES option is not specified, the default of REASONCODES(ICSF) is used. A REASONCODES line in the description indicates a conversion was done as a result of the REASONCODES option in the installation options data set.

If you specified REASONCODES(ICSF) and your service was processed on a PCICC, PCIXCC, CEX2C, or CEX3C, a TSS reason code may be returned if there is no 1–1 corresponding ICSF reason code.

Return Codes and Reason Codes

This topic describes return codes and reason codes.

The TSS return and reason codes have been merged with the ICSF codes in this release. If there is a REASONCODES line in the description, it will indicate an alternate reason code you should investigate.

Each return code returns unique reason codes to your application program. The reason codes associated with each return code are described in these topics. The reason code tables present the hexadecimal code followed by the decimal code in parenthesis.

Return Codes

Table 234 lists return codes from the ICSF callable services.

Table 234. Return Codes

<table>
<thead>
<tr>
<th>Return Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return Code 0 (0)</td>
<td>The call to the service was successfully processed. See the reason code for more information.</td>
</tr>
<tr>
<td>Return Code 4 (4)</td>
<td>The call to the service was successfully processed, but some minor event occurred during processing. See the reason code for more information.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Review the reason code.</td>
</tr>
<tr>
<td>Return Code 8 (8)</td>
<td>The call to the service was unsuccessful. The parameters passed into the call are unchanged, except for the return code and reason code. There are rare examples where output areas are filled, but their contents are not guaranteed to be accurate. These are described under the appropriate reason code descriptions. The reason code identifies which error was found.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Review the reason code, correct the problem, and retry the call.</td>
</tr>
</tbody>
</table>
Table 234. Return Codes (continued)

<table>
<thead>
<tr>
<th>Return Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C (12)</td>
<td>The call to the service could not be processed because ICSF was not active, ICSF found something wrong in its environment, a TSS security product is not available, or a processing error occurred in a TSS product. The parameters passed into the call are unchanged, except for the return code and reason code.</td>
</tr>
</tbody>
</table>

**User action:** Review the reason code and take the appropriate action.

| Return Code 10 (16)     | The call to the service could not be processed because ICSF found something seriously wrong in its environment or a processing error occurred in the PCICC, PCIXCC, CEX2C, or CEX3C. The parameters passed into the call are unchanged, except for the return code and reason code. |

**User action:** Review the reason code and contact your system programmer.

Reason Codes for Return Code 0 (0)

Table 235 lists reason codes returned from callable services that give return code 0.

Table 235. Reason Codes for Return Code 0 (0)

<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>The call to the ICSF callable service was successfully processed. No error was encountered.</td>
</tr>
</tbody>
</table>

**User action:** None.

| 2 (2)                    | The call to the ICSF callable service was successfully processed. A minor error was detected. A key used in the service did not have odd parity. This key could be one provided by you as a parameter or be one (perhaps of many) that was retrieved from the in-storage CKDS. |

**User action:** Refer to the reason code obtained when the key passed to this service was transformed into operational form using clear key import, multiple clear key import, key import, secure key import, or multiple secure key import callable services. Check if any of the services prepared an even parity key. If one of these service reported an even parity key, you need to know which key is affected. If none of these services identified an even parity key, then the even parity key detected was found on the CKDS. Report this to your administrator.

**REASONCODES:** ICSF 4(4)

| 4 (4)                    | The call to the ICSF callable service was successfully processed. A minor error was detected. A key used in the service did not have odd parity. This key could be one provided by you as a parameter or be one (perhaps of many) that was retrieved from the in-storage CKDS. |

**User action:** Refer to the reason code obtained when the key passed to this service was transformed into operational form using clear key import, multiple clear key import, key import, secure key import, or multiple secure key import callable services. Check if any of the services prepared an even parity key. If one of these service reported an even parity key, you need to know which key is affected. If none of these services identified an even parity key, then the even parity key detected was found on the CKDS. Report this to your administrator.

**REASONCODES:** TSS 2(2)

| 8 (8)                    | The key record read callable service attempted to read a NULL key record. The returned key token contains only binary zeros. |

**User action:** None required.
Table 235. Reason Codes for Return Code 0 (0) (continued)

<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC2 (3010)</td>
<td>The call to CSFIQF was successful. Additionally, the PCICC, PCIXCC, CEX2C, or CEX3C adapter is disabled by TKE.</td>
</tr>
<tr>
<td>2710 (10000)</td>
<td>The call to the callable service was successfully processed. The keys in one or more key identifiers have been reenciphered from encipherment under the old master key to encipherment under the current master key.</td>
</tr>
<tr>
<td>User action: If you obtained your operational token from a file, replace the token in the file with the token just returned from ICSF.</td>
<td></td>
</tr>
<tr>
<td>Management of internal tokens is a user responsibility. Consider the possible case where the token for this call was fetched from a file, and where this reason code is ignored. For the next invocation of the service, the token will be fetched from the file again, and the service will give this reason code again. If this continues until the master key is changed again, then the next use of the internal token will fail.</td>
<td></td>
</tr>
<tr>
<td>2711 (10001)</td>
<td>The call to the callable service was successfully processed. The keys in one or more key identifiers were encrypted under the old master key. The callable service was unable to reencipher the key.</td>
</tr>
</tbody>
</table>

Table 236. Reason Codes for Return Code 4 (4)

Table 236 lists reason codes returned from callable services that give return code 4.

<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 (1)</td>
<td>The verification test failed.</td>
</tr>
<tr>
<td>REASONCODES: This reason code also corresponds to these ICSF reason codes: FA0 (4000), 1F40 (8000), 1F44 (8004), 2328 (9000), 232C (9004), 2AF8 (11000), or 36B8 (14008).</td>
<td></td>
</tr>
<tr>
<td>013 (19)</td>
<td>This is a combination reason code value. The call to the Encrypted PIN verify (PINVER) callable service was successfully processed. However, the trial PIN that was supplied does not match the PIN in the PIN block.</td>
</tr>
<tr>
<td>User action: The PIN is incorrect. If you expected the reason code to be zero, check that you are using the correct key.</td>
<td></td>
</tr>
<tr>
<td>REASONCODES: ICSF BD4 (3028)</td>
<td></td>
</tr>
<tr>
<td>In addition, a key in a key identifier token has been reenciphered.</td>
<td></td>
</tr>
<tr>
<td>User action: See reason code 10000 (return code 0) for more detail about the key reencipherment.</td>
<td></td>
</tr>
<tr>
<td>014 (20)</td>
<td>The input text length was odd rather than even. The right nibble of the last byte is padded with 'X'00'.</td>
</tr>
<tr>
<td>User action: None</td>
<td></td>
</tr>
<tr>
<td>REASONCODES: ICSF 7D0 (2000)</td>
<td></td>
</tr>
<tr>
<td>0A6 (166)</td>
<td>The control vector is not valid because of parity bits, anti-variant bits, inconsistent KEK bits, or because bits 59 to 62 are not zero.</td>
</tr>
<tr>
<td>0B3 (179)</td>
<td>The control vector keywords that are in the rule array are ignored.</td>
</tr>
<tr>
<td>Reason Code Hex (Decimal)</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>1AD (429)</td>
<td>The digital signature verify ICSF callable service completed successfully but the supplied digital signature failed verification.</td>
</tr>
<tr>
<td>7D0 (2000)</td>
<td>The input text length was odd rather than even. The right nibble of the last byte is padded with X'00'.</td>
</tr>
<tr>
<td>81E (2078)</td>
<td>The call to Key Record Read was successful. The key label exists in the CKDS. The key label contains a clear DES or AES key token and is not returned to the caller.</td>
</tr>
<tr>
<td>BBA (3002)</td>
<td>The call to the CVV Verify callable service was successfully processed. However, the trial CVV that was supplied does not match the generated CVV. In addition, a key in the key identifier has been reenciphered.</td>
</tr>
<tr>
<td>BC9 (3017)</td>
<td>The call to create a list of information completed successfully, however the storage supplied for the list was insufficient to hold the complete list.</td>
</tr>
<tr>
<td>BD4 (3028)</td>
<td>The call to the Encrypted PIN verify (PINVER) callable service was successfully processed. However, the trial PIN that was supplied does not match the PIN in the PIN block.</td>
</tr>
<tr>
<td>BD8 (3032)</td>
<td>This is a combination reason code value. The call to the Encrypted PIN verify (PINVER) callable service was successfully processed. However, the trial PIN that was supplied does not match the PIN in the PIN block.</td>
</tr>
<tr>
<td>BFC (3068)</td>
<td>The verification pattern of an encrypted CPACF key block doesn't match the current wrapping key's verification pattern.</td>
</tr>
<tr>
<td>FA0 (4000)</td>
<td>The CVV did not verify.</td>
</tr>
<tr>
<td>1F40 (8000)</td>
<td>The call to the MAC verification (MACVER) callable service was successfully processed. However, the trial MAC that you supplied does not match that of the message text.</td>
</tr>
<tr>
<td>Reason Code Hex (Decimal)</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>1F44 (8004)</td>
<td>This is a combination reason code value. The call to the MAC verification (MACVER) callable service was successfully processed. However, the trial MAC that was supplied does not match the message text provided. In addition, a key in a key identifier token has been reenciphered. <strong>User action:</strong> See reason code 8000 (return code 4) for more detail about the incorrect MAC. See reason code 10000 (return code 0) for more detail about the key reencipherment. <strong>REASONCODES:</strong> TSS 01 (01)</td>
</tr>
<tr>
<td>2328 (9000)</td>
<td>The call to the key test service processed successfully, but the key test pattern was not verified. <strong>User action:</strong> Investigate why the key failed. When determining this, you can reinstall or regenerate the key. <strong>REASONCODES:</strong> TSS 01 (01)</td>
</tr>
<tr>
<td>232C (9004)</td>
<td>This is a combination reason code value. The call to the key test service processed successfully, but the key test pattern was not verified. Also, the key token has been reenciphered. <strong>User action:</strong> Investigate why the key failed. When determining this, you can reinstall or regenerate the key. <strong>REASONCODES:</strong> TSS 01 (01)</td>
</tr>
<tr>
<td>2AF8 (11000)</td>
<td>The digital signature verify ICSF callable service completed successfully but the supplied digital signature failed verification. <strong>User action:</strong> None. <strong>REASONCODES:</strong> TSS 1AD (429)</td>
</tr>
<tr>
<td>36B8 (14008)</td>
<td>The PKDS record failed the authentication test. <strong>User action:</strong> The record has changed since ICSF wrote it to the PKDS. The user action is application dependent. <strong>REASONCODES:</strong> TSS 01 (01)</td>
</tr>
</tbody>
</table>

**Reason Codes for Return Code 8 (8)**

Table 237 on page 562 lists reason codes returned from callable services that give return code 8.

Most of these reason codes indicate that the call to the service was unsuccessful. No cryptographic processing took place. Therefore, no output parameters were filled. Exceptions to this are noted in the descriptions.
### Table 237. Reason Codes for Return Code 8 (8)

<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 00C (12)                 | A key identifier was passed to a service or token. It is checked in detail to ensure that it is a valid token, and that the fields within it are valid values. There is a token validation value (TVV) in the token, which is a non-cryptographic value. This value was again computed from the rest of the token, and compared to the stored TVV. If these two values are not the same, this reason code is returned.  
  **User action:** The contents of the token have been altered because it was created by ICSF or TSS. Review your program to see how this could have been caused. |
| 016 (22)                 | The ID number in the request field is not valid. The PAN data is incorrect for VISA CVV.                                                                                                                     |
| 017 (23)                 | Offset length not correct for data to be inserted.                                                                                                                                                           |
| 018 (24)                 | A key identifier was passed to a service. The master key verification pattern in the token shows that the key was created with a master key that is neither the current master key nor the old master key. Therefore, it cannot be reenciphered to the current master key.  
  **User action:** Re-import the key from its importable form (if you have it in this form), or repeat the process you used to create the operational key form. If you cannot do one of these, you cannot repeat any previous cryptographic process that you performed with this token. |
| 019 (025)                | A length parameter has an incorrect value. The value in the length parameter could have been zero (when a positive value was required) or a negative value. If the supplied value was positive, it could have been larger than your installation's defined maximum, or for MDC generation with no padding, it could have been less than 16 or not an even multiple of 8.  
  **User action:** Check the length you specified. If necessary, check your installation's maximum length with your ICSF administrator. Correct the error. |
| 01D (29)                 | A key identifier was passed to a service or token. It is checked in detail to ensure that it is a valid token, and that the fields within it are valid values. There is a token validation value (TVV) in the token, which is a non-cryptographic value. This value was again computed from the rest of the token, and compared to the stored TVV. If these two values are not the same, this reason code is returned.  
  **User action:** The contents of the token have been altered because it was created by ICSF or TSS. Review your program to see how this could have been caused. |
| 01E (30)                 | A key label was supplied for a key identifier parameter. This label is the label of a key in the in-storage CKDS or the PKDS. Either the key could not be found, or a key record with that label and the specific type required by the ICSF callable service could not be found. For a retained key label, this error code is also returned if the key is not found in the PCICC, PCIXCC, CEX2C, or CEX3C specified in the PKDS record.  
  **User action:** Check with your administrator if you believe that this key should be in the in-storage CKDS or the PKDS. The administrator may be able to bring it into storage. If this key cannot be in storage, use a different label. |
| 01F (31)                 | The control vector did not specify a DATA key.                                                                                                                                                                |

**REASONCODES:** ICSF 2714 (10004)
<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>020 (32)</td>
<td>You called the key record create callable service, but the <em>key_label</em> parameter syntax was incorrect.</td>
</tr>
<tr>
<td>User action: Correct <em>key_label</em> syntax.</td>
<td></td>
</tr>
<tr>
<td>REASONCODES: ICSF 3EA0 (16032)</td>
<td></td>
</tr>
<tr>
<td>021 (33)</td>
<td>The <em>rule_array</em> parameter contents or a parameter value is not correct.</td>
</tr>
<tr>
<td>User action: Refer to the <em>rule_array</em> parameter described in this publication under the appropriate callable service for the correct value.</td>
<td></td>
</tr>
<tr>
<td>REASONCODES: ICSF 7E0 (2016)</td>
<td></td>
</tr>
<tr>
<td>022 (34)</td>
<td>A <em>rule_array</em> keyword combination is not valid.</td>
</tr>
<tr>
<td>REASONCODES: ICSF 7E0 (2016)</td>
<td></td>
</tr>
<tr>
<td>023 (35)</td>
<td>The <em>rule_array_count</em> parameter contains a number that is not valid.</td>
</tr>
<tr>
<td>User action: Refer to the <em>rule_array_count</em> parameter described in this publication under the appropriate callable service for the correct value.</td>
<td></td>
</tr>
<tr>
<td>REASONCODES: ICSF 7DC (2012)</td>
<td></td>
</tr>
<tr>
<td>027 (39)</td>
<td>A control vector violation occurred.</td>
</tr>
<tr>
<td>REASONCODES: This reason code also corresponds to these ICSF reason codes: 272C (10028), 2730 (10032), 2734 (10036), 2744 (10052), 2768 (10088), 278C (10124), 3E90 (16016), 2724 (10020).</td>
<td></td>
</tr>
<tr>
<td>028 (40)</td>
<td>The service code does not contain numerical data.</td>
</tr>
<tr>
<td>REASONCODES: ICSF BE0 (3040)</td>
<td></td>
</tr>
<tr>
<td>029 (41)</td>
<td>The <em>key_form</em> parameter is neither IM nor OP. Most constants, these included, can be supplied in lower or uppercase. Note that this parameter is 4 bytes long, so the value IM or OP is not valid. They must be padded on the right with blanks.</td>
</tr>
<tr>
<td>User action: Review the value provided and change it to IM or OP, as required.</td>
<td></td>
</tr>
<tr>
<td>02A (42)</td>
<td>The expiration date is not numeric (X’F0’ through X’F9’). The parameter must be character representations of numerics or hexadecimal data.</td>
</tr>
<tr>
<td>User action: Review the numeric parameters or fields required in the service that you called and change to the format and values required.</td>
<td></td>
</tr>
<tr>
<td>REASONCODES: ICSF BE0 (3040)</td>
<td></td>
</tr>
<tr>
<td>02B (43)</td>
<td>The <em>key_length</em> parameter passed to the key generate callable service holds a value that is not valid.</td>
</tr>
<tr>
<td>User action: Review the value provided and change it as appropriate.</td>
<td></td>
</tr>
<tr>
<td>REASONCODES: See also the ICSF reason code 80C (2060) or 2710 (10000) for additional information.</td>
<td></td>
</tr>
<tr>
<td>02C (44)</td>
<td>The key record create callable service requires that the key created not already exist in the CKDS. A key of the same label was found.</td>
</tr>
<tr>
<td>User action: Make sure the application specifies the correct label. If the label is correct, contact your ICSF security administrator or system programmer.</td>
<td></td>
</tr>
<tr>
<td>Reason Code Hex (Decimal)</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 02D (45) | An input character is not in the code table.  
**User action:** Correct the code table or the source text. |
| 02F (47) | A source key token is unusable because it contains data that is not valid or undefined.  
**REASONCODES:** This reason code also corresponds to these ICSF reason codes: 83C (2108), 2754 (10068), 2758 (10072), 275C (10076), 2AFC (11004), 2B04 (11012), 2B08 (11016), 2B10 (11024). Please see those reason codes for additional information. |
| 030 (48) | One or more keys has a master key verification pattern that is not valid.  
This reason code also corresponds to these ICSF reason codes: 2714 (10004) and 2B0C (11020). Please see those reason codes for additional information. |
| 031 (49) | Key identifiers contain a version number. The version number in a supplied key identifier (internal or external) is inconsistent with one or more fields in the key identifier, making the key identifier unusable.  
**User action:** Use a token containing the required version number.  
**REASONCODES:** ICSF 2738 (10040) |
| 033 (51) | The encipher and decipher callable services sometime require text (plaintext or ciphertext) to have a length that is an exact multiple of 8 bytes. Padding schemes always create ciphertext with a length that is an exact multiple of 8. If you want to decipher ciphertext that was produced by a padding scheme, and the text length is not an exact multiple of 8, then an error has occurred. The CBC mode of enciphering requires a text length that is an exact multiple of 8.  
The ciphertext translate callable service cannot process ciphertext whose length is not an exact multiple of 8.  
The value that the `text_length` parameter specifies is not a multiple of the cryptographic algorithm block length.  
**User action:** Review the requirements of the service you are using. Either adjust the text you are processing or use another process rule. |
| 038 (56) | The master key verification pattern in the OCV is not valid. |
| 03D (61) | The keyword supplied with the `key_type` parameter is not valid.  
**REASONCODES:** This reason code also corresponds to these ICSF reason codes: 2720 (10016), 2740 (10048), 274C (10060). Please see those reason codes for additional information. |
| 03E (62) | The source key was not found.  
**REASONCODES:** ICSF 271C (10012) |
| 03F (63) | This check is based on the first byte in the key identifier parameter. The key identifier provided is either an internal token, where an external or null token was required; or an external or null token, where an internal token was required. The token provided may be none of these, and, therefore, the parameter is not a key identifier at all. Another cause is specifying a `key_type` of IMP-PKA for a key in importable form.  
**User action:** Check the type of key identifier required and review what you have provided. Also check that your parameters are in the required sequence.  
**REASONCODES:** ICSF 7F8 (2040) |
<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 040 (64)                 | The supplied private key can be used only for digital signature. Key management services are disallowed.  
User action: Supply a key with key management enabled.  
OR  
This service requires an RSA private key that is for signature use. The specified key may be used for key management purposes only.  
User action: Re-invoke the service with a supported private key.  
OR  
This service requires an RSA private key that is translatable. The specified key may not be used in the PKA Key Translate callable service.  
User action: Re-invoke the service with a supported private key. To make a key translatable, XLATE-OK must be turned on. |
| 041 (65)                 | The RSA public or private key specified a modulus length that is incorrect for this service.  
User action: Re-invoke the service with an RSA key with the proper modulus length.  
REASONCODES: ICSF 2B18 (11032) and 2B58 (11096) |
| 042 (66)                 | The recovered encryption block was not a valid PKCS-1.2 or zero-pad format. (The format is verified according to the recovery method specified in the rule-array.) If the recovery method specified was PKCS-1.2, refer to PKCS-1.2 for the possible error in parsing the encryption block.  
User action: Ensure that the parameters passed to CSNDSYI or CSNFSYI are correct. Possible causes for this error are incorrect values for the RSA private key or incorrect values in the RSA_enciphered_key parameter, which must be formatted according to PKCS-1.2 or zero-pad rules when created.  
REASONCODES: ICSF 2B20 (11040) |
| 043 (67)                 | DES or RSA encryption failed. |
| 044 (68)                 | DES or RSA decryption failed. |
| 048 (72)                 | The value specified for length parameter for a key token, key, or text field is not valid.  
User action: Correct the appropriate length field parameter.  
REASONCODES: This reason code also corresponds to these ICSF reason codes: 2AF8 (11000) and 2B14 (11028). Please see those reason codes for additional information. |
| 05A (90)                 | Access is denied for this request.  
User action: If access to the service is to be allowed, enable the required access control point(s) via the TKE. |
| 064 (100)                | A request was made to the Clear PIN generate or Encrypted PIN verify callable service, and the PIN_length parameter has a value outside the valid range. The valid range is from 4 to 16, inclusive.  
User action: Correct the value in the PIN_length parameter to be within the valid range from 4 to 16.  
REASONCODES: ICSF BBC (3004) |
Table 237. Reason Codes for Return Code 8 (8) (continued)

<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>065 (101)</td>
<td>A request was made to the Clear PIN generate callable service, and the <code>PIN_check_length</code> parameter has a value outside the valid range. The valid range is from 4 to 16, inclusive. <strong>User action</strong>: Correct the value in the <code>PIN_check_length</code> parameter to be within the valid range from 4 to 16. <strong>REASONCODES</strong>: ICSF BC0 (3008)</td>
</tr>
<tr>
<td>066 (102)</td>
<td>The value of the decimalization table is not valid. <strong>REASONCODES</strong>: ICSF BE0 (3040)</td>
</tr>
<tr>
<td>067 (103)</td>
<td>The value of the validation date is not valid. <strong>REASONCODES</strong>: ICSF BE0 (3040)</td>
</tr>
<tr>
<td>068 (104)</td>
<td>The value of the customer-selected PIN is not valid or the PIN length does not match the value specified. <strong>REASONCODES</strong>: ICSF BE0 (3040)</td>
</tr>
<tr>
<td>069 (105)</td>
<td>A request was made to the Clear PIN generate callable service, and the <code>PIN_check_length</code> parameter has a value outside the valid range. The valid range is from 4 to 16, inclusive. <strong>User action</strong>: Correct the value in the <code>PIN_check_length</code> parameter to be within the valid range from 4 to 16. <strong>REASONCODES</strong>: ICSF BE0 (3040)</td>
</tr>
<tr>
<td>06A (106)</td>
<td>A request was made to the Encrypted PIN Translate or the Encrypted PIN verify callable service, and the PIN block value in the <code>input_PIN_profile</code> or <code>output_PIN_profile</code> parameter has a value that is not valid. <strong>User action</strong>: Correct the PIN block value.</td>
</tr>
<tr>
<td>06B (107)</td>
<td>A request was made to the Encrypted PIN Translate callable service and the format control value in the <code>input_PIN_profile</code> or <code>output_PIN_profile</code> parameter has a value that is not valid. The valid values are NONE or PBVC. <strong>User action</strong>: Correct the format control value to either NONE or PBVC.</td>
</tr>
<tr>
<td>06C (108)</td>
<td>The value of the PAD data is not valid. <strong>REASONCODES</strong>: ICSF B08 (3016)</td>
</tr>
<tr>
<td>06D (109)</td>
<td>The extraction method keyword is not valid.</td>
</tr>
<tr>
<td>06E (110)</td>
<td>The value of the PAD data is not numeric character date. <strong>REASONCODES</strong>: ICSF BE0 (3040)</td>
</tr>
<tr>
<td>06F (111)</td>
<td>A request was made to the Encrypted PIN Translate callable service. The <code>sequence_number</code> parameter was required, but was not the integer value 99999. <strong>User action</strong>: Specify the integer value 99999.</td>
</tr>
<tr>
<td>074 (116)</td>
<td>The supplied PIN value is incorrect. <strong>User action</strong>: Correct the PIN value. <strong>REASONCODES</strong>: ICSF BBC (3004)</td>
</tr>
<tr>
<td>Reason Code Hex (Decimal)</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 079 (121)                | The source_key_identifier or inbound_key_identifier you supplied is not a valid string.  
**User action:** In an ANSI X9.17 service, check that you specified a valid ASCII string for the source_key_identifier or inbound_key_identifier parameter. In the PKA key generate service, an invalid exponent or modulus length was specified. |
| 07A (122)                | The outbound KEK_count or inbound KEK_count you supplied is not a valid ASCII hexadecimal string.  
**User action:** Check that you specified a valid ASCII hexadecimal string for the outbound KEK_count or inbound KEK_count parameter. |
| 09A (154)                | This check is based on the first byte in the key identifier parameter. The key identifier provided is either an internal token, where an external or null token was required; or an external or null token, where an internal token was required. The token provided may be none of these, and, therefore, the parameter is not a key identifier at all. Another cause is specifying a key_type of IMP-PKA for a key in importable form.  
**User action:** Check the type of key identifier required and review what you have provided. Also check that your parameters are in the required sequence.  
**REASONCODES:** ICSF 7F8 (2040) |
| 09B (155)                | The value that the generated_key_identifier parameter specifies is not valid, or it is not consistent with the value that the key_form parameter specifies. |
| 09C (156)                | A keyword is not valid with the specified parameters.  
**REASONCODES:** ICSF 2790 (10128) |
| 09D (157)                | The rule_array parameter contents are incorrect.  
**User action:** Refer to the rule_array parameter described in this publication under the appropriate callable service for the correct value.  
**REASONCODES:** ICSF 7E0 (2016) |
| 0A0 (160)                | The key_type and the key_length are not consistent.  
**User action:** Review the key_type parameter provided and match it with the key_length parameter. |
| A4 (164)                 | Two parameters (perhaps the plaintext and ciphertext areas, or text_in and text_out areas) overlap each other. That is, some part of these two areas occupy the same address in memory. This condition cannot be processed.  
**User action:** Determine which two areas are responsible, and redefine their positions in memory. |
<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0A5 (165)</td>
<td>The contents of a chaining vector passed to a callable service are not valid. If you called the MAC generation callable service, or the MDC generation callable service with a MIDDLE or LAST segmenting rule, the count field has a number that is not valid. If you called the MAC verification callable service, then this will have been a MIDDLE or LAST segmenting rule. <strong>User action:</strong> Check to ensure that the chaining vector is not modified by your program. The chaining vector returned by ICSF should only be used to process one message set, and not intermixed between alternating message sets. This means that if you receive and process two or more independent message streams, each should have its own chaining vector. Similarly, each message stream should have its own key identifier. If you use the same chaining vector and key identifier for alternating message streams, you will <strong>not</strong> get the correct processing performed. <strong>REASONCODES:</strong> ICSF 7F4 (2036)</td>
</tr>
<tr>
<td>0B4 (180)</td>
<td>A null key token was passed in the key identifier parameter. When the key type is TOKEN, a valid token is required. <strong>User action:</strong> Supply a valid token to the key identifier parameter.</td>
</tr>
<tr>
<td>0B5 (181)</td>
<td>This check is based on the first byte in the key identifier parameter. The key identifier provided is either an internal token, where an external or null token was required; or an external or null token, where an internal token was required. The token provided may be none of these, and, therefore, the parameter is not a key identifier at all. Another cause is specifying a key-type of IMP-PKA for a key in importable form. <strong>User action:</strong> Check the type of key identifier required and review what you have provided. Also check that your parameters are in the required sequence. This reason code also corresponds to these ICSF reason codes: 7F8 (2040), 2B24 (11044) and 3E98 (16024). Please see those reason codes for additional information.</td>
</tr>
<tr>
<td>0B7 (183)</td>
<td>A cross-check of the control vector the key type implies has shown that it does not correspond with the control vector present in the supplied internal key identifier. <strong>User action:</strong> Change either the key type or key identifier. <strong>REASONCODES:</strong> ICSF 273C (10044)</td>
</tr>
<tr>
<td>0B8 (184)</td>
<td>An input pointer is null.</td>
</tr>
<tr>
<td>0CC (204)</td>
<td>A memory allocation failed.</td>
</tr>
<tr>
<td>14F (335)</td>
<td>The requested function is not implemented on the coprocessor.</td>
</tr>
<tr>
<td>154 (340)</td>
<td>One of the input control vectors has odd parity.</td>
</tr>
<tr>
<td>157 (343)</td>
<td>Either the data block or the buffer for the block is too small.</td>
</tr>
<tr>
<td>159 (345)</td>
<td>Insufficient storage space exists for the data in the data block buffer.</td>
</tr>
<tr>
<td>15A (346)</td>
<td>The requested command is not valid in the current state of the cryptographic hardware component.</td>
</tr>
<tr>
<td>176 (374)</td>
<td>Less data was supplied than expected or less data exists than was requested. <strong>REASONCODES:</strong> ICSF 7D4 (2004) and ICSF 7E0 (2016)</td>
</tr>
<tr>
<td>181 (385)</td>
<td>The cryptographic hardware component reported that the data passed as part of the command is not valid for that command.</td>
</tr>
<tr>
<td>197 (407)</td>
<td>A PIN block consistency check error occurred. <strong>REASONCODES:</strong> ICSF BC8 (3016)</td>
</tr>
<tr>
<td>Reason Code Hex (Decimal)</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>25D (605)</td>
<td>The number of output bytes is greater than the number that is permitted.</td>
</tr>
<tr>
<td>2BF (703)</td>
<td>A new master key value was found to be one of the weak DES keys.</td>
</tr>
<tr>
<td>2C0 (704)</td>
<td>The new master key would have the same master key verification pattern as the current master key.</td>
</tr>
<tr>
<td>2C1 (705)</td>
<td>The same key-encrypting key was specified for both exporter keys.</td>
</tr>
</tbody>
</table>
| 2C2 (706)                 | While deciphering ciphertext that had been created using a padding technique, it was found that the last byte of the plaintext did not contain a valid count of pad characters. Note that all cryptographic processing has taken place, and the `clear_text` parameter contains the deciphered text.  

**User action:** The `text_length` parameter was not reduced. Therefore, it contains the length of the base message, plus the length of the padding bytes and the count byte. Review how the message was padded prior to being enciphered. The count byte that is not valid was created prior to the message's encipherment. You may need to check whether the ciphertext was not created using a padding scheme. Otherwise, check with the creator of the ciphertext on the method used to create it. You could also look at the plaintext to review the padding scheme used, if any.  

**REASONCODES:** ICSF 7EC (2028) |
| 2C3 (707)                 | The master key registers are not in the state required for the requested function.  

**Contact your ICSF administrator.** |
| 2CA (714)                 | A reserved parameter was not a null pointer or an expected value.  

**REASONCODES:** ICSF 844 (2116) |
| 2CB (715)                 | You supplied a `pad_character` that is not valid for a Transaction Security System compatibility parameter for which ICSF supports only one value; or, you supplied a KEY keyword and a non-zero `master_key_version_number` in the Key Token Build service; or, you supplied a non-zero regeneration data length for a DSS key in the PKA Generate service.  

**User action:** Check that you specified the valid value for the TSS compatibility parameter.  

**REASONCODES:** ICSF 834 (2100) |
| 2CF (719)                 | The RSA-OAEP block did not verify when it decomposed. The block type is incorrect (must be X'03').  

**User action:** Recreate the RSA-OAEP block.  

**REASONCODES:** ICSF 2B38 (11064) |
| 2D0 (720)                 | The RSA-OAEP block did not verify when it decomposed. The random number I is not correct (must be non-zero with the high-order bit equal to zero).  

**User action:** Recreate the RSA-OAEP block.  

**REASONCODES:** ICSF 2B40 (11072) |
| 2D1 (721)                 | The RSA-OAEP block did not verify when it decomposed. The verification code is not correct (must be all zeros).  

**User action:** Recreate the RSA-OAEP block.  

**REASONCODES:** ICSF 2BC3 (11068) |
Table 237. Reason Codes for Return Code 8 (8) (continued)

<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 2F8 (760)                 | The RSA public or private key specified a modulus length that is incorrect for this service.  
**User action:** Re-invoke the service with an RSA key with the proper modulus length.  
**REASONCODES:** ICSF 2B48 (11080) |
| 302 (770)                 | A reserved field in a parameter, probably a key identifier, has a value other than zero.  
**User action:** Key identifiers should not be changed by application programs for other uses. Review any processing you are performing on key identifiers and leave the reserved fields in them at zero.  
This reason code also corresponds to these ICSF reason codes: 7E8 (2024) and 2B00 (11008). Please see those reason codes for additional information.  
**REASONCODES:** ICSF 2B00 (11008) |
| 30F (783)                 | The command is not permitted by the Function Control Vector value.  
**REASONCODES:** ICSF Return code 12, reason code 2B0C (11020) |
| 401 (1025)                | Registered public key or retained private key name already exists. |
| 402 (1026)                | Registered public key or retained private key name does not exist. |
| 405 (1029)                | There is an error in the Environment Identification data. |
| 40B (1035)                | The signature does not match the certificate signature during an RKX call.  
**User Action:** Check that the key used to check the signatures is the correct. |
| 41A (1050)                | A KEK RSA-enciphered at this node (EID) cannot be imported at this same node. |
| 41C (1052)                | Token identifier of the trusted block's header section is in the range 0x20 and 0xFF.  
**User Action:** Check the token identifier of the trusted block. |
| 41D (1053)                | The Active flag in the trusted block's trusted block section 0x14 is not disabled.  
**User Action:** Use the trusted block create callable service to create an inactive/external trusted block. |
| 41E (1054)                | Token identifier of the trusted block's header section is not 0x1E (external).  
**User Action:** Use the trusted block create callable service to create an inactive/external trusted block. |
| 41F (1055)                | The Active flag of the trusted block's trusted block section 0x14 is not enabled.  
**User Action:** Use the trusted block create callable service to create an active/external trusted block. |
| 420 (1056)                | Token identifier of the trusted block's header section is not 0x1F (internal).  
**User Action:** Use the PKA public key import callable service to import the trusted block. |
| 421 (1057)                | Trusted block rule section 0x12 Rule ID does not match input parameter rule ID.  
**User Action:** Verify the trusted block used has the rule section specified. |
| 422 (1058)                | Trusted block contains a value that is too small/too large. |
| 423 (1059)                | A trusted block parameter that must have a value of zero (or a grouping of bits set to zero) is invalid. |
| 424 (1060)                | Trusted block public key section failed consistency checking. |
Table 237. Reason Codes for Return Code 8 (8) (continued)

<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 425 (1061)               | Trusted block contains extraneous sections or subsections (TLVs).  
**User Action:** Check the trusted block for undefined sections of subsections. |
| 426 (1062)               | Trusted block contains missing sections or subsections (TLVs).  
**User Action:** Check the trusted block for required sections and subsections applicable to the callable service invoked. |
| 427 (1063)               | Trusted block contains duplicate sections or subsections (TLVs).  
**User Action:** Check the trusted block's sections and subsections for duplicates. Multiple rule sections are allowed. |
| 428 (1064)               | Trusted block expiration date has expired (as compared to the 4764 clock).  
**User Action:** Validate the expiration date in the trusted block's trusted information section's Activation and Expiration Date TLV Object. |
| 429 (1065)               | Trusted block expiration date is at a date prior to the activation date.  
**User Action:** Validate the expiration date in the trusted block's trusted information section's Activation and Expiration Date TLV Object. |
| 42A (1066)               | Trusted Block Public Key Modulus bit length is not consistent with the byte length. The bit length must be less than or equal to byte length * 8 and greater than (byte length - 1) * 8. |
| 42B (1067)               | Trusted block Public Key Modulus Length in bits exceeds the maximum allowed bit length as defined by the Function Control Vector. |
| 42C (1068)               | One or more trusted block sections or TLV Objects contained data which is invalid (an example would be invalid label data in label section 0x13). |
| 42D (1069)               | Trusted block verification was attempted by a function other than CSNDDSV, CSNDKTC, CSNDKPI, CSNDRKX, or CSNDTBC. |
| 42E (1070)               | Trusted block rule ID contained within a Rule section contains invalid characters. |
| 42F (1071)               | The source key's length or CV does not match what is expected by the rule section in the trusted block that was selected by the rule ID input parameter. |
| 430 (1072)               | The activation data is not valid.  
**User Action:** Validate the activation data in the trusted block's trusted information section's Activation and Expiration Date TLV Object. |
| 431 (1073)               | The source-key label does not match the template in the export key DES token parameters TLV object of the selected trusted block rule section. |
| 432 (1074)               | The control-vector value specified in the common export key parameters TLV object in the selected rule section of the trusted block contains a control vector that is not valid. |
| 433 (1075)               | The source-key label template in the export key DES token parameters TLV object in the selected rule section of the trusted block contains a label template that is not valid. |
| 7D1 (2001)               | TKE: DH generator is greater than the modulus. |
| 7D2 (2002)               | TKE: DH registers are not in a valid state for the requested operation. |
### Reason Codes for Return Code 8 (8) (continued)

<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 7D4 (2004)                | A length parameter has an incorrect value. The value in the length parameter could have been zero (when a positive value was required) or a negative value. If the supplied value was positive, it could have been larger than your installation's defined maximum, or for MDC generation with no padding, it could have been less than 16 or not an even multiple of 8.  
User action: Check the length you specified. If necessary, check your installation's maximum length with your ICSF administrator. Correct the error.  
REASONCODES: TSS 019 (025) |
| 7D8 (2008)                | Two parameters (perhaps the plaintext and ciphertext areas, or text_in and text_out areas) overlap each other. That is, some part of these two areas occupy the same address in memory. This condition cannot be processed.  
User action: Determine which two areas are responsible, and redefine their positions in memory.  
REASONCODES: TSS 0A4 (164) |
| 7D9 (2009)                | TKE: ACI can not load both loads and profiles in one call. |
| 7DA (2010)                | TKE: ACI can only load one role or one profile at a time. |
| 7DC (2012)                | The rule_array_count parameter contains a number that is not valid.  
User action: Refer to the rule_array_count parameter described in this publication under the appropriate callable service for the correct value.  
REASONCODES: TSS 023 (035) |
| 7DD (2013)                | TKE: Length of hash pattern for keypart is not valid for DH transport key algorithm specified. |
| 7DE (2014)                | TKE: PCB buffer is empty. |
| 7E0 (2016)                | The rule_array parameter contents are incorrect. One or more of the rules specified are not valid for this service OR some of the rules specified together may not be combined.  
User action: Refer to the rule_array parameter described in this publication under the appropriate callable service for the correct value. |
| 7E2 (2018)                | The form parameter specified in the random number generate callable service should be ODD, EVEN, or RANDOM. One of these values was not supplied.  
User action: Change your parameter to use one of the required values for the form parameter.  
REASONCODES: TSS 021 (033) |
| 7E3 (2019)                | TKE: Signature in request CPRB did not verify. |
| 7E4 (2020)                | TKE: TSN in request CPRB is not valid. |
| 7E8 (2024)                | A reserved field in a parameter, probably a key identifier, has a value other than zero.  
User action: Key identifiers should not be changed by application programs for other uses. Review any processing you are performing on key identifiers and leave the reserved fields in them at zero. |
<p>| 7EB (2027)                | TKE: DH transport key hash pattern doesn't match. |</p>
<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7EC (2028)</td>
<td>While deciphering ciphertext that had been created using a padding technique, it was found that the last byte of the plaintext did not contain a valid count of pad characters. Note that all cryptographic processing has taken place, and the <code>clear_text</code> parameter contains the deciphered text. <strong>User action:</strong> The <code>text_length</code> parameter was not reduced. Therefore, it contains the length of the base message, plus the length of the padding bytes and the count byte. Review how the message was padded prior to it being enciphered. The count byte that is not valid was created prior to the message's encipherment. You may need to check whether the ciphertext was not created using a padding scheme. Otherwise, check with the creator of the ciphertext on the method used to create it. You could also look at the plaintext to review the padding scheme used, if any. <strong>REASONCODES:</strong> TSS 2C2 (706)</td>
</tr>
<tr>
<td>7ED (2029)</td>
<td>TKE: Request data block hash does not match hash in CPRB.</td>
</tr>
<tr>
<td>7EE (2030)</td>
<td>TKE: DH supplied hash length is not correct.</td>
</tr>
<tr>
<td>7EF (2031)</td>
<td>Reply data block too large.</td>
</tr>
<tr>
<td>7F0 (2032)</td>
<td>The <code>key_form</code>, <code>key_type_1</code>, and <code>key_type_2</code> parameters for the key generate callable service form a combination, a three-element string. This combination is checked against all valid combinations. Your combination was not found among this list. <strong>User action:</strong> Check the allowable combinations described for each parameter in Key Generate callable service and correct the appropriate parameter(s).</td>
</tr>
<tr>
<td>7F1 (2033)</td>
<td>TKE: Change type does not match PCB change type.</td>
</tr>
<tr>
<td>7F4 (2036)</td>
<td>The contents of a chaining vector passed to a callable service are not valid. If you called the MAC generation callable service, or the MDC generation callable service with a MIDDLE or LAST segmenting rule, the count field has a number that is not valid. If you called the MAC verification callable service, then this will have been a MIDDLE or LAST segmenting rule. <strong>User action:</strong> Check to ensure that the chaining vector is not modified by your program. The chaining vector returned by ICSF should only be used to process one message set, and not intermixed between alternating message sets. This means that if you receive and process two or more independent message streams, each should have its own chaining vector. Similarly, each message stream should have its own key identifier. If you use the same chaining vector and key identifier for alternating message streams, you will <strong>not</strong> get the correct processing performed. <strong>REASONCODES:</strong> TSS 0A5 (165)</td>
</tr>
<tr>
<td>7F6 (2038)</td>
<td>No RSA private key information was provided in the supplied token. <strong>User action:</strong> Check that the token supplied was of the correct type for the service.</td>
</tr>
<tr>
<td>7F8 (2040)</td>
<td>This check is based on the first byte in the key identifier parameter. The key identifier provided is either an internal token, where an external or null token was required; or an external or null token, where an internal token was required. The token provided may be none of these, and, therefore, the parameter is not a key identifier at all. Another cause is specifying a <code>key_type</code> of IMP-PKA for a key in importable form. <strong>User action:</strong> Check the type of key identifier required and review what you have provided. Also check that your parameters are in the required sequence. <strong>REASONCODES:</strong> TSS 03F (063) and TSS 09A (154)</td>
</tr>
<tr>
<td>Reason Code Hex (Decimal)</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>7FC (2044)</td>
<td>The caller must be in task mode, not SRB mode.</td>
</tr>
<tr>
<td>800 (2048)</td>
<td>The <code>key_form</code> is not valid for the <code>key_type</code></td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Review the <code>key_form</code> and <code>key_type</code> parameters. For a <code>key_type</code> of IMP-PKA, the secure key import callable service supports only a <code>key_form</code> of OP.</td>
</tr>
<tr>
<td>802 (2050)</td>
<td>A UKPT keyword was specified, but there is an error in the <code>PIN_profile</code> key serial number.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Correct the <code>PIN_profile</code> key serial number.</td>
</tr>
<tr>
<td>803 (2051)</td>
<td>Invalid message length in OAEP-decoded information.</td>
</tr>
<tr>
<td>804 (2052)</td>
<td>A single-length key, passed to the secure key import callable service in the <code>clear_key</code> parameter, must be padded on the right with binary zeros. The fact that it is a single-length key is identified by the <code>key_form</code> parameter, which identifies the key as being DATA, MACGEN, MACVER, and so on.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> If you are providing a single-length key, pad the parameter on the right with zeros. Alternatively, if you meant to pass a double-length key, correct the <code>key_form</code> parameter to a valid double-length key type.</td>
</tr>
<tr>
<td>805 (2053)</td>
<td>No message found in OAEP-decoded information.</td>
</tr>
<tr>
<td>806 (2054)</td>
<td>Invalid RSA enciphered key cryptogram; OAEP optional encoding parameters failed validation.</td>
</tr>
<tr>
<td>807 (2055)</td>
<td>The RSA public key is too small to encrypt the DES key.</td>
</tr>
<tr>
<td>808 (2056)</td>
<td>The <code>key_form</code> parameter is neither IM nor OP. Most constants, these included, can be supplied in lower or uppercase. Note that this parameter is 4 bytes long, so the value IM or OP is not valid. They must be padded on the right with blanks.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Review the value provided and change it to IM or OP, as required.</td>
</tr>
<tr>
<td></td>
<td><strong>REASONCODES:</strong> TSS 029 (041)</td>
</tr>
<tr>
<td>80C (2060)</td>
<td>The <code>key_length</code> parameter passed to the key generate callable service holds a value that is not valid.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Review the value provided and change it as appropriate.</td>
</tr>
<tr>
<td></td>
<td><strong>REASONCODES:</strong> TSS 02B (043)</td>
</tr>
<tr>
<td>810 (2064)</td>
<td>The <code>key_type</code> and the <code>key_length</code> are not consistent.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Review the <code>key_type</code> parameter provided and match it with the <code>key_length</code> parameter.</td>
</tr>
<tr>
<td></td>
<td><strong>REASONCODES:</strong> TSS 0A0 (160)</td>
</tr>
<tr>
<td>813 (2067)</td>
<td>TKE: A key part register is in an invalid state. This includes the case where an attempt is made to load a FIRST key part, but a register already contains a key or key part with the same key name.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Supply a different label name for the key part register or clear the existing key part register with the same label name.</td>
</tr>
</tbody>
</table>
### Table 237. Reason Codes for Return Code 8 (8) (continued)

<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 814 (2068)                | You supplied a key identifier or token to the key generate, key import, multiple secure key import, key export, or key record write callable service. This key identifier holds an importer or exporter key, and the NOCV bit is on in the token. Only programs running in supervisor state or in a system key (key 0–7) may provide a key identifier with this bit set on. Your program was not running in supervisor state or a system key.  
**User action:** Either use a different key identifier, or else run in supervisor state or a system key. |
| 815 (2069)                | TKE: The control vector in the key part register does not match the control vector in the key structure. |
| 816 (2070)                | TKE: All key part registers are already in use.  
**User action:** Either free existing key part registers by loading keys from ICSF or clearing selected key part registers from TKE or select another PCIXCC, CEX2C, or CEX3C for loading the key part register. |
| 817 (2071)                | TKE: The key part hash pattern supplied does not match the hash pattern of the key part currently in the register. |
| 818 (2072)                | A request was made to the key generate callable service to generate double-length keys of SINGLE effective length, in the IMEX form. This request is valid only if the KEK_key_identifier parameter identifies a NOCV importer, and the caller (wrongly) supplies a CV importer. The combination of IMEX for the key_form parameter and a CV importer key-encrypting key can only be used for single-length keys.  
**User action:** Either use a key identifier that holds (or identifies) a NOCV importer, or specify a single-length key in the key_type parameter. |
| 81B (2075)                | TKE: The length of the key part received is different from the length of the accumulated value already in the key part register. |
| 81C (2076)                | A request was made to the key import callable service to import a single-length key. However, the right half of the key in the source_key_identifier parameter is not zeros. Therefore, it appears to identify the right half of a double-length key. This combination is not valid. This error does not occur if you are using the word TOKEN in the key_type parameter.  
**User action:** Check that you specified the value in the key_type parameter correctly, and that you are using the correct or corresponding source_key_identifier parameter. |
| 81D (2077)                | TKE: An error occurred storing or retrieving the key part register data.  
**User action:** Verify that the selected PCIXCC, CEX2C, or CEX3C is functioning correctly and retry the operation. |
| 81F (2079)                | An encrypted key token is not supported in the service. |
| 824 (2084)                | The key token is not valid for the CSNBTCX service. If the source_key_identifier is an external token, then the KEK_key_identifier cannot be marked as CDMF.  
**User action:** Correct the appropriate key identifiers. |
| 828 (2088)                | The origin_identifier or destination_identifier you supplied is not a valid ASCII hexadecimal string.  
**User action:** Check that you specified a valid ASCII string for the origin_identifier or destination_identifier parameter. |
<p>| 829 (2089)                | The algorithm does not match the algorithm of the key identifier. |</p>
<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 82C (2092)               | The `source_key_identifier` or `inbound_key_identifier` you supplied in an ANSI X9.17 service is not a valid ASCII hexadecimal string.  
**User action:** Check that you specified a valid ASCII string for the `source_key_identifier` or `inbound_key_identifier` parameter.  
**REASONCODES:** TSS 079 (121) |
| 82D (2093)               | Key identifiers contain a version number. The version number in a supplied key identifier (internal or external) is inconsistent with one or more fields in the key identifier, making the key identifier unusable.  
**User action:** Use a token containing the required version number. |
| 82F (2095)               | The value in the `key_form` parameter is incompatible with the value in the `key_type` parameter.  
**User action:** Ensure compatibility of the selected parameters. |
| 830 (2096)               | The `outbound KEK_count` or `inbound KEK_count` you supplied is not a valid ASCII hexadecimal string.  
**User action:** Check that you specified a valid ASCII hexadecimal string for the `outbound KEK_count` or `inbound KEK_count` parameter.  
**REASONCODES:** TSS 07A (122) |
| 831 (2097)               | The value in the `key_identifier_length` parameter is incompatible with the value in the `key_type` parameter.  
**User action:** Ensure compatibility of the selected parameters. |
| 832 (2098)               | Either a key bit length that was not valid was found in an AES key token (length not 128, 192, or 256 bits) or a version X'01' DES token had a token-marks field that was not valid. |
| 833 (2099)               | An encrypted key length in an AES key token was not valid when an encrypted key is present in the token. |
| 834 (2100)               | You supplied a `pad_character` that is not valid for a Transaction Security System compatibility parameter for which ICSF supports only one value; or, you supplied a KEY keyword and a non-zero `master_key_version_number` in the Key Token Build service; or, you supplied a non-zero regeneration data length for a DSS key in the PKA Generate service.  
**User action:** Check that you specified the valid value for the TSS compatibility parameter.  
**REASONCODES:** TSS 2CB (715) |
| 838 (2104)               | An input character is not in the code table.  
**User action:** Correct the code table or the source text.  
**REASONCODES:** TSS 02D (045) |
| 83C (2108)               | An unused field must be binary zeros, and an unused key identifier field generally must be zeros.  
**User action:** Correct the parameter list.  
**REASONCODES:** TSS 02F (047) |
### Table 237. Reason Codes for Return Code 8 (8) (continued)

<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 840 (2112)               | The length is incorrect for the key type.  
**User action**: Check the key length parameter. DATA keys may have a length of 8, 16, or 24. DATAxLAT and MAC keys must have a length of 8. All other keys should have a length of 16. Also check that the parameters are in the required sequence. |
| 844 (2116)               | Parameter contents or a parameter value is not correct.  
**User action**: Specify a valid value for the parameter. |
| BB9 (3001)               | SET block decompose service was called with an encrypted OAEP block with a block contents identifier that indicates a PIN block is present. No PIN encrypting key was supplied to process the PIN block. The block contents identifier is returned in the block_contents_identifier parameter.  
**User action**: Supply a PIN encrypting key and resubmit the job. |
| BBB (3003)               | An output parameter is too short to hold the output of the request. The length parameter for the output parameter has been updated with the required length for the request.  
**User action**: Update the size of the output parameter and length specified in the length field and resubmit the request. |
| BBC (3004)               | A request was made to the Clear PIN generate or Encrypted PIN verify callable service, and the PIN_length parameter has a value outside the valid range. The valid range is from 4 to 16, inclusive.  
**User action**: Correct the value in the PIN_length parameter to be within the valid range from 4 to 16.  
**REASONCODES**: TSS 064 (100) |
| BBE (3006)               | The UDX verb in the PCICC, PCIXCC, CEX2C, or CEX3C is not authorized to be executed. |
| BC0 (3008)               | A request was made to the Clear PIN generate callable service, and the PIN_check_length parameter has a value outside the valid range. The valid range is from 4 to 16, inclusive.  
**User action**: Correct the value in the PIN_check_length parameter to be within the valid range from 4 to 16.  
**REASONCODES**: TSS 065 (101) |
| BC1 (3009)               | For PKCS #11 attribute processing, an attribute has been specified in the template that is not consistent with another attribute of the object being created or updated.  
**User action**: Correct the template for the object. |
| BC3 (3011)               | The CRT value (p, q, Dp, Dq or U) is longer than the length allowed by the parameter block for clear key processing on an accelerator. A modulus whose length is less than or equal to 1024 bits is 64 bytes in length. A modulus whose length is greater than 1024 bits but less than or equal to 2048 bits is 128 bytes in length.  
**User action**: Reconfigure CEX2A as a CEX2C or CEX3A as a CEX3C to make use of the key (if the CRT value is not in error and there is no CEX2C or CEX3C installed).  
**REASONCODES**: TSS 065 (101) |
<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
</table>
| BC4 (3012)               | A request was made to the Clear PIN generate callable service to generate a VISA-PVV PIN, and the `trans_sec_parm` field has a value outside the valid range. The field being checked in the `trans_sec_parm` is the key index, in the 12th byte. This `trans_sec_parm` field is part of the `data_array` parameter.  
**User action:** Correct the value in the key index, held within the `trans_sec_parm` field in the `data_array` parameter, to hold a number from the valid range.  
**REASONCODES:** TSS 069 (105) |
| BC5 (3013)               | The AES clear key value LRC in the token failed validation.  
**User action:** Correct the AES clear key value.  
**REASONCODES:** TSS 06A (106) |
| BC8 (3016)               | A request was made to the Encrypted PIN Translate or the Encrypted PIN verify callable service, and the PIN block value or PADDIGIT value in the `input_PIN_profile` or `output_PIN_profile` parameter has a value that is not valid.  
**User action:** Correct the PIN block value.  
**REASONCODES:** TSS 06A (106) |
| BCB (3019)               | The call to insert or delete a z/OS PKCS #11 token object failed because the token was not found in the TKDS data space or a request to delete a PKCS #11 session object failed because the token was not found in the session data space. |
| BCC (3020)               | For a PKCS #11 callable service, the PKCS #11 object specified is the incorrect class for the request.  
**User action:** Specify the correct class of object for the service. |
| BCD (3021)               | The call to add a z/OS PKCS #11 token failed because the token already exists in the TKDS data space or a request to add a z/OS PKCS #11 token object failed because an object with the same handle already exists. |
| BCE (3022)               | The call to add or update a z/OS PKCS #11 tokens object failed because the supplied attributes are too large to be stored in the TKDS. |
| BD0 (3024)               | A request was made to the Encrypted PIN Translate callable service and the format control value in the `input_PIN_profile` or `output_PIN_profile` parameter has a value that is not valid. The valid values are NONE or PBVC.  
**User action:** Correct the format control value to either NONE or PBVC.  
**REASONCODES:** TSS 06B (107) |
| BD1 (3025)               | The call to create a list of z/OS PKCS #11 tokens, a list of objects of a z/OS PKCS #11 token, the information for a z/OS PKCS #11 token or the attributes of a PKCS #11 object failed because the length of the output field was insufficient to hold the data. The length field has been updated with the length of a single list or entry, token information or object attributes. |
| BD2 (3026)               | The z/OS PKCS #11 token or object handle syntax is invalid. |
| BD3 (3027)               | The call to read or update a z/OS PKCS #11 token or token object failed because the token or object was not found in the TKDS data space, or if the call to read or update a PKCS #11 session object failed because the object was not found. |
### Table 237. Reason Codes for Return Code 8 (8) (continued)

<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BD4 (3028)</td>
<td>A request was made to the Clear PIN generate callable service. The clear_PIN supplied as part of the data_array parameter for an GBP-PINO request begins with a zero (0). This value is not valid.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Correct the clear_PIN value.</td>
</tr>
<tr>
<td></td>
<td><strong>REASONCODES:</strong> TSS 074 (116)</td>
</tr>
<tr>
<td>BD5 (3029)</td>
<td>For PKCS #11 attribute processing, an invalid attribute was specified in the template. The attribute is neither a PKCS #11 or vendor-specified attribute supported by this implementation of PKCS #11.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Correct the template by removing the invalid attribute or changing the attribute to a valid attribute.</td>
</tr>
<tr>
<td>BD6 (3030)</td>
<td>An invalid value was specified for a particular PKCS #11 attribute in a template when creating or updating an object.</td>
</tr>
<tr>
<td>BD7 (3031)</td>
<td>The certificate specified in creating a PKCS #11 certificate object was not properly encoded.</td>
</tr>
<tr>
<td>BD9 (3033)</td>
<td>The attribute template for creating or updating a PKCS #11 object was incomplete. Required attributes for the object class were not specified in the template.</td>
</tr>
<tr>
<td>BDA (3034)</td>
<td>The call to modify PKCS #11 object attributes failed because the CKA_MODIFIABLE attribute was set to false when the object was recreated.</td>
</tr>
<tr>
<td>BDB (3035)</td>
<td>For PKCS #11 attribute processing, an attribute was specified in the template which can not be set or updated by the application. See z/OS Cryptographic Services ICSF Writing PKCS #11 Applications for a definition of attributes that can be set or updated by the application.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Remove the offending attribute from the template.</td>
</tr>
<tr>
<td>BDC (3036)</td>
<td>A request was made to the Encrypted PIN Translate callable service. The sequence_number parameter was required, but was not the integer value 99999.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Specify the integer value 99999.</td>
</tr>
<tr>
<td></td>
<td><strong>REASONCODES:</strong> TSS 06F (111)</td>
</tr>
<tr>
<td>BDE (3038)</td>
<td>For a PKCS #11 callable service, the attributes of the PKCS #11 object specified do not permit the requested function.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Specify an object that permits the requested function.</td>
</tr>
<tr>
<td>BDF (3039)</td>
<td>For a PKCS #11 callable service, where a PKCS #11 key object is required, the specified object is not of the correct key type for the requested function.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Specify an object that is the correct class of key.</td>
</tr>
<tr>
<td>BE0 (3040)</td>
<td>The PAN, expiration date, service code, decimalization table data, validation data, or pad data is not numeric (XF0' through X'F9'). The parameter must be character representations of numerics or hexadecimal data.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Review the numeric parameters or fields required in the service that you called and change to the format and values required.</td>
</tr>
<tr>
<td></td>
<td><strong>REASONCODES:</strong> TSS 028 (040), TSS 02A (042), TSS 066 (102), TSS 067 (103), TSS 068 (104), TSS 069 (105), TSS 06E (110)</td>
</tr>
<tr>
<td>BE1 (3041)</td>
<td>PKCS #11 wrap key callable service failed because the wrapping key object is not of the correct class to wrap the key specified to be wrapped.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Specify a wrapping key object of the correct class to wrap the key object.</td>
</tr>
<tr>
<td>Reason Code Hex (Decimal)</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| BE3 (3043)               | PKCS #11 wrap key callable service failed because the key object to be wrapped does not exist or the key class does not match the wrapping mechanism.  
**User action:** Specify an existing key object that is correct for the wrapping mechanism. |
| BE4 (3044)               | A PKCS #11 session data space is full. The request to create or update an object failed and the object was not created or updated.  
**User action:** Delete unused session objects and cryptographic state objects from incomplete chained operations to create space for new or updated objects. |
| BE5 (3045)               | PKCS #11 wrap key callable service failed because the key object to be wrapped has CKA_EXTRACTABLE set to false.  
**User action:** Specify another key object that can be extracted. |
| BE7 (3047)               | A clear key was provided when a secure key was required.  
**User action:** Correct the appropriate key identifier. |
| BEA (3050)               | A caller is attempting to overwrite one token type with another (for example, AES over DES). |
| BEC (3052)               | A clear key token was supplied to a service where a secure token is required. |
| BED (3053)               | A service was called with no parameter list, but a parameter list was expected.  
**User action:** Call the service with a parameter list. |
| BF5 (3061)               | The provided asymmetric key identifier can not be used for the requested function. PKA Key Management Extensions have been enabled by a CSF.PKAEXTNS.ENABLE profile in the X FACILIT class. A CSFKEYS profile covering the key includes an ICSF segment, and the ASYMUSAGE field of that segment restricts the key from being used for the specified function.  
An SMF type 82 subtype 27 record is logged in the SMF database. |
| BF6 (3062)               | The provided symmetric key identifier can not be exported using the provided asymmetric key identifier. PKA Key Management Extensions have been enabled by a CSF.PKAEXTNS.ENABLE profile in the X FACILIT class. A CSFKEYS or XCSFKEY profile covering the symmetric key includes an ICSF segment and the SYMEXPORTABLE field of that segment places restrictions on how the key can be exported. The SYMEXPORTABLE field either specifies BYNONE, or else specifies BYLIST but the provided asymmetric key identifier is not one of those permitted to export the symmetric key (as identified by the SYMEXPORTCERTS or SYMEXPORTKEYS fields).  
An SMF type 82 subtype 27 record is logged to the SMF database. |
| BF7 (3063)               | ICSF key store policy checking is active. The request failed the ICSF token policy check because the caller is not authorized to the label for the token in the key data set (CKDS or PKDS). The request is not allowed to continue because the token check policy is in FAIL mode.  
SMF type 82 subtype 25 records are logged in the SMF dataset. An SMF type 80 with event code qualifier of ACCESS is logged.  
The policy is defined by the CSF.CKDS.TOKEN CHECK.LABEL.FAIL resource or the CSF.PKDS.TOKEN CHECK.LABEL.FAIL resource in the X FACILIT class. |
<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BF8 (3064)</td>
<td>ICSF key store policy checking is active. The specified token does not exist in the key data set (CKDS or PKDS as appropriate). The CSF-CKDS-DEFAULT OR CSF-PKDS-DEFAULT resource in the CSFKEYS class is either not defined or the caller is not authorized to the CSF-CKDS-DEFAULT OR CSF-PKDS-DEFAULT resource. The resource is not in WARNING mode, so the request is not allowed to continue. An SMF type 80 record with event qualifier ACCESS is logged indicating the request failed. The policy is defined by the CSF.CKDS.TOKEN.CHECK.DEFAULT.LABEL or the CSF.PKDS.TOKEN.CHECK.DEFAULT.LABEL resource in the XFACILIT class.</td>
</tr>
<tr>
<td>BF9 (3065)</td>
<td>ICSF token policy checking is active. The caller is requesting to add a token to the key data set (CKDS or PKDS as appropriate) that already exists within the key data set. The request fails. The policy is defined by the CSF.CKDS.TOKEN.NODUPLICATES resource or the CSF.PKDS.TOKEN.NODUPLICATES resource in the XFACILIT class.</td>
</tr>
<tr>
<td>BFB (3067)</td>
<td>The provided symmetric key label refers to an encrypted CCA key token, and the CSFKEYS profile covering it does not allow its use in high performance encrypted key operations. <strong>User action:</strong> Contact your ICSF or RACF administrator if you need to use this key in calls to Symmetric Key Encrypt (CSFNBSYE) or Symmetric Key Decrypt (CSNBSYD). Otherwise, use Encipher (CSNBENC) or Decipher (CSNBDEC) instead.</td>
</tr>
<tr>
<td>BFD (3069)</td>
<td>A cryptographic operation that requires FIPS 140-2 compliance is being requested. Either ICSF has not been configured to run in FIPS mode or the system environment does not support it. The request is not processed. <strong>User action:</strong> Contact your ICSF administrator to request that ICSF be configured for either FIPS standard mode or FIPS compatibility mode.</td>
</tr>
<tr>
<td>BFE (3070)</td>
<td>A cryptographic operation that requires FIPS 140-2 compliance is being requested. The desired algorithm, mode, or key size is not approved for FIPS 140-2. The request is not processed. <strong>User action:</strong> Repeat the request using an algorithm, mode, and/or key size approved for FIPS 140-2. Refer to <a href="https://www.ibm.com/support/knowledgecenter/SS7Q3D_9.1.0/com.ibm.zos.v9r1%E8%B5%84%E6%96%99/fips140-2.htm">z/OS Cryptographic Services ICSF Writing PKCS #11 Applications</a> for this list of approved algorithms, modes, and key sizes.</td>
</tr>
</tbody>
</table>
| BFF (3071)              | An application using a z/OS PKCS #11 token that is marked ‘Write Protected’ is attempting to do one of the following:  
  - Store a persistent object in the token.  
  - Delete the token.  
  - Reinitialize the token.  
  The request is not processed. **User action:** Use a z/OS PKCS #11 token that is not marked ‘Read Only’. |
<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
</table>
| FA0 (4000)               | The encipher and decipher callable services sometime require text (plaintext or ciphertext) to have a length that is an exact multiple of 8 bytes. Padding schemes always create ciphertext with a length that is an exact multiple of 8. If you want to decipher ciphertext that was produced by a padding scheme, and the text length is not an exact multiple of 8, then an error has occurred. The CBC mode of enciphering requires a text length that is an exact multiple of 8.

The ciphertext translate callable service cannot process ciphertext whose length is not an exact multiple of 8.

User action: Review the requirements of the service you are using. Either adjust the text you are processing or use another process rule. |

<table>
<thead>
<tr>
<th>REASONCODES: TSS 033 (051)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1388 (5000)</td>
</tr>
<tr>
<td>User action: Correct the target cryptographic module parameter and resubmit.</td>
</tr>
<tr>
<td>138C (5004)</td>
</tr>
<tr>
<td>User action: Correct the request and resubmit it.</td>
</tr>
<tr>
<td>1390 (5008)</td>
</tr>
<tr>
<td>User action: Message length of request must be nonzero, a multiple of eight, and less than the system maximum. Correct the request and resubmit it.</td>
</tr>
<tr>
<td>2710 (10000)</td>
</tr>
<tr>
<td>User action: The contents of the token have been altered because it was created by ICSF or TSS. Review your program to see how this could have been caused.</td>
</tr>
<tr>
<td>REASONCODES: TSS 00C (012) and 11D (029)</td>
</tr>
<tr>
<td>2714 (10004)</td>
</tr>
<tr>
<td>User action: Re-import the key from its importable form (if you have it in this form), or repeat the process you used to create the operational key form. If you cannot do one of these, you cannot repeat any previous cryptographic process that you performed with this token.</td>
</tr>
<tr>
<td>REASONCODES: TSS 030 (048)</td>
</tr>
<tr>
<td>271C (10012)</td>
</tr>
<tr>
<td>User action: Check with your administrator if you believe that this key should be in the in-storage CKDS or the PKDS. The administrator may be able to bring it into storage. If this key cannot be in storage, use a different label.</td>
</tr>
<tr>
<td>REASONCODES: TSS 01E (030)</td>
</tr>
<tr>
<td>Reason Code Hex (Decimal)</td>
</tr>
<tr>
<td>--------------------------</td>
</tr>
</tbody>
</table>
| 2720 (10016)             | You specified a value for a key_type parameter that is not an ICSF-defined name.  
**User action:** Review the ICSF key types and use the appropriate one.  
**REASONCODES:** TSS 03D (061) |
| 2724 (10020)             | You specified the word TOKEN for a key_type parameter, but the corresponding key identifier, which implies the key type to use, has a value that is not valid in the control vector field. Therefore, a valid key type cannot be determined.  
**User action:** Review the value that you stored in the corresponding key identifier. Check that the value for key_type is obtained from the appropriate key_identifier parameter.  
**REASONCODES:** TSS 027 (039) |
| 272C (10028)             | Either the left half of the control vector in a key identifier (internal or external) equates to a key type that is not valid for the service you are using, or the value is not that of any ICSF control vector. For example, an exporter key-encrypting key is not valid in the key import callable service.  
**User action:** Determine which key identifier is in error and use the key identifier that is required by the service.  
**REASONCODES:** TSS 027 (039) |
| 2730 (10032)             | Either the right half of the control vector in a key identifier (internal or external) equates to a key type that is not valid for the service you are using, or the value is not that of any ICSF control vector. For example, an exporter key-encrypting key is not valid in the key import callable service.  
**User action:** Determine which key identifier is in error and use the key identifier that is required by the service.  
**REASONCODES:** TSS 027 (039) |
| 2734 (10036)             | Either the complete control vector (CV) in a key identifier (internal or external) equates to a key type that is not valid for the service you are using, or the value is not that of any ICSF control vector.  
The difference between this and reason codes 10028 and 10032 is that each half of the control vector is valid, but as a combination, the whole is not valid. For example, the left half of the control vector may be the importer key-encrypting key and the right half may be the input PIN-encrypting (IPINENC) key.  
**User action:** Determine which key identifier is in error and use the key identifier that is required by the service.  
**REASONCODES:** TSS 027 (039) |
| 2738 (10040)             | Key identifiers contain a version number. The version number in a supplied key identifier (internal or external) is inconsistent with one or more fields in the key identifier, making the key identifier unusable.  
**User action:** Use a token containing the required version number.  
**REASONCODES:** TSS 031 (049) |
| 273C (10044)             | A cross-check of the control vector the key type implies has shown that it does not correspond with the control vector present in the supplied internal key identifier.  
**User action:** Change either the key type or key identifier.  
**REASONCODES:** TSS 0B7 (183) |
Table 237. Reason Codes for Return Code 8 (8) (continued)

<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 2740 (10048)              | The `key_type` parameter does not contain one of the valid types for the service or the keyword TOKEN.  
**User action:** Check the supplied parameter with the ICSF key types. If you supplied the keyword TOKEN, check that you have padded it on the right with blanks.  
**REASONCODES:** TSS 03D (061) |
| 2744 (10052)              | A null key identifier was supplied and the `key_type` parameter contained the word TOKEN. This combination of parameters is not valid.  
**User action:** Use either a null key identifier or the word TOKEN, not both.  
**REASONCODES:** TSS 027 (039) |
| 2748 (10056)              | You called the key import callable service. The importer key-encrypting key is a NOCV importer and you specified TOKEN for the `key_type` parameter. This combination is not valid.  
**User action:** Specify a value in the `key_type` parameter for the operational key form.  
**REASONCODES:** TSS 03D (061) |
| 274C (10060)              | You called the key export callable service. A label was supplied in the `key_identifier` parameter for the key to be exported and the `key_type` was TOKEN. This combination is not valid because the service needs a key type in order to retrieve a key from the CKDS.  
**User action:** Specify the type of key to be exported in the `key_type` parameter.  
**REASONCODES:** TSS 03D (061) |
| 2754 (10068)              | A flag in a key identifier indicates the master key verification pattern (MKVP) is not present in an internal key token. This setting is not valid.  
**User action:** Use a token containing the required flag values.  
**REASONCODES:** TSS 02F (047) |
| 2758 (10072)              | A flag in a key identifier indicates the encrypted key is not present in an external token. This setting is not valid.  
**User action:** Use a token containing the required flag values.  
**REASONCODES:** TSS 02F (047) |
| 275C (10076)              | A flag in a key identifier indicates the control vector is not present. This setting is not valid.  
**User action:** Use a token containing the required flag values.  
**REASONCODES:** TSS 02F (047) |
| 2760 (10080)              | An ICSF private flag in a key identifier has been set to a value that is not valid.  
**User action:** Use a token containing the required flag values. Do not modify ICSF or the reserved flags for your own use.  
**REASONCODES:** TSS 02F (047) |
| 2768 (10088)              | If you supplied a label in the `key_identifier` parameter, a record with the supplied label was found in the CKDS, but the key type (CV) is not valid for the service. If you supplied an internal key token for the `key_identifier` parameter, it contained a key type that is not valid.  
**User action:** Check with your ICSF administrator if you believe that this key should be in the in-storage CKDS. The administrator may be able to bring it into storage. If this key cannot be in storage, use a different label.  
**REASONCODES:** TSS 027 (039) |
<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 276C (10092)             | You supplied a source key that does not have odd parity and specified ENFORCE as the parity rule on the rule_array parameter for either the ANSI X9.17 key export, ANSI X9.17 key import, or ANSI X9.17 key translate callable service.  
**User action:** Either supply an ODD parity key or change the rule_array parameter to specify a parity rule of IGNORE. |
| 2770 (10096)             | The transport key you specified is a single-length key, which cannot be used to encrypt a double-length AKEK or (*KK).  
**User action:** Use a double-length AKEK for the transport key. |
| 2774 (10100)             | You specified a transport key that cannot be notarized and specified the keyword NOTARIZE in the rule_array parameter. The transport key may have already been partially notarized.  
**User action:** Use a transport key that allows notarization or change the rule_array parameter keyword to CPLT-NOT. |
| 2778 (10104)             | The AKEK you specified is either partially notarized or is a partial AKEK, which is not valid for this service.  
**User action:** Use a correct AKEK that is not partially notarized. A partially notarized key can be used as a transport key if you specify CPLT-NOT in the rule_array parameter. |
| 277C (10108)             | You did not supply a partial AKEK for the key_identifier parameter of the key part import service.  
**User action:** Correct the key_id parameter. |
| 2780 (10112)             | The transport key you specified has not been partially notarized and you have specified CPLT-NOT for the rule_array parameter.  
**User action:** Use a transport key that has been partially notarized or change the rule_array parameter. |
| 2784 (10116)             | You attempted to export an AKEK with a CCA key export service, which is not supported.  
**User action:** Use the ANSI X9.17 key export callable service (CSNAKEX). |
| 2788 (10120)             | The internal key token you supplied, or the key token that was retrieved by the label you supplied, contains a flag setting or data encryption algorithm bit that is not valid for this service.  
**User action:** Ensure that you supply a key token, or label, for a non-ANSI key type. |
| 278C (10124)             | The key identifier you supplied cannot be exported because there is a prohibit-export restriction on the key.  
**User action:** Use the correct key for the service.  
**REASONCODES:** TSS 027 (039) |
| 2790 (10128)             | The keyword you supplied in the rule_array parameter is not consistent or not valid with another parameter you specified. For example, the keyword SINGLE is not valid with the key type of EXPORTER in the key token build callable service.  
**User action:** Correct either the rule_array parameter or the other parameter.  
**REASONCODES:** TSS 09C (156) |
<p>| 2791 (10129)             | S390 KEKs with NOCV (flagged as such by the MASK_NOCV bit in the flags field of the token), are not permitted in the RKX service. |</p>
<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 2AF8 (11000)             | The value specified for length parameter for a key token, key, or text field is not valid.  
**User action:** Correct the appropriate length field parameter.  
**REASONCODES:** TSS 048 (072) |
| 2AFC (11004)             | The hash value (of the secret quantities) in the private key section of the internal token failed validation. The values in the token are corrupted. You cannot use this key.  
**User action:** Recreate the token using the appropriate combination of the PKA key token build, PKA key generate, and PKA key import callable services.  
**REASONCODES:** TSS 02F (047) |
| 2B00 (11008)             | The public or private key values are not valid. (For example, the modulus or an exponent is zero.) You cannot use the key.  
**User action:** You may need to recreate the token using the PKA key token build or PKA key import callable service or regenerate the key values on another platform.  
**REASONCODES:** TSS 302 (770) |
| 2B04 (11012)             | The internal or external private key token contains flags that are not valid.  
**User action:** You may need to recreate the token using the PKA key token build or PKA key import callable service.  
**REASONCODES:** TSS 02F (047) |
| 2B08 (11016)             | The calculated hash of the public information in the PKA token does not match the hash in the private section of the token. The values in the token are corrupted.  
**User action:** Verify the public key section and the key name section of the token. If the token is still rejected, then you need to recreate the token using the appropriate combination of the PKA key token build, PKA key generate, and PKA key import callable services.  
**REASONCODES:** TSS 02F (047) |
| 2B0C (11020)             | The hash pattern of the PKA master key (SMK or KMMK) in the supplied internal PKA private key token does not match the current system's PKA master key. This indicates the system PKA master key has changed since the token was created. You cannot use the token.  
**User action:** Recreate the token using the appropriate combination of the PKA key token build, PKA key generate, and PKA key import callable services.  
**REASONCODES:** TSS 030 (048) |
| 2B10 (11024)             | The PKA tokens have incomplete values, for example, a PKA public key token without modulus.  
**User action:** Recreate the key.  
**REASONCODES:** TSS 02F (047) |
| 2B14 (11028)             | The modulus of the PKA key is too short for processing the hash or PKCS block.  
**User action:** Either use a PKA key with a larger modulus size, use a hash algorithm that generates a smaller hash (digital signature services), or specify a shorter DATA key size (symmetric key export, symmetric key generate).  
**REASONCODES:** TSS 048 (072) |
<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2B18 (11032)</td>
<td>The supplied private key can be used only for digital signature. Key management services are disallowed.</td>
</tr>
<tr>
<td>User action:</td>
<td>Supply a key with key management enabled.</td>
</tr>
<tr>
<td>REASONCODES:</td>
<td>TSS 040 (064)</td>
</tr>
<tr>
<td>2B20 (11040)</td>
<td>The recovered encryption block was not a valid PKCS-1.2 or zero-pad format. (The format is verified according to the recovery method specified in the rule-array.) If the recovery method specified was PKCS-1.2, refer to PKCS-1.2 for the possible error in parsing the encryption block.</td>
</tr>
<tr>
<td>User action:</td>
<td>Ensure that the parameters passed to CSNDSYI or CSNFSYI are correct. Possible causes for this error are incorrect values for the RSA private key or incorrect values in the RSA_enciphered_key parameter, which must be formatted according to PKCS-1.2 or zero-pad rules when created.</td>
</tr>
<tr>
<td>REASONCODES:</td>
<td>TSS 042 (066)</td>
</tr>
<tr>
<td>2B24 (11044)</td>
<td>The first section of a supplied PKA token was not a private or public key section.</td>
</tr>
<tr>
<td>User action:</td>
<td>Recreate the key.</td>
</tr>
<tr>
<td>REASONCODES:</td>
<td>TSS 0B5 (181)</td>
</tr>
<tr>
<td>2B28 (11048)</td>
<td>The eyecatcher on the PKA internal private token is not valid.</td>
</tr>
<tr>
<td>User action:</td>
<td>Reimport the private token using the PKA key import callable service.</td>
</tr>
<tr>
<td>2B2C (11052)</td>
<td>An incorrect PKA token was supplied. The service requires a private key token.</td>
</tr>
<tr>
<td>User action:</td>
<td>Supply a PKA private key token as input.</td>
</tr>
<tr>
<td>2B30 (11056)</td>
<td>The input PKA token contains length fields that are not valid.</td>
</tr>
<tr>
<td>User action:</td>
<td>Recreate the key token.</td>
</tr>
<tr>
<td>2B38 (11064)</td>
<td>The RSA-OAEP block did not verify when it decomposed. The block type is incorrect (must be X'03').</td>
</tr>
<tr>
<td>User action:</td>
<td>Recreate the RSA-OAEP block.</td>
</tr>
<tr>
<td>REASONCODES:</td>
<td>TSS 2CF (719)</td>
</tr>
<tr>
<td>2B3C (11068)</td>
<td>The RSA-OAEP block did not verify when it decomposed. The verification code is not correct (must be all zeros).</td>
</tr>
<tr>
<td>User action:</td>
<td>Recreate the RSA-OAEP block.</td>
</tr>
<tr>
<td>REASONCODES:</td>
<td>TSS 2D1 (721)</td>
</tr>
<tr>
<td>2B40 (11072)</td>
<td>The RSA-OAEP block did not verify when it decomposed. The random number I is not correct (must be non-zero with the high-order bit equal to zero).</td>
</tr>
<tr>
<td>User action:</td>
<td>Recreate the RSA-OAEP block.</td>
</tr>
<tr>
<td>REASONCODES:</td>
<td>TSS 2D0 (720)</td>
</tr>
<tr>
<td>2B48 (11080)</td>
<td>The RSA public or private key specified a modulus length that is incorrect for this service.</td>
</tr>
<tr>
<td>User action:</td>
<td>Re-invoke the service with an RSA key with the proper modulus length.</td>
</tr>
<tr>
<td>REASONCODES:</td>
<td>See reason codes 041 (065) and 2F8 (760)</td>
</tr>
<tr>
<td>Reason Code Hex (Decimal)</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>2B4C (11084)</td>
<td>This service requires an RSA public key and the key identifier specified is not a public key. <strong>User action:</strong> Re-invoke the service with an RSA public key.</td>
</tr>
<tr>
<td>2B50 (11088)</td>
<td>This service requires an RSA private key that is for signature use only. <strong>User action:</strong> Re-invoke the service with a supported private key.</td>
</tr>
<tr>
<td>2B54 (11092)</td>
<td>There was an invalid subsection in the PKA token. <strong>User action:</strong> Correct the PKA token.</td>
</tr>
<tr>
<td>2B58 (11096)</td>
<td>This service requires an RSA private key that is for signature use. The specified key may be used for key management purposes only. <strong>User action:</strong> Re-invoke the service with a supported private key.</td>
</tr>
<tr>
<td>REASONCODES: TSS 040 (064)</td>
<td></td>
</tr>
<tr>
<td>3E80 (16000)</td>
<td>RACF failed your request to use this service. <strong>User action:</strong> Contact your ICSF or RACF administrator if you need this service.</td>
</tr>
<tr>
<td>3E84 (16004)</td>
<td>RACF failed your request to use the key label. This may be caused by either CSFKEYS or XCSFKEY class, depending on the setting of the Granular Keylabel Access Controls and the type of token provided. <strong>User action:</strong> Contact your ICSF or RACF administrator if you need this key.</td>
</tr>
<tr>
<td>3E8C (16012)</td>
<td>You requested the conversion service, but you are not running in an authorized state. <strong>User action:</strong> You must be running in supervisor state to use the conversion service. Contact your ICSF administrator.</td>
</tr>
<tr>
<td>3E90 (16016)</td>
<td>The input/output field contained a valid internal token with the NOCV bit on or encryption algorithm mark, but the key type was incorrect or did not match the type of the generated or imported key. Processing failed. <strong>User action:</strong> Correct the calling application. <strong>REASONCODES:</strong> TSS 027 (039)</td>
</tr>
<tr>
<td>3E94 (16020)</td>
<td>You requested dynamic CKDS update services for a system key, which is not allowed. <strong>User action:</strong> Correct the calling application. <strong>REASONCODES:</strong> TSS 0B5 (181)</td>
</tr>
<tr>
<td>3E98 (16024)</td>
<td>You called the key record write callable service, but the key token you supplied is not valid. <strong>User action:</strong> Check with your ICSF administrator if you believe that this key should be in the in-storage CKDS. The administrator may be able to bring it into storage. If this key cannot be in storage, use a different label.</td>
</tr>
<tr>
<td>3EA0 (16032)</td>
<td>Invalid syntax for CKDS or PKDS label name. <strong>User action:</strong> Correct key_label syntax. <strong>REASONCODES:</strong> TSS 020 (032)</td>
</tr>
</tbody>
</table>
Table 237. Reason Codes for Return Code 8 (8) (continued)

<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 3EA4 (16036)              | The key record create callable service requires that the key created not already exist in the CKDS or PKDS. A key of the same label was found. 
  **User action:** Make sure the application specifies the correct label. If the label is correct, contact your ICSF security administrator or system programmer. |
| 3EA8 (16040)              | Data in the PKDS record did not match the expected data. This occurs if the record does not contain a null PKA token and CHECK was specified. 
  **User action:** If the record is to be overwritten regardless of its content, specify OVERLAY. |
| 3EAC (16044)              | One or more key labels specified as input to the PKA key generate or PKA key import service incorrectly refer to a retained private key. If generating a retained private key, this error may result from one of these conditions: 
  - The private key name of the retained private key being generated is the same as an existing PKDS record, but the PKDS record label was not specified as the input skeleton (source) key identifier. 
  - The label specified in the `generated_key_token` parameter as the target for the retained private key was not the same as the private key name. 
  If generating or importing a non-retained key, this error occurs when the label specified as the target key specifies a retained private key. The retained private key cannot be over-written. 
  **User action:** Make sure the application specifies the correct label. If the label is correct, contact your ICSF security administrator or system programmer. |
| 3EB0 (16048)              | Retained keys on the PKDS cannot be deleted or updated using the PKDS key record delete or PKDS key record write callable services, respectively. 
  **User action:** Use the retained key delete callable service to delete retained keys. |

Reason code 0, return code 308 (776)

RACF failed your request to use this service. 
  **User action:** Contact your ICSF or RACF administrator if you need this service.

Reason code 1, return code 308 (776)

RACF failed your request to use the key label. 
  **User action:** Contact your ICSF or RACF administrator if you need this key.

06E (110)-PAN, 028 (040)-ser. code, 02A (042)-exp. date, 066 (102)-dec table, 067 (103)-val. table, 06C (198)-pad data

The PAN, expiration date, service code, decimalization table data, validation data, or pad data is not numeric (X’F0’ through X’F9’). The parameter must be character representations of numerics or hexadecimal data. 
  **User action:** Review the numeric parameters or fields required in the service that you called and change to the format and values required.

### Reason Codes for Return Code C (12)

Table 238 on page 590 lists reason codes returned from callable services that give return code 12. These reason codes indicate that the call to the callable service was not successful. Either cryptographic processing did not take place, or the last cryptographic unit was switched offline. Therefore, no output parameters were filled.

**Note:** The higher-order halfword of the reason code field for return code C (12) may contain additional coding. See reason codes 273C and 2740 in this table. For example, in the reason code 42738, the 4 is an SVC 99 error code and the 2738 is listed in this table:
<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>ICSF is not available. One of the following situations is possible:</td>
</tr>
<tr>
<td></td>
<td>• ICSF is not started</td>
</tr>
<tr>
<td></td>
<td>• ICSF is started, but does not have access to any cryptographic units.</td>
</tr>
<tr>
<td></td>
<td>• ICSF is started, but the DES-MK or AES-MK is not defined.</td>
</tr>
<tr>
<td></td>
<td><strong>User action</strong>: Check the availability of ICSF with your ICSF administrator.</td>
</tr>
<tr>
<td>4 (4)</td>
<td>The CKDS or PKDS management service you called is not available because it has been disallowed by the ICSF User Control Functions panel.</td>
</tr>
<tr>
<td></td>
<td><strong>User action</strong>: Contact the security administrator or system programmer to determine why the CKDS or PKDS management services have been disallowed.</td>
</tr>
<tr>
<td>8 (8)</td>
<td>The service or algorithm is not available on current hardware. Your request cannot be processed.</td>
</tr>
<tr>
<td></td>
<td><strong>User action</strong>: Correct the calling program or run on applicable hardware.</td>
</tr>
<tr>
<td>C (12)</td>
<td>The service that you called is unavailable because the installation exit for that service had previously failed.</td>
</tr>
<tr>
<td></td>
<td><strong>User action</strong>: Contact your ICSF administrator or system programmer.</td>
</tr>
<tr>
<td>10 (16)</td>
<td>A requested installation service routine could not be found. Your request was not processed.</td>
</tr>
<tr>
<td></td>
<td><strong>User action</strong>: Contact your ICSF administrator or system programmer.</td>
</tr>
<tr>
<td>1C (28)</td>
<td>Cryptographic asynchronous processor failed.</td>
</tr>
<tr>
<td></td>
<td><strong>User action</strong>: Contact your IBM support center.</td>
</tr>
<tr>
<td>20 (32)</td>
<td>Cryptographic asynchronous instruction was not executed.</td>
</tr>
<tr>
<td></td>
<td><strong>User action</strong>: Ensure cryptographic services are enabled.</td>
</tr>
<tr>
<td>28 (40)</td>
<td>The callable service that you called is unsupported for AMODE(64) applications. Your request cannot be processed.</td>
</tr>
<tr>
<td>2C (44)</td>
<td>The callable service that you called was linked with the AMODE(64) stub. The application is not running AMODE(64). Your request cannot be processed.</td>
</tr>
<tr>
<td></td>
<td><strong>User action</strong>: Link your application with the service stub with the appropriate addressing mode.</td>
</tr>
<tr>
<td>0C5 (197)</td>
<td>I/O error reading or writing to the DASD copy of the CKDS or PKDS in use by ICSF.</td>
</tr>
<tr>
<td></td>
<td><strong>User action</strong>: Contact your ICSF security administrator or system programmer. The RPL feedback code will be placed in the high-order halfword of the reason code field.</td>
</tr>
<tr>
<td>144 (324)</td>
<td>There was insufficient coprocessor memory available to process your request. This could include the Flash EPROM used to store keys, profiles and other application data.</td>
</tr>
<tr>
<td></td>
<td><strong>User action</strong>: Contact your system programmer or the IBM Support Center.</td>
</tr>
<tr>
<td>2FC (764)</td>
<td>The master key is not in a valid state.</td>
</tr>
<tr>
<td></td>
<td><strong>User action</strong>: Contact your ICSF administrator.</td>
</tr>
<tr>
<td>7D7 (2007)</td>
<td>TKE: Change type in PCB is not recognized.</td>
</tr>
<tr>
<td>Reason Code Hex (Decimal)</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>7E1 (2017)</td>
<td>MKVP mismatch on Set MK.</td>
</tr>
<tr>
<td>7E5 (2021)</td>
<td>PCI Cryptographic Coprocessor , PCI X Cryptographic Coprocessor, Crypto Express2 Coprocessor, or Crypto Express3 Coprocessor adapter disabled.</td>
</tr>
<tr>
<td>7E9 (2025)</td>
<td>Enforcement mask error.</td>
</tr>
<tr>
<td>7F3 (2035)</td>
<td>Intrusion latch has been tripped. Services disabled.</td>
</tr>
<tr>
<td>7F5 (2037)</td>
<td>The domain specified is not valid.</td>
</tr>
<tr>
<td>7FB (2043)</td>
<td>OA certificate not found.</td>
</tr>
<tr>
<td>819 (2073)</td>
<td>The PCIXCC, CEX2C, or CEX3C has been disabled on the Support Element. It must be enabled on the Support Element prior to TKE accessing it.</td>
</tr>
<tr>
<td><strong>User action</strong>: Permit the selected PCIXCC, CEX2C, or CEX3C for TKE Commands on the Support Element and then re-open the Host on TKE.</td>
<td></td>
</tr>
<tr>
<td>835 (2101)</td>
<td>AES flags in the function control vector are not valid.</td>
</tr>
<tr>
<td>BBD (3005)</td>
<td>The CKDS I/O subtask timed out waiting for an exclusive ENQ on the SYSZCKDS.CKDSdsn resource. A timeout will occur if one or more members of the ICSF sysplex group has not relinquished its ENQ on the resource. The CKDS update operation has failed.</td>
</tr>
<tr>
<td><strong>User action</strong>: Issue D GRS,RES=(nnnnn), where nnnnn is the CKDS resource name from message CSFM302A, to determine which system or systems hold the resource. Determine if action should be taken to cause the holding system to release its ENQ on the CKDS resource.</td>
<td></td>
</tr>
<tr>
<td>BBE (3006)</td>
<td>Failure after exhausting retry attempts. IXCMSGO issued from CSFMIOST.</td>
</tr>
<tr>
<td><strong>User action</strong>: Contact your system programmer or the IBM Support Center.</td>
<td></td>
</tr>
<tr>
<td>BBF (3007)</td>
<td>The CKDS service failed due to unexpected termination of the ICSF Cross-System Services environment. The termination of the ICSF Cross-System Services environment was caused by a failure when ICSF issued the IXCMSGI macro. Message CSFM603 has been issued.</td>
</tr>
<tr>
<td><strong>User action</strong>: Report the occurrence of this error to your ICSF system programmer.</td>
<td></td>
</tr>
<tr>
<td>BC0 (3008)</td>
<td>The TKDS I/O subtask timed out waiting for an exclusive ENQ on the SYSZTKDS.TKDSdsn resource. A timeout will occur if one or more members of the ICSF sysplex group has not relinquished its ENQ on the resource. The TKDS update operation has failed. The operator should issue D GRS,RES=(nnnnn) (where nnnnn is the TKDS resource name from message CSFM305A) to determine which system or systems hold the resource. Then the operator should determine if action should be taken to cause the holding system to release its ENQ on the TKDS resource.</td>
</tr>
<tr>
<td><strong>User action</strong>: Report the occurrence of this error to your ICSF system programmer.</td>
<td></td>
</tr>
<tr>
<td>BC6 (3014)</td>
<td>There is an I/O error reading or writing to the DASD copy of the TKDS in use by ICSF.</td>
</tr>
<tr>
<td><strong>User action</strong>: Report the occurrence of this error to your ICSF system programmer.</td>
<td></td>
</tr>
<tr>
<td>BC7 (3015)</td>
<td>A bad header record is detected for the TKDS in CSFMTDSL.</td>
</tr>
<tr>
<td><strong>User action</strong>: Report the occurrence of this error to your ICSF system programmer.</td>
<td></td>
</tr>
<tr>
<td>BCF (3023)</td>
<td>The PKCS #11 TKDS is not available for processing.</td>
</tr>
<tr>
<td><strong>User action</strong>: Report the occurrence of this error to your ICSF system programmer.</td>
<td></td>
</tr>
<tr>
<td>BE6 (3046)</td>
<td>An RSA retained key can no longer be generated with its key-usage flag set to allow key unwrapping (KM-ONLY or KEY-MGMT). Key usage must be SIG-ONLY.</td>
</tr>
<tr>
<td><strong>User action</strong>: None required.</td>
<td></td>
</tr>
</tbody>
</table>

Appendix A. ICSF and TSS Return and Reason Codes 591
<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
</table>
| BE8 (3048) | The services using secure AES keys or secure DES keys are not available because the master key is required but not loaded or there is no access to any cryptographic units. Your request cannot be processed.  
**User action:** Check the availability of ICSF with your ICSF administrator |
| BF2 (3058) | A PKDS sysplex operation has been waiting unsuccessfully to obtain an enqueue. A failure to obtain either the SYSZPKDS or SYSZPKUP resource will result in the timeout. |
| C00 (3072) | The serialization subtask terminated for an unexpected reason prior to completing the request. No dynamic CKDS or PKDS update services are possible at this point.  
**User action:** Contact your system programmer who can investigate the problem and restart the I/O subtask by stopping and restarting ICSF. |
| C01 (3073) | An error occurred attempting to obtain the system ENQ for a key data set update.  
**User action:** If the error is common and persistent, contact your system programmer or the IBM Support Center. |
| 1780 (6016) | A DASD IO error was encountered during access of the CKDS, PKDS, or TKDS.  
**User action:** Contact your ICSF security administrator or system programmer. The SVC 99 error code will be placed in the high-order halfword of the reason code field. |
| 178C (6028) | ESTAE could not be established in common I/O routines.  
**User action:** Contact your system programmer or the IBM Support Center. |
| 1790 (6032) | The dynamic allocation of the DASD copy of the CKDS, PKDS, or TKDS in use by ICSF failed.  
**User action:** Contact your ICSF security administrator or system programmer. The SVC 99 error code will be placed in the high-order halfword of the reason code field. |
| 1794 (6036) | A dynamic deallocation error occurred when closing and deallocating a CKDS, PKDS, or TKDS.  
**User action:** Contact your security administrator or system programmer. The SVC 99 error code will be placed in the high-order halfword of the reason code field. |
| 2724 (10020) | A key retrieved from the in-storage CKDS failed the MAC verification (MACVER) check and is unusable.  
**User action:** Contact your ICSF administrator. |
| 2728 (10024) | A key retrieved from the in-storage CKDS or a key to be written to the PKDS was rejected for use by the installation exit.  
**User action:** Contact your ICSF administrator or system programmer. |
| 272C (10028) | You cannot use the secure key import or multiple secure key import callable services because the cryptographic unit is not enabled for processing. The cryptographic unit is not in special secure mode or is disabled in the environment control mask (ECM).  
**User action:** Contact your ICSF administrator (your administrator can enable the processing mode or the ECM). |
| 2734 (10036) | More than one key with the same label was found in the CKDS or PKDS. This function requires a unique key per label. The probable cause may be the use of an incorrect label pointing to a key type that allows multiple keys per label.  
**User action:** Make sure the application specifies the correct label. If the label is correct, contact your ICSF security administrator or system programmer to verify the contents of the CKDS or PKDS. |
<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
<th>User action</th>
</tr>
</thead>
<tbody>
<tr>
<td>273C (10044)</td>
<td>OPEN of the PKDS in use by ICSF failed.</td>
<td>Contact your ICSF security administrator or system programmer.</td>
</tr>
<tr>
<td>2740 (10048)</td>
<td>I/O error reading or writing to the DASD copy of the CKDS or PKDS in use by ICSF.</td>
<td>Contact your ICSF security administrator or system programmer. The RPL feedback code will be placed in the high-order halfword of the reason code field.</td>
</tr>
<tr>
<td>2744 (10052)</td>
<td>Automatic REFRESH to free storage in the linear section of the CKT failed.</td>
<td>Contact your ICSF security administrator or system programmer and request that a REFRESH be done.</td>
</tr>
<tr>
<td>274C (10060)</td>
<td>The I/O subtask terminated for an unexpected reason prior to completing the request. No dynamic CKDS or PKDS update services are possible at this point.</td>
<td>Contact your system programmer who can investigate the problem and restart the I/O subtask by stopping and restarting ICSF.</td>
</tr>
<tr>
<td>2B04 (11012)</td>
<td>This function is disabled in the environment control mask (ECM).</td>
<td>Contact your ICSF administrator.</td>
</tr>
<tr>
<td>2B08 (11016)</td>
<td>The master key is not in a valid state.</td>
<td>Contact your ICSF administrator.</td>
</tr>
<tr>
<td>2B0C (11020)</td>
<td>The modulus of the public or private key is larger than allowed and configured in the CCC or FCV. You cannot use this key on this system.</td>
<td>Regenerate the key with a smaller modulus size.</td>
</tr>
<tr>
<td>2B10 (11024)</td>
<td>The system administrator has used the ICSF User Control Functions panel to disable the PKA functions.</td>
<td>Wait until administrator functions are complete and the PKA functions are again enabled.</td>
</tr>
<tr>
<td>2B18 (11032)</td>
<td>A CAMQ is valid for PKSC but not for PKA.</td>
<td>Contact your ICSF administrator.</td>
</tr>
<tr>
<td>2B1C (11036)</td>
<td>A PKDS is not available for processing.</td>
<td>Contact your ICSF administrator.</td>
</tr>
<tr>
<td>2B20 (11040)</td>
<td>The PKDS Control Record hash pattern is not valid.</td>
<td>Contact your ICSF administrator.</td>
</tr>
<tr>
<td>2B24 (11044)</td>
<td>The PKDS could not be accessed.</td>
<td>Contact your ICSF administrator.</td>
</tr>
<tr>
<td>2B28 (11048)</td>
<td>The PCICC, PCIXCC, CEX2C, or CEX3C failed.</td>
<td>Contact your IBM support center.</td>
</tr>
<tr>
<td>Reason Code Hex (Decimal)</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>2B2C (11052)</td>
<td>The specific PCICC, PCIXCC, CEX2C, or CEX3C requested for service is temporarily unavailable. PKDS could not be accessed. The specific PCICC, PCIXCC, CEX2C, or CEX3C may be attempting some recovery action. If recovery action is successful, the PCICC, PCIXCC, CEX2C, or CEX3C will be made available. If the recovery action fails, the PCICC, PCIXCC, CEX2C, or CEX3C will be made permanently unavailable. <strong>User action:</strong> Retry the function.</td>
<td></td>
</tr>
<tr>
<td>2B30 (11056)</td>
<td>The PCICC, PCIXCC, CEX2C, or CEX3C failed. The response from the processor was incomplete. <strong>User action:</strong> Contact your IBM support center.</td>
<td></td>
</tr>
<tr>
<td>2B34 (11060)</td>
<td>The service could not be performed because the required PCICC, PCIXCC, CEX2C, or CEX3C was not active, or did not have a master key set. <strong>User action:</strong> If the service required a specific PCICC, PCIXCC, CEX2C, or CEX3C, verify that the value specified is correct. Reissue the request when the required PCICC, PCIXCC, CEX2C, or CEX3C is available, and has the master key set.</td>
<td></td>
</tr>
<tr>
<td>2B38 (11064)</td>
<td>Service could not be performed because of a hardware error on the PCICC, PCIXCC, CEX2C, or CEX3C.</td>
<td></td>
</tr>
<tr>
<td>2B40 (11072)</td>
<td>CEX2C has been reconfigured to a CEX2A, or CEX3C has been reconfigured to a CEX3A. TKE will not recognize the coprocessor until it is reconfigured back to a CEX2C or CEX3C.</td>
<td></td>
</tr>
<tr>
<td>2EDC (11996)</td>
<td>The Cryptographic Coprocessor Feature is not available for CKDS initialization because the cryptographic unit is not in special secure mode. <strong>User action:</strong> Contact your ICSF administrator.</td>
<td></td>
</tr>
<tr>
<td>2EE0 (12000)</td>
<td>You cannot use the Clear PIN generate callable service because the cryptographic unit is not enabled for processing. The cryptographic unit is not in special secure mode. <strong>User action:</strong> Contact your ICSF administrator who can enable the processing mode.</td>
<td></td>
</tr>
<tr>
<td>8CB4 (36020)</td>
<td>A refresh of the CKDS failed because the DASD copy of the CKDS is enciphered under the wrong master key. This may have resulted from an automatic refresh during processing of the key record create callable service. <strong>User action:</strong> Contact your ICSF administrator.</td>
<td></td>
</tr>
<tr>
<td>8D14 (36116)</td>
<td>The PKDS specified for refresh, reencipher or activate has an incorrect dataset attribute. <strong>User action:</strong> Create a larger PKDS. See <a href="http://www.ibm.com/support/knowledgecenter/">z/OS Cryptographic Services ICSF System Programmer's Guide</a>.</td>
<td></td>
</tr>
<tr>
<td>8D3C (36156)</td>
<td>A PKCS #11 service is being requested. The service is disabled due to an ICSF FIPS self test failure. The request is not processed. <strong>User action:</strong> Report the problem to your IBM support center</td>
<td></td>
</tr>
</tbody>
</table>

### Reason Codes for Return Code 10 (16)

Table 239 on page 595 lists reason codes returned from callable services that give return code 16.
<table>
<thead>
<tr>
<th>Reason Code Hex (Decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 (4)</td>
<td>ICSF: Your call to an ICSF callable service resulted in an abnormal ending.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Contact your system programmer or the IBM Support Center.</td>
</tr>
<tr>
<td>150 (336)</td>
<td>An error occurred in the cryptographic hardware component.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Contact your system programmer or the IBM Support Center.</td>
</tr>
<tr>
<td></td>
<td><strong>REASONCODES:</strong> ICSF 4 (4)</td>
</tr>
<tr>
<td>22C (556)</td>
<td>The request parameter block failed consistency checking.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Contact your system programmer or the IBM Support Center.</td>
</tr>
<tr>
<td></td>
<td><strong>REASONCODES:</strong> ICSF 4 (4)</td>
</tr>
<tr>
<td>2C4 (708)</td>
<td>Inconsistent data was returned from the cryptographic engine.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Contact your system programmer or the IBM Support Center.</td>
</tr>
<tr>
<td></td>
<td><strong>REASONCODES:</strong> ICSF 4 (4)</td>
</tr>
<tr>
<td>2C5 (709)</td>
<td>Cryptographic engine internal error; could not access the master key data.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Contact your system programmer or the IBM Support Center.</td>
</tr>
<tr>
<td></td>
<td><strong>REASONCODES:</strong> ICSF 4 (4)</td>
</tr>
<tr>
<td>2C8 (712)</td>
<td>An unexpected error occurred in the Master Key manager.</td>
</tr>
<tr>
<td></td>
<td><strong>User action:</strong> Contact your system programmer or the IBM Support Center.</td>
</tr>
<tr>
<td></td>
<td><strong>REASONCODES:</strong> ICSF 4 (4)</td>
</tr>
</tbody>
</table>
Appendix B. Key Token Formats

For debugging purposes, this appendix provides the formats for AES, DES internal, external, and null key tokens and for PKA key tokens.

- "AES Internal Key Token"
- "DES Internal Key Token" on page 598
- "DES External Key Token" on page 599
- "External RKX DES Key Token" on page 600
- "DES Null Key Token" on page 601
- "PKA Null Key Token" on page 602
- "RSA Public Key Token" on page 602
- "RSA Private External Key Token" on page 603
  - "RSA Private Key Token, 1024-bit Modulus-Exponent External Form" on page 604
  - "RSA Private Key Token, 4096-bit Modulus-Exponent External Form" on page 605
  - "RSA Private Key Token, 4096-bit Chinese Remainder Theorem External Form" on page 606
- "RSA Private Internal Key Token" on page 608
  - "RSA Private Key Token, 1024-bit Modulus-Exponent Internal Form for Cryptographic Coprocessor Feature" on page 609
  - "RSA Private Key Token, 1024-bit Modulus-Exponent Internal Form for PCICC, PCIXCC, CEX2C, or CEX3C" on page 610
  - "RSA Private Key Token, 4096-bit Chinese Remainder Theorem Internal Form" on page 611
- "DSS Public Key Token" on page 612
- "DSS Private External Key Token" on page 613
- "DSS Private Internal Key Token" on page 615
- "Trusted Block Key Token" on page 616

AES Key Token Formats

AES Internal Key Token

Table 240 shows the format for an AES internal key token.

Table 240. Internal Key Token Format

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X'01' (flag indicating this is an internal key token)</td>
</tr>
<tr>
<td>1–3</td>
<td>Implementation-dependent bytes (X'000000' for ICSF)</td>
</tr>
<tr>
<td>4</td>
<td>Key token version number (X'04')</td>
</tr>
<tr>
<td>5</td>
<td>Reserved - must be set to X'00'</td>
</tr>
</tbody>
</table>
Table 240. Internal Key Token Format (continued)

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Flag byte</td>
</tr>
<tr>
<td></td>
<td><strong>Bit</strong></td>
</tr>
</tbody>
</table>
| 0     | Encrypted key and master key verification pattern (MKVP) are present.  
|       | Off for a clear key token, on for an encrypted key token. |
| 1     | Control vector (CV) value in this token has been applied to the key. |
| 2     | No key is present or the AES MKVP is not present if the key is encrypted. |
| 3 - 7 | Reserved. Must be set to 0. |
| 7     | 1-byte LRC checksum of clear key value. |
| 8–15  | Master key verification pattern (MKVP)  
|       | (For a clear AES key token this value will be hex zeros.) |
| 16–47 | 128-bit, 192-bit, or 256-bit key value, left-justified and padded on the right with hex zeros. |
| 48–55 | 8-byte control vector.  
|       | (For a clear AES key token this value will be hex zeros.) |
| 56–57 | 2-byte integer specifying the length in bits of the clear key value. |
| 58–59 | 2-byte integer specifying the length in bytes of the encrypted key value.  
|       | (For a clear AES key token this value will be hex zeros.) |
| 60–63 | Token validation value (TVV). See [Token Validation Value](#) for more information. |

**Token Validation Value**

ICSF uses the *token validation value (TVV)* to verify that a token is valid. The TVV prevents a key token that is not valid or that is overlaid from being accepted by ICSF. It provides a checksum to detect a corruption in the key token.

When an ICSF callable service generates a key token, it generates a TVV and stores the TVV in bytes 60-63 of the key token. When an application program passes a key token to a callable service, ICSF checks the TVV. To generate the TVV, ICSF performs a two's complement ADD operation (ignoring carries and overflow) on the key token, operating on four bytes at a time, starting with bytes 0-3 and ending with bytes 56-59.

**DES Key Token Formats**

**DES Internal Key Token**

[Table 241](#) shows the format for a DES internal key token.

Table 241. Internal Key Token Format

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X'01' (flag indicating this is an internal key token)</td>
</tr>
<tr>
<td>1–3</td>
<td>Implementation-dependent bytes (X'000000' for ICSF)</td>
</tr>
<tr>
<td>4</td>
<td>Key token version number (X'00' or X'01')</td>
</tr>
<tr>
<td>5</td>
<td>Reserved (X'00')</td>
</tr>
</tbody>
</table>
### Table 241. Internal Key Token Format (continued)

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Flag byte</td>
</tr>
<tr>
<td>Bit</td>
<td>Meaning When Set On</td>
</tr>
<tr>
<td>0</td>
<td>Encrypted key and master key verification pattern (MKVP) are present.</td>
</tr>
<tr>
<td>1</td>
<td>Control vector (CV) value in this token has been applied to the key.</td>
</tr>
<tr>
<td>2</td>
<td>Key is used for no control vector (NOCV) processing. Valid for transport keys only.</td>
</tr>
<tr>
<td>3</td>
<td>Key is an ANSI key-encrypting key (AKEK).</td>
</tr>
<tr>
<td>4</td>
<td>AKEK is a double-length key (16 bytes). <strong>Note:</strong> When bit 3 is on and bit 4 is off, AKEK is a single-length key (8 bytes).</td>
</tr>
<tr>
<td>5</td>
<td>AKEK is partially notarized.</td>
</tr>
<tr>
<td>6</td>
<td>Key is an ANSI partial key.</td>
</tr>
<tr>
<td>7</td>
<td>Export prohibited.</td>
</tr>
<tr>
<td>7</td>
<td>Reserved (X'00')</td>
</tr>
<tr>
<td>8–15</td>
<td>Master key verification pattern (MKVP)</td>
</tr>
<tr>
<td>16–23</td>
<td>A single-length key, the left half of a double-length key, or Part A of a triple-length key. The value is encrypted under the master key when flag bit 0 is on, otherwise it is in the clear.</td>
</tr>
<tr>
<td>24–31</td>
<td>X'0000000000000000' if a single-length key, or the right half of a double-length operational key, or Part B of a triple-length operational key. The right half of the double-length key or Part B of the triple-length key is encrypted under the master key when flag bit 0 is on, otherwise it is in the clear.</td>
</tr>
<tr>
<td>32–39</td>
<td>The control vector (CV) for a single-length key or the left half of the control vector for a double-length key.</td>
</tr>
<tr>
<td>40–47</td>
<td>X'0000000000000000' if a single-length key or the right half of the control vector for a double-length operational key.</td>
</tr>
<tr>
<td>48–55</td>
<td>X'0000000000000000' if a single-length key or double-length key, or Part C of a triple-length operational key. Part C of a triple-length key is encrypted under the master key when flag bit 0 is on, otherwise it is in the clear.</td>
</tr>
<tr>
<td>56–58</td>
<td>Reserved (X'000000')</td>
</tr>
<tr>
<td>59 bits 0 and 1</td>
<td>B'10' Indicates CDMF DATA or KEK.</td>
</tr>
<tr>
<td></td>
<td>B'00' Indicates DES for DATA keys or the system default algorithm for a KEK.</td>
</tr>
<tr>
<td></td>
<td>B'01' Indicates DES for a KEK.</td>
</tr>
<tr>
<td>59 bits 2 and 3</td>
<td>B'00' Indicates single-length key (version 0 only).</td>
</tr>
<tr>
<td></td>
<td>B'01' Indicates double-length key (version 1 only).</td>
</tr>
<tr>
<td></td>
<td>B'10' Indicates triple-length key (version 1 only).</td>
</tr>
<tr>
<td>59 bits 4–7</td>
<td>B'0000'</td>
</tr>
<tr>
<td>60–63</td>
<td>Token validation value (TVV).</td>
</tr>
</tbody>
</table>

**Note:** A key token stored in the CKDS will not have an MKVP or TVV. Before such a key token is used, the MKVP is copied from the CKDS header record and the TVV is calculated and placed in the token. See [Token Validation Value](#) on page 598 for more information.

### DES External Key Token

[Table 242 on page 600](#) shows the format for a DES external key token.
Table 242. Format of External Key Tokens

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X'02' (flag indicating an external key token)</td>
</tr>
<tr>
<td>1</td>
<td>Reserved (X'00')</td>
</tr>
<tr>
<td>2–3</td>
<td>Implementation-dependent bytes (X'0000' for ICSF)</td>
</tr>
<tr>
<td>4</td>
<td>Key token version number (X'00' or X'01')</td>
</tr>
<tr>
<td>5</td>
<td>Reserved (X'00')</td>
</tr>
<tr>
<td>6</td>
<td>Flag byte</td>
</tr>
<tr>
<td></td>
<td><strong>Bit</strong></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Reserved (X'00')</td>
</tr>
<tr>
<td>8–15</td>
<td>Reserved (X'0000000000000000')</td>
</tr>
<tr>
<td>16–23</td>
<td>Single-length key or left half of a double-length key, or Part A of a triple-length key. The value is encrypted under a transport key-encrypting key when flag bit 0 is on, otherwise it is in the clear.</td>
</tr>
<tr>
<td>24–31</td>
<td>X'000000000000000000000000' if a single-length key or right half of a double-length key, or Part B of a triple-length key. The right half of a double-length key or Part B of a triple-length key is encrypted under a transport key-encrypting key when flag bit 0 is on, otherwise it is in the clear.</td>
</tr>
<tr>
<td>32–39</td>
<td>Control vector (CV) for single-length key or left half of CV for double-length key</td>
</tr>
<tr>
<td>40–47</td>
<td>X'000000000000000000000000' if single-length key or right half of CV for double-length key</td>
</tr>
<tr>
<td>48–55</td>
<td>X'00000000000000000000000000000000' if a single-length key, double-length key, or Part C of a triple-length key. This key part is encrypted under a transport key-encrypting key when flag bit 0 is on, otherwise it is in the clear.</td>
</tr>
<tr>
<td>56–58</td>
<td>Reserved (X'00000000')</td>
</tr>
<tr>
<td>59 bits 0 and 1</td>
<td>B'00'</td>
</tr>
<tr>
<td>59 bits 2 and 3</td>
<td><strong>B'00'</strong></td>
</tr>
<tr>
<td></td>
<td><strong>B'01'</strong></td>
</tr>
<tr>
<td></td>
<td><strong>B'10'</strong></td>
</tr>
<tr>
<td>60–63</td>
<td>Token validation value (see “Token Validation Value” on page 598 for a description).</td>
</tr>
</tbody>
</table>

**External RKX DES Key Token**

Table 243 on page 601 defines an external DES key-token called an **RKX key-token**. An RKX key-token is a special token used exclusively by the Remote Key Export (CSNDRKX) and DES key-storage callable services (for example, Key Record Write). No other callable services use or reference an RKX key-token or key-token record. For additional information about the usage of RKX key tokens, see “Remote Key Loading” on page 29.

**Note:** Callable services other than CSNDRKX and the DES key-storage do not support RKX key tokens or RKX key token records.

As can be seen in the table, RKX key tokens are 64 bytes in length, have a token identifier flag (X'02'), a token version number (X'10'), and room for encrypted keys like normal CCA DES key tokens. Unlike normal CCA DES key-tokens, RKX key tokens do not have a control vector, flag bits, and a token-validation value. In
addition, they have a confounder value, a MAC value, and room for a third encrypted key.

Table 243. External RKX DES key-token format, version X'10'

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1</td>
<td>X'02' (a token identifier flag that indicates an external key-token)</td>
</tr>
<tr>
<td>01</td>
<td>3</td>
<td>Reserved, binary zero</td>
</tr>
<tr>
<td>04</td>
<td>1</td>
<td>The token version number (X'10')</td>
</tr>
<tr>
<td>05</td>
<td>2</td>
<td>Reserved, binary zero</td>
</tr>
<tr>
<td>07</td>
<td>1</td>
<td>Key length in bytes, including confounder</td>
</tr>
<tr>
<td>08</td>
<td>8</td>
<td>Confounder</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>Key left</td>
</tr>
<tr>
<td>24</td>
<td>8</td>
<td>Key middle (binary zero if not used)</td>
</tr>
<tr>
<td>32</td>
<td>8</td>
<td>Key right (binary zero if not used)</td>
</tr>
<tr>
<td>40</td>
<td>8</td>
<td>Rule ID</td>
</tr>
</tbody>
</table>

The trusted block rule identifier used to create this key token. A subsequent call to Remote Key Export (CSNDRKX) can use this token with a trusted block rule that references the rule ID that must have been used to create this token. The trusted block rule can be compared with this rule ID for verification purposes.

The Rule ID is an 8-byte string of ASCII characters, left justified and padded on the right with space characters. Acceptable characters are A...Z, a...z, 0...9, X'2D', and X'5F'. All other characters are reserved for future use.

| 48     | 8      | Reserved, binary zero |
| 56     | 8      | MAC value |

ISO 16609 TDES CBC-mode MAC, computed over the 56 bytes starting at offset 0 and including the encrypted key value and the rule ID using the same MAC key that is used to protect the trusted block itself.

This MAC value guarantees that the key and the rule ID cannot be modified without detection, providing integrity and binding the rule ID to the key itself. This MAC value must verify with the same trusted block used to create the key, thus binding the key structure to that specific trusted block.

Notes:
1. A fixed, randomly derived variant is exclusive-ORed with the MAC key before it is used to encipher the generated or exported key and confounder.
2. The MAC key is located within a trusted block (internal format) and can be recovered by decipherment under a variant of the PKA master key.
3. The trusted block is originally created in external form by the CSNDTBC callable service and then converted to internal form by the CSNDPKI callable service prior to the CSNDRKX call.

**DES Null Key Token**

Table 244 on page 602 shows the format for a DES null key token.
Table 244. Format of Null Key Tokens

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X'00' (flag indicating this is a null key token).</td>
</tr>
<tr>
<td>1–15</td>
<td>Reserved (set to binary zeros).</td>
</tr>
<tr>
<td>16–23</td>
<td>Single-length encrypted key, or left half of double-length encrypted key, or Part A of triple-length encrypted key.</td>
</tr>
<tr>
<td>24–31</td>
<td>X'0000000000000000' if a single-length encrypted key, the right half of double-length encrypted key, or Part B of triple-length encrypted key.</td>
</tr>
<tr>
<td>32–39</td>
<td>X'0000000000000000' if a single-length encrypted key or double-length encrypted key.</td>
</tr>
<tr>
<td>40–47</td>
<td>Reserved (set to binary zeros).</td>
</tr>
<tr>
<td>56–63</td>
<td>Reserved (set to binary zeros).</td>
</tr>
</tbody>
</table>

PKA Key Token Formats

PKA Null Key Token

Table 245 shows the format for a PKA null key token.

Table 245. Format of PKA Null Key Tokens

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X'00' Token identifier (indicates that this is a null key token).</td>
</tr>
<tr>
<td>1</td>
<td>Version, X'00'.</td>
</tr>
<tr>
<td>2–3</td>
<td>Length of the key token structure.</td>
</tr>
<tr>
<td>4–7</td>
<td>Ignored (should be zero).</td>
</tr>
</tbody>
</table>

RSA Key Token Formats

RSA Public Key Token

An RSA public key token contains the following sections:

- A required token header, starting with the token identifier X'1E'  
- A required RSA public key section, starting with the section identifier X'04'

Table 246 presents the format of an RSA public key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address, S/390 format).

Table 246. RSA Public Key Token

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token Header (required)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>001</td>
<td>Token identifier. X'1E' indicates an external token.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Version, X'00'.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the key token structure.</td>
</tr>
<tr>
<td>004</td>
<td>004</td>
<td>Ignored. Should be zero.</td>
</tr>
<tr>
<td>RSA Public Key Section (required)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>001</td>
<td>X'04', section identifier, RSA public key.</td>
</tr>
</tbody>
</table>
### Table 246. RSA Public Key Token (continued)

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>001</td>
<td>X’00’, version.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length, 12+xxx+yyy.</td>
</tr>
<tr>
<td>004</td>
<td>002</td>
<td>Reserved field.</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>RSA public key exponent field length in bytes, “xxx”.</td>
</tr>
<tr>
<td>008</td>
<td>002</td>
<td>Public key modulus length in bits.</td>
</tr>
<tr>
<td>010</td>
<td>002</td>
<td>RSA public key modulus field length in bytes, “yyy”.</td>
</tr>
<tr>
<td>012</td>
<td>xxx</td>
<td>Public key exponent (this is generally a 1-, 3-, or 64- to 512-byte quantity), e. e must be odd and 1&lt;e&lt;n. (Frequently, the value of e is $2^{16} + 1$)</td>
</tr>
<tr>
<td>12+xxx</td>
<td>yyy</td>
<td>Modulus, n.</td>
</tr>
</tbody>
</table>

### RSA Private External Key Token

An RSA private external key token contains the following sections:

- A required PKA token header starting with the token identifier X’1E’
- A required RSA private key section starting with one of the following section identifiers:
  - X’02’ which indicates a modulus-exponent form RSA private key section (not optimized) with modulus length of up to 1024 bits for use with the Cryptographic Coprocessor Feature or the PCI Cryptographic Coprocessor.
  - X’08’ which indicates an optimized Chinese Remainder Theorem form private key section with modulus bit length of up to 4096 bits for use with the PCICC, PCIXCC, CEX2C, or CEX3C.
  - X’09’ which indicates a modulus-exponent form RSA private key section (not optimized) with modulus length of up to 4096 bits for use with the CEX2C or CEX3C.
- A required RSA public key section, starting with the section identifier X’04’
- An optional private key name section, starting with the section identifier X’10’

Table 247 presents the basic record format of an RSA private external key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address, S/390 format). All binary fields (exponents, modulus, and so on) in the private sections of tokens are right-justified and padded with zeros to the left.

### Table 247. RSA Private External Key Token Basic Record Format

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token Header (required)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>001</td>
<td>Token identifier. X’1E’ indicates an external token. The private key is either in cleartext or enciphered with a transport key-encrypting key.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Version, X’00’.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the key token structure.</td>
</tr>
<tr>
<td>004</td>
<td>004</td>
<td>Ignored. Should be zero.</td>
</tr>
</tbody>
</table>
Table 247. RSA Private External Key Token Basic Record Format (continued)

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RSA Private Key Section (required)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• For 1024-bit Modulus-Exponent form refer to [RSA Private Key Token, 1024-bit Modulus-Exponent External Form](on page 605)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• For 4096-bit Modulus-Exponent form refer to [RSA Private Key Token, 4096-bit Modulus-Exponent External Form](on page 606)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• For 4096-bit Chinese Remainder Theorem form refer to [RSA Private Key Token, 4096-bit Chinese Remainder Theorem External Form](on page 606)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RSA Public Key Section (required)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000 001</td>
<td>X'04', section identifier, RSA public key.</td>
<td></td>
</tr>
<tr>
<td>001 001</td>
<td>X'00', version.</td>
<td></td>
</tr>
<tr>
<td>002 002</td>
<td>Section length, 12+xxx.</td>
<td></td>
</tr>
<tr>
<td>004 002</td>
<td>Reserved field.</td>
<td></td>
</tr>
<tr>
<td>006 002</td>
<td>RSA public key exponent field length in bytes, “xxx”.</td>
<td></td>
</tr>
<tr>
<td>008 002</td>
<td>Public key modulus length in bits.</td>
<td></td>
</tr>
<tr>
<td>010 002</td>
<td>RSA public key modulus field length in bytes, which is zero for a private token.</td>
<td>Note: In an RSA private key token, this field should be zero. The RSA private key section contains the modulus.</td>
</tr>
<tr>
<td>012 xxx</td>
<td>Public key exponent, e (this is generally a 1-, 3-, or 64- to 512-byte quantity). e must be odd and 1&lt;e&lt;n. (Frequently, the value of e is 2^16+1 (=65,537).)</td>
<td></td>
</tr>
<tr>
<td><strong>Private Key Name (optional)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000 001</td>
<td>X'10', section identifier, private key name.</td>
<td></td>
</tr>
<tr>
<td>001 001</td>
<td>X'00', version.</td>
<td></td>
</tr>
<tr>
<td>002 002</td>
<td>Section length, X'0044' (68 decimal).</td>
<td></td>
</tr>
<tr>
<td>004 064</td>
<td>Private key name (in ASCII), left-justified, padded with space characters (X'20'). An access control system can use the private key name to verify that the calling application is entitled to use the key.</td>
<td></td>
</tr>
</tbody>
</table>

**RSA Private Key Token, 1024-bit Modulus-Exponent External Form:** This RSA private key token and the external X'02' token is supported on the Cryptographic Coprocessor Feature and PCI Cryptographic Coprocessor.

Table 248. RSA Private Key Token, 1024-bit Modulus-Exponent External Format

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000 001</td>
<td>X'02', section identifier, RSA private key, modulus-exponent format (RSA-PRIV)</td>
<td></td>
</tr>
<tr>
<td>001 001</td>
<td>X'00', version.</td>
<td></td>
</tr>
<tr>
<td>002 002</td>
<td>Length of the RSA private key section X'016C' (364 decimal).</td>
<td></td>
</tr>
<tr>
<td>004 020</td>
<td>SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.</td>
<td></td>
</tr>
<tr>
<td>024 004</td>
<td>Reserved; set to binary zero.</td>
<td></td>
</tr>
<tr>
<td>028 001</td>
<td>Key format and security: X'00' Unencrypted RSA private key subsection identifier. X'82' Encrypted RSA private key subsection identifier.</td>
<td></td>
</tr>
<tr>
<td>029 001</td>
<td>Reserved, binary zero.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 248. RSA Private Key Token, 1024-bit Modulus-Exponent External Format (continued)

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>030</td>
<td>020</td>
<td>SHA-1 hash of the optional key-name section. If there is no key-name section, then 20 bytes of X'00'.</td>
</tr>
<tr>
<td>050</td>
<td>004</td>
<td>Key use flag bits.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Bit</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All other bits reserved, set to binary zero.</td>
</tr>
<tr>
<td>054</td>
<td>006</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>060</td>
<td>024</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>084</td>
<td></td>
<td>Start of the optionally-encrypted secure subsection.</td>
</tr>
<tr>
<td>084</td>
<td>024</td>
<td>Random number, confounder.</td>
</tr>
<tr>
<td>108</td>
<td>128</td>
<td>Private-key exponent, d. d=e^{-1} mod((p-1)(q-1)), and 1&lt;d&lt;n where e is the public exponent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>End of the optionally-encrypted subsection; the confounder field and the private-key exponent field are enciphered for key confidentiality when the key format and security flags (offset 28) indicate that the private key is enciphered. They are enciphered under a double-length transport key using the ede2 algorithm.</td>
</tr>
<tr>
<td>236</td>
<td>128</td>
<td>Modulus, n. n=pq where p and q are prime and 1&lt;n&lt;2^{1024}.</td>
</tr>
</tbody>
</table>

### RSA Private Key Token, 4096-bit Modulus-Exponent External Form:

This RSA private key token and the external X'09' token is supported on the Crypto Express2 Coprocessor and Crypto Express3 Coprocessor.

### Table 249. RSA Private Key Token, 4096-bit Modulus-Exponent External Format

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>X'09', section identifier, RSA private key, modulus-exponent format (RSAMEVAR).</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X'00', version.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>X'00' hash value of the private key section 132+ddd+nnn+xxx.</td>
</tr>
<tr>
<td>004</td>
<td>020</td>
<td>SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.</td>
</tr>
<tr>
<td>024</td>
<td>002</td>
<td>Length of the encrypted private key section 8+ddd+xxx.</td>
</tr>
<tr>
<td>026</td>
<td>002</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>028</td>
<td>001</td>
<td>Key format and security:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'82'</td>
</tr>
<tr>
<td>029</td>
<td>001</td>
<td>Reserved, set to binary zero.</td>
</tr>
<tr>
<td>030</td>
<td>020</td>
<td>SHA-1 hash of the optional key-name section. If there is no key-name section, then 20 bytes of X'00'.</td>
</tr>
</tbody>
</table>
### Table 249. RSA Private Key Token, 4096-bit Modulus-Exponent External Format (continued)

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>050</td>
<td>001</td>
<td>Key use flag bits.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit Meaning When Set On</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>Key management usage permitted.</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Signature usage not permitted.</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>The key is translatable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All other bits reserved, set to binary zero.</td>
</tr>
<tr>
<td>051</td>
<td>001</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>052</td>
<td>048</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>100</td>
<td>016</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>116</td>
<td>002</td>
<td>Length of private exponent, d, in bytes: ddd.</td>
</tr>
<tr>
<td>118</td>
<td>002</td>
<td>Length of modulus, n, in bytes: nnn.</td>
</tr>
<tr>
<td>120</td>
<td>002</td>
<td>Length of padding field, in bytes: xxx.</td>
</tr>
<tr>
<td>122</td>
<td>002</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>124</td>
<td></td>
<td>Start of the optionally-encrypted secure subsection.</td>
</tr>
<tr>
<td>124</td>
<td>008</td>
<td>Random number, confounder.</td>
</tr>
<tr>
<td>132</td>
<td>ddd</td>
<td>Private-key exponent, d, (d = e^{-1} \mod((p-1)(q-1))), and (1 &lt; d &lt; n) where e is the public exponent.</td>
</tr>
<tr>
<td>132+ddd</td>
<td>xxx</td>
<td>X’00’ padding of length xxx bytes such that the length from the start of the random number above to the end of the padding field is a multiple of eight bytes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>End of the optionally-encrypted subsection; the confounder field and the private-key exponent field are enciphered for key confidentiality when the key format and security flags (offset 28) indicate that the private key is enciphered. They are enciphered under a double-length transport key using the ede2 algorithm.</td>
</tr>
<tr>
<td>132+ddd+xxx</td>
<td>nnn</td>
<td>Modulus, n, (n = pq) where p and q are prime and (1 &lt; n &lt; 2^{4096}).</td>
</tr>
</tbody>
</table>

### RSA Private Key Token, 4096-bit Chinese Remainder Theorem External Form:
This RSA private key token (up to 2048-bit modulus) is supported on the PCICC, PCIXCC, CEX2C, or CEX3C. The 4096-bit modulus private key token is supported on the z9 EC, z9 BC, z10 EC and z10 BC with the Nov. 2007 or later version of the licensed internal code installed on the CEX2C or CEX3C.

### Table 250. RSA Private Key Token, 4096-bit Chinese Remainder Theorem External Format

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>X’08’, section identifier, RSA private key, CRT format (RSA-CRT)</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X’00’, version.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the RSA private-key section, 132 + ppp + qqq + rrr + sss + uuu + xxx + nnn.</td>
</tr>
<tr>
<td>004</td>
<td>020</td>
<td>SHA-1 hash value of the private key subsection cleartext, offset 28 to the end of the modulus.</td>
</tr>
<tr>
<td>024</td>
<td>004</td>
<td>Reserved; set to binary zero.</td>
</tr>
</tbody>
</table>
Table 250. RSA Private Key Token, 4096-bit Chinese Remainder Theorem External Format (continued)

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
</table>
| 028         | 001            | Key format and security:  
X’40’ Unencrypted RSA private-key subsection identifier, Chinese Remainder form.  
X’42’ Encrypted RSA private-key subsection identifier, Chinese Remainder form. |
| 029         | 001            | Reserved; set to binary zero. |
| 030         | 020            | SHA-1 hash of the optional key-name section and any following optional sections. If there are no optional sections, then 20 bytes of X’00’. |
| 050         | 004            | Key use flag bits.  
**Bit** | **Meaning When Set On** |
| 0           | Key management usage permitted.  
1          | Signature usage not permitted.  
6           | The key is translatable. |
|             | All other bits reserved, set to binary zero. |
| 054         | 002            | Length of prime number, p, in bytes: ppp. |
| 056         | 002            | Length of prime number, q, in bytes: qqq. |
| 058         | 002            | Length of dp, in bytes: rrr. |
| 060         | 002            | Length of d q, in bytes: sss. |
| 062         | 002            | Length of U, in bytes: uuu. |
| 064         | 002            | Length of modulus, n, in bytes: nnn. |
| 066         | 004            | Reserved; set to binary zero. |
| 070         | 002            | Length of padding field, in bytes: xxx. |
| 072         | 004            | Reserved, set to binary zero. |
| 076         | 016            | Reserved, set to binary zero. |
| 092         | 032            | Reserved; set to binary zero. |
| 124         |                | Start of the optionally-encrypted secure subsection. |
| 124         | 008            | Random number, confounder. |
| 132         | ppp            | Prime number, p. |
| 132 + ppp   | qqq            | Prime number, q. |
| 132 + ppp + qqq | rrr | \(d_q = d \mod(p - 1)\) |
| 132 + ppp + qqq + rrr | sss | \(d_u = d \mod(q - 1)\) |
| 132 + ppp + qqq + rrr + sss | uuu | \(U = q^{-1} \mod(p)\) |
| 132 + ppp + qqq + rrr + sss + uuu | xxx | X’00’ padding of length xxx bytes such that the length from the start of the random number above to the end of the padding field is a multiple of eight bytes. |
| 132 + ppp + qqq + rrr + sss + uuu + xxx | nnn | Modulus, n. \(n = pq\) where p and q are prime and \(1 < n < 2^{4096}\). |

End of the optionally-encrypted secure subsection; all of the fields starting with the confounder field and ending with the variable length pad field are enciphered for key confidentiality when the key format-and-security flags (offset 28) indicate that the private key is enciphered. They are enciphered under a double-length transport key using the TDES (CBC outer chaining) algorithm.
RSA Private Internal Key Token
An RSA private internal key token contains the following sections:
- A required PKA token header, starting with the token identifier X'1F'
- Basic record format of an RSA private internal key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address, S/390 format). All binary fields (exponents, modulus, and so on) in the private sections of tokens are right-justified and padded with zeros to the left.

Table 251. RSA Private Internal Key Token Basic Record Format

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Token Header (required)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>001</td>
<td>Token identifier. X'1F' indicates an internal token. The private key is enciphered with a PKA master key.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Version, X'00'.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the key token structure excluding the internal information section.</td>
</tr>
<tr>
<td>004</td>
<td>004</td>
<td>Ignored; should be zero.</td>
</tr>
</tbody>
</table>

**RSA Private Key Section and Secured Subsection (required)**
- For 1024-bit X'02' Modulus-Exponent form refer to "RSA Private Key Token, 1024-bit Modulus-Exponent Internal Form for Cryptographic Coprocessor Feature" on page 609
- For 1024-bit X'06' Modulus-Exponent form refer to "RSA Private Key Token, 1024-bit Modulus-Exponent Internal Form for PCICC, PCIXCC, CEX2C, or CEX3C" on page 610
- For 4096-bit X'08' Chinese Remainder Theorem form refer to "RSA Private Key Token, 4096-bit Chinese Remainder Theorem Internal Form" on page 611

**RSA Public Key Section (required)**
<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>X'04', section identifier, RSA public key.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X'00', version.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length, 12+xxx.</td>
</tr>
<tr>
<td>004</td>
<td>002</td>
<td>Reserved field.</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>RSA public key exponent field length in bytes, “xxx”.</td>
</tr>
<tr>
<td>008</td>
<td>002</td>
<td>Public key modulus length in bits.</td>
</tr>
<tr>
<td>010</td>
<td>002</td>
<td>RSA public key modulus field length in bytes, which is zero for a private token.</td>
</tr>
<tr>
<td>012</td>
<td>xxx</td>
<td>Public key exponent (this is generally a 1, 3, or 64 to 512 byte quantity), e. e must be odd and 1&lt;e&lt;n. (Frequently, the value of e is 2**16+1 (65,537).</td>
</tr>
</tbody>
</table>

**Private Key Name (optional)**
<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>X'10', section identifier, private key name.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X'00', version.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length, X'0044' (68 decimal).</td>
</tr>
<tr>
<td>004</td>
<td>064</td>
<td>Private key name (in ASCII), left-justified, padded with space characters (X'20'). An access control system can use the private key name to verify that the calling application is entitled to use the key.</td>
</tr>
</tbody>
</table>

**Internal Information Section (required)**
<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>004</td>
<td>Eye catcher 'PKTN'.</td>
</tr>
</tbody>
</table>
**Table 251. RSA Private Internal Key Token Basic Record Format (continued)**

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>004</td>
<td>004</td>
<td>PKA token type.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Bit</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>008</td>
<td>004</td>
<td>Address of token header.</td>
</tr>
<tr>
<td>012</td>
<td>002</td>
<td>Total length of total structure including this information section.</td>
</tr>
<tr>
<td>014</td>
<td>002</td>
<td>Count of number of sections.</td>
</tr>
<tr>
<td>016</td>
<td>016</td>
<td>PKA master key hash pattern.</td>
</tr>
<tr>
<td>032</td>
<td>001</td>
<td>Domain of retained key.</td>
</tr>
<tr>
<td>033</td>
<td>008</td>
<td>Serial number of processor holding retained key.</td>
</tr>
<tr>
<td>041</td>
<td>007</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

**RSA Private Key Token, 1024-bit Modulus-Exponent Internal Form for Cryptographic Coprocessor Feature:**

**Table 252. RSA Private Internal Key Token, 1024-bit ME Form for Cryptographic Coprocessor Feature**

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>X'02', section identifier, RSA private key.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X'00', version.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the RSA private key section X'016C' (364 decimal).</td>
</tr>
<tr>
<td>004</td>
<td>020</td>
<td>SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.</td>
</tr>
<tr>
<td>024</td>
<td>004</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>028</td>
<td>001</td>
<td>Key format and security: X'02' RSA private key.</td>
</tr>
<tr>
<td>029</td>
<td>001</td>
<td>Format of external key from which this token was derived: X'21' External private key was specified in the clear. X'22' External private key was encrypted.</td>
</tr>
<tr>
<td>030</td>
<td>020</td>
<td>SHA-1 hash of the key token structure contents that follow the public key section. If no sections follow, this field is set to binary zeros.</td>
</tr>
<tr>
<td>050</td>
<td>001</td>
<td>Key use flag bits.</td>
</tr>
</tbody>
</table>

**Bit** | **Meaning When Set On** |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Key management usage permitted.</td>
</tr>
<tr>
<td>1</td>
<td>Signature usage not permitted.</td>
</tr>
</tbody>
</table>

All other bits reserved, set to binary zero.
### Table 252. RSA Private Internal Key Token, 1024-bit ME Form for Cryptographic Coprocessor Feature (continued)

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>005</td>
<td>009</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>051</td>
<td>009</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>060</td>
<td>048</td>
<td>Object Protection Key (OPK) encrypted under a PKA master key—can be under the Signature Master Key (SMK) or Key Management Master Key (KMMK) depending on key use.</td>
</tr>
<tr>
<td>108</td>
<td>128</td>
<td>Secret key exponent d, encrypted under the OPK. d=e(^{-1}) mod((p-1)(q-1))</td>
</tr>
<tr>
<td>236</td>
<td>128</td>
<td>Modulus, n. n=pq where p and q are prime and 1&lt;n&lt;2(^{1024}).</td>
</tr>
</tbody>
</table>

### RSA Private Key Token, 1024-bit Modulus-Exponent Internal Form for PCICC, PCIXCC, CEX2C, or CEX3C

### Table 253. RSA Private Internal Key Token, 1024-bit ME Form for PCICC, PCIXCC, CEX2C, or CEX3C

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>X’06’, section identifier, RSA private key modulus-exponent format (RSA-PRIV).</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X’00’, version.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the RSA private key section X’0198’ (408 decimal) + rrr + iii + xxx.</td>
</tr>
<tr>
<td>004</td>
<td>020</td>
<td>SHA-1 hash value of the private key subsection cleartext, offset 28 to and including the modulus at offset 236.</td>
</tr>
<tr>
<td>024</td>
<td>004</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>028</td>
<td>001</td>
<td>Key format and security: X’02‘ RSA private key.</td>
</tr>
<tr>
<td>029</td>
<td>001</td>
<td>Format of external key from which this token was derived: X’21‘ External private key was specified in the clear. X’22‘ External private key was encrypted. X’23‘ Private key was generated using regeneration data. X’24‘ Private key was randomly generated.</td>
</tr>
<tr>
<td>030</td>
<td>020</td>
<td>SHA-1 hash of the optional key-name section and any following optional sections. If there are no optional sections, this field is set to binary zeros.</td>
</tr>
<tr>
<td>050</td>
<td>004</td>
<td>Key use flag bits.</td>
</tr>
<tr>
<td>Bit</td>
<td>Meaning</td>
<td>When Set On</td>
</tr>
<tr>
<td>0</td>
<td>Key management usage permitted.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Signature usage not permitted.</td>
<td></td>
</tr>
<tr>
<td>All other bits reserved, set to binary zeros.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>054</td>
<td>006</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>060</td>
<td>048</td>
<td>Object Protection Key (OPK) encrypted under the Asymmetric Keys Master Key using the ede3 algorithm.</td>
</tr>
<tr>
<td>108</td>
<td>128</td>
<td>Private key exponent d, encrypted under the OPK using the ede5 algorithm. d=e(^{-1}) mod((p-1)(q-1)), and 1&lt;d&lt;n where e is the public exponent.</td>
</tr>
<tr>
<td>236</td>
<td>128</td>
<td>Modulus, n. n=pq where p and q are prime and 2(^{512})&lt;n&lt;2(^{1024}).</td>
</tr>
<tr>
<td>364</td>
<td>016</td>
<td>Asymmetric-Keys Master Key hash pattern.</td>
</tr>
<tr>
<td>380</td>
<td>020</td>
<td>SHA-1 hash value of the blinding information subsection cleartext, offset 400 to the end of the section.</td>
</tr>
<tr>
<td>400</td>
<td>002</td>
<td>Length of the random number r, in bytes: rrr.</td>
</tr>
</tbody>
</table>
Table 253. RSA Private Internal Key Token, 1024-bit ME Form for PCICC, PCIXCC, CEX2C, or CEX3C (continued)

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>402</td>
<td>002</td>
<td>Length of the random number r−1, in bytes: iii.</td>
</tr>
<tr>
<td>404</td>
<td>002</td>
<td>Length of the padding field, in bytes: xxx.</td>
</tr>
<tr>
<td>406</td>
<td>002</td>
<td>Reserved; set to binary zeros.</td>
</tr>
<tr>
<td>408</td>
<td></td>
<td>Start of the encrypted blinding subsection</td>
</tr>
<tr>
<td>408</td>
<td>rrr</td>
<td>Random number r (used in blinding).</td>
</tr>
<tr>
<td>408 + rrr</td>
<td>iii</td>
<td>Random number r−1 (used in blinding).</td>
</tr>
<tr>
<td>408 + rrr + iii</td>
<td>xxx</td>
<td>X’00’ padding of length xxx bytes such that the length from the start of the encrypted blinding subsection to the end of the padding field is a multiple of eight bytes.</td>
</tr>
</tbody>
</table>

End of the encrypted blinding subsection; all of the fields starting with the random number r and ending with the variable length pad field are encrypted under the OPK using TDES (CBC outer chaining) algorithm.

RSA Private Key Token, 4096-bit Chinese Remainder Theorem Internal Form:
This RSA private key token (up to 2048-bit modulus) is supported on the PCICC, PCIXCC, CEX2C, or CEX3C. The 4096-bit modulus private key token is supported on the z9 EC, z9 BC, z10 EC and z10 BC with the Nov. 2007 or later version of the licensed internal code installed on the CEX2C or CEX3C.

Table 254. RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>X’08’, section identifier, RSA private key, CRT format (RSA-CRT)</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X’00’, version.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the RSA private-key section, 132 + ppp + qqq + rrr + sss + uuu + ttt + iii + xxx + nnn.</td>
</tr>
<tr>
<td>004</td>
<td>020</td>
<td>SHA-1 hash value of the private-key subsection cleartext, offset 28 to the end of the modulus.</td>
</tr>
<tr>
<td>024</td>
<td>004</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>028</td>
<td>001</td>
<td>Key format and security: X’08’ Encrypted RSA private-key subsection identifier, Chinese Remainder form.</td>
</tr>
<tr>
<td>029</td>
<td>001</td>
<td>Key derivation method: X’21’ External private key was specified in the clear. X’22’ External private key was encrypted. X’23’ Private key was generated using regeneration data. X’24’ Private key was randomly generated.</td>
</tr>
<tr>
<td>030</td>
<td>020</td>
<td>SHA-1 hash of the optional key-name section and any following sections. If there are no optional sections, then 20 bytes of X’00’.</td>
</tr>
<tr>
<td>050</td>
<td>004</td>
<td>Key use flag bits: Bit Meaning When Set On 0 Key management usage permitted. 1 Signature usage not permitted. All other bits reserved, set to binary zero.</td>
</tr>
<tr>
<td>054</td>
<td>002</td>
<td>Length of prime number, p, in bytes: ppp.</td>
</tr>
<tr>
<td>056</td>
<td>002</td>
<td>Length of prime number, q, in bytes: qqq.</td>
</tr>
</tbody>
</table>
### Table 254. RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format (continued)

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>058</td>
<td>002</td>
<td>Length of (d_p), in bytes: (rrr).</td>
</tr>
<tr>
<td>060</td>
<td>002</td>
<td>Length of (d_q), in bytes: (sss).</td>
</tr>
<tr>
<td>062</td>
<td>002</td>
<td>Length of (U), in bytes: (uuu).</td>
</tr>
<tr>
<td>064</td>
<td>002</td>
<td>Length of modulus, (n), in bytes: (nnn).</td>
</tr>
<tr>
<td>066</td>
<td>002</td>
<td>Length of the random number (r), in bytes: (ttt).</td>
</tr>
<tr>
<td>068</td>
<td>002</td>
<td>Length of the random number (r^{-1}), in bytes: (iii).</td>
</tr>
<tr>
<td>070</td>
<td>002</td>
<td>Length of padding field, in bytes: (xxx).</td>
</tr>
<tr>
<td>072</td>
<td>004</td>
<td>Reserved, set to binary zero.</td>
</tr>
<tr>
<td>076</td>
<td>016</td>
<td>Asymmetric-Keys Master Key hash pattern.</td>
</tr>
<tr>
<td>092</td>
<td>032</td>
<td>Object Protection Key (OPK) encrypted under the Asymmetric-Keys Master Key using the TDES (CBC outer chaining) algorithm.</td>
</tr>
<tr>
<td>124</td>
<td></td>
<td>Start of the encrypted secure subsection, encrypted under the OPK using TDES (CBC outer chaining).</td>
</tr>
<tr>
<td>124</td>
<td>008</td>
<td>Random number, confounder.</td>
</tr>
<tr>
<td>132</td>
<td>(ppp)</td>
<td>Prime number, (p).</td>
</tr>
<tr>
<td>132</td>
<td>(qqq)</td>
<td>Prime number, (q).</td>
</tr>
<tr>
<td>132 + (ppp) + (qqq)</td>
<td>(rrr)</td>
<td>(d_p = d \mod(p - 1))</td>
</tr>
<tr>
<td>132 + (ppp) + (qqq)  + (rrr)</td>
<td>(sss)</td>
<td>(d_q = d \mod(q - 1))</td>
</tr>
<tr>
<td>132 + (ppp) + (qqq)  + (rrr)  + (sss) + (uuu) + (ttt)</td>
<td>(uuu)</td>
<td>(U = q^{-1}\mod(p)).</td>
</tr>
<tr>
<td>132 + (ppp) + (qqq)  + (rrr)  + (sss) + (uuu) + (ttt)</td>
<td>(ttt)</td>
<td>Random number (r) (used in blinding).</td>
</tr>
<tr>
<td>132 + (ppp) + (qqq)  + (rrr)  + (sss) + (uuu) + (ttt) + (iii)</td>
<td>(iii)</td>
<td>Random number (r^{-1}) (used in blinding).</td>
</tr>
<tr>
<td>132 + (ppp) + (qqq)  + (rrr)  + (sss) + (uuu) + (ttt) + (iii) + (xxx)</td>
<td>(xxx)</td>
<td>(X'00') padding of length (xxx) bytes such that the length from the start of the confounder at offset 124 to the end of the padding field is a multiple of eight bytes.</td>
</tr>
<tr>
<td>132 + (ppp) + (qqq)  + (rrr)  + (sss) + (uuu) + (ttt) + (iii) + (xxx)</td>
<td>(nnn)</td>
<td>Modulus, (n = pq) where (p) and (q) are prime and (1 &lt; n &lt; 2^{4096}).</td>
</tr>
</tbody>
</table>

### DSS Key Token Formats

**DSS Public Key Token**

A DSS public key token contains the following sections:

- A required token header, starting with the token identifier X'1E'
- A required DSS public key section, starting with the section identifier X'03'

Table 255 on page 613 presents the format of a DSS public key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address, S/390 format).
### Table 255. DSS Public Key Token

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Token Header (required)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>001</td>
<td>Token identifier. X'1E' indicates an external token.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Version, X'00'.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the key token structure.</td>
</tr>
<tr>
<td>004</td>
<td>004</td>
<td>Ignored. Should be zero.</td>
</tr>
<tr>
<td><strong>DSS Public Key Section (required)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>001</td>
<td>X'03', section identifier, DSS public key.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X'00', version.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length, 14+ppp+qqq+ggg+yyy.</td>
</tr>
<tr>
<td>004</td>
<td>002</td>
<td>Size of p in bits. The size of p must be one of: 512, 576, 640, 704, 768, 832, 896, 960, or 1024.</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>Size of the p field in bytes, &quot;ppp&quot;.</td>
</tr>
<tr>
<td>008</td>
<td>002</td>
<td>Size of the q field in bytes, &quot;qqq&quot;.</td>
</tr>
<tr>
<td>010</td>
<td>002</td>
<td>Size of the g field in bytes, &quot;ggg&quot;.</td>
</tr>
<tr>
<td>012</td>
<td>002</td>
<td>Size of the y field in bytes, &quot;yyy&quot;.</td>
</tr>
<tr>
<td>014</td>
<td>ppp</td>
<td>Prime modulus (large public modulus), p.</td>
</tr>
<tr>
<td>014 +ppp</td>
<td>qqq</td>
<td>Prime divisor (small public modulus), q. 2^{159}&lt;q&lt;2^{160}.</td>
</tr>
<tr>
<td>014 +ppp +qqq</td>
<td>ggg</td>
<td>Public key generator, g.</td>
</tr>
<tr>
<td>014 +ppp +qqq</td>
<td>yyy</td>
<td>Public key, y. y=g^x mod(p); 1&lt;y&lt;p.</td>
</tr>
</tbody>
</table>

### DSS Private External Key Token

A DSS private external key token contains the following sections:
- A required PKA token header, starting with the token identifier X'1E'
- A required DSS private key section, starting with the section identifier X'01'
- A required DSS public key section, starting with the section identifier X'03'
- An optional private key name section, starting with the section identifier X'10'

Table 256 presents the format of a DSS private external key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address, S/390 format). All binary fields (exponents, modulus, and so on) in the private sections of tokens are right-justified and padded with zeros to the left.

### Table 256. DSS Private External Key Token

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Token Header (required)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>001</td>
<td>Token identifier. X'1E' indicates an external token. The private key is enciphered with a PKA master key.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Version, X'00'.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the key token structure.</td>
</tr>
<tr>
<td>004</td>
<td>004</td>
<td>Ignored. Should be zero.</td>
</tr>
<tr>
<td><strong>DSS Private Key Section and Secured Subsection (required)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>001</td>
<td>X'01', section identifier, DSS private key.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X'00', version.</td>
</tr>
</tbody>
</table>
Table 256. DSS Private External Key Token (continued)

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the DSS private key section, 436, X'01B4'.</td>
</tr>
<tr>
<td>004</td>
<td>020</td>
<td>SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.</td>
</tr>
<tr>
<td>024</td>
<td>004</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>028</td>
<td>001</td>
<td>Key security: X'00' Unencrypted DSS private key subsection identifier. X'81' Encrypted DSS private key subsection identifier.</td>
</tr>
<tr>
<td>029</td>
<td>001</td>
<td>Padding, X'00'.</td>
</tr>
<tr>
<td>030</td>
<td>020</td>
<td>SHA-1 hash of the key token structure contents that follow the public key section. If no sections follow, this field is set to binary zeros.</td>
</tr>
<tr>
<td>050</td>
<td>010</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>060</td>
<td>048</td>
<td>Ignored; set to binary zero.</td>
</tr>
<tr>
<td>080</td>
<td>128</td>
<td>Public key generator, g. 1&lt;g&lt;p.</td>
</tr>
<tr>
<td>236</td>
<td>128</td>
<td>Prime modulus (large public modulus), p. 2&lt;sup&gt;159&lt;/sup&gt;&lt;p&lt;2&lt;sup&gt;160&lt;/sup&gt; and L (the modulus length) must be a multiple of 64.</td>
</tr>
<tr>
<td>364</td>
<td>020</td>
<td>Prime divisor (small public modulus), q. 2&lt;sup&gt;160&lt;/sup&gt;&lt;q&lt;2&lt;sup&gt;160&lt;/sup&gt;.</td>
</tr>
<tr>
<td>384</td>
<td>004</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>388</td>
<td>024</td>
<td>Random number, confounder. <strong>Note:</strong> This field and the next two fields are enciphered for key confidentiality when the key security flag (offset 28) indicates the private key is enciphered.</td>
</tr>
<tr>
<td>412</td>
<td>020</td>
<td>Secret DSS key, x; x is random. (See the preceding note.)</td>
</tr>
<tr>
<td>432</td>
<td>004</td>
<td>Random number, generated when the secret key is generated. (See the preceding note.)</td>
</tr>
</tbody>
</table>

**DSS Public Key Section (required)**

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>X'03', section identifier, DSS public key.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X'00', version.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length, 14+yyy.</td>
</tr>
<tr>
<td>004</td>
<td>002</td>
<td>Size of p in bits. The size of p must be one of: 512, 576, 640, 704, 768, 832, 896, 960, or 1024.</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>Size of the p field in bytes, which is zero for a private token.</td>
</tr>
<tr>
<td>008</td>
<td>002</td>
<td>Size of the q field in bytes, which is zero for a private token.</td>
</tr>
<tr>
<td>010</td>
<td>002</td>
<td>Size of the g field in bytes, which is zero for a private token.</td>
</tr>
<tr>
<td>012</td>
<td>002</td>
<td>Size of the y field in bytes, “yyy”.</td>
</tr>
<tr>
<td>014</td>
<td>yyy</td>
<td>Public key, y. y=g&lt;sup&gt;x&lt;/sup&gt; mod(p) <strong>Note:</strong> p, q, and y are defined in the DSS public key token.</td>
</tr>
</tbody>
</table>

**Private Key Name (optional)**

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>X'10', section identifier, private key. name</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X'00', version.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length, X'0044' (68 decimal).</td>
</tr>
<tr>
<td>004</td>
<td>064</td>
<td>Private key name (in ASCII), left-justified, padded with space characters (X'20'). An access control system can use the private key name to verify that the calling application is entitled to use the key.</td>
</tr>
</tbody>
</table>
DSS Private Internal Key Token

A DSS private internal key token contains the following sections:
- A required PKA token header, starting with the token identifier X'1F'
- A required DSS private key section, starting with the section identifier X'01'
- A required DSS public key section, starting with the section identifier X'03'
- An optional private key name section, starting with the section identifier X'10'
- A required internal information section, starting with the eyecatcher 'PKTN'

Table 257 presents the format of a DSS private internal token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address, S/390 format). All binary fields (exponents, modulus, and so on) in the private sections of tokens are right-justified and padded with zeros to the left.

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Token Header (required)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>001</td>
<td>Token identifier. X'1F' indicates an internal token. The private key is enciphered with a PKA master key.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Version, X'00'.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the key token structure excluding the internal information section.</td>
</tr>
<tr>
<td>004</td>
<td>004</td>
<td>Ignored; should be zero.</td>
</tr>
<tr>
<td><strong>DSS Private Key Section and Secured Subsection (required)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>001</td>
<td>X'01', section identifier, DSS private key.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X'00', version.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the DSS private key section, 436, X'01B4'.</td>
</tr>
<tr>
<td>004</td>
<td>020</td>
<td>SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.</td>
</tr>
<tr>
<td>024</td>
<td>004</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>028</td>
<td>001</td>
<td>Key security: X'01' DSS private key.</td>
</tr>
<tr>
<td>029</td>
<td>001</td>
<td>Format of external key token: X'10' Private key generated on an ICSF host. X'11' External private key was specified in the clear. X'12' External private key was encrypted.</td>
</tr>
<tr>
<td>030</td>
<td>020</td>
<td>SHA-1 hash of the key token structure contents that follow the public key section. If no sections follow, this field is set to binary zeros.</td>
</tr>
<tr>
<td>050</td>
<td>010</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>060</td>
<td>048</td>
<td>The OPK encrypted under a PKA master key (Signature Master Key (SMK)).</td>
</tr>
<tr>
<td>108</td>
<td>128</td>
<td>Public key generator, g. 1&lt;g&lt;p.</td>
</tr>
<tr>
<td>236</td>
<td>128</td>
<td>Prime modulus (large public modulus), p. $2^{L-1}&lt;p&lt;2^L$ for $512\leq L\leq 1024$, and L (the modulus length) must be a multiple of 64.</td>
</tr>
<tr>
<td>364</td>
<td>020</td>
<td>Prime divisor (small public modulus), q. $2^{159}&lt;q&lt;2^{160}$.</td>
</tr>
<tr>
<td>384</td>
<td>004</td>
<td>Reserved; set to binary zero.</td>
</tr>
<tr>
<td>388</td>
<td>024</td>
<td>Random number, confounder. <strong>Note:</strong> This field and the two that follow are enciphered under the OPK.</td>
</tr>
<tr>
<td>412</td>
<td>020</td>
<td>Secret DSS key, x. x is random. (See the preceding note.)</td>
</tr>
</tbody>
</table>
Table 257. DSS Private Internal Key Token (continued)

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>432</td>
<td>004</td>
<td>Random number, generated when the secret key is generated. (See the preceding note.)</td>
</tr>
</tbody>
</table>

**DSS Public Key Section (required)**

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>X'03', section identifier, DSS public key.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X'00', version.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length, 14+yyy.</td>
</tr>
<tr>
<td>004</td>
<td>002</td>
<td>Size of p in bits. The size of p must be one of: 512, 576, 640, 704, 768, 832, 896, 960, or 1024.</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>Size of the p field in bytes, which is zero for a private token.</td>
</tr>
<tr>
<td>008</td>
<td>002</td>
<td>Size of the q field in bytes, which is zero for a private token.</td>
</tr>
<tr>
<td>010</td>
<td>002</td>
<td>Size of the g field in bytes, which is zero for a private token.</td>
</tr>
<tr>
<td>012</td>
<td>002</td>
<td>Size of the y field in bytes, “yyy”.</td>
</tr>
<tr>
<td>014</td>
<td>yyy</td>
<td>Public key, y. y=g^x mod(p);</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> p, g, and y are defined in the DSS public key token.</td>
</tr>
</tbody>
</table>

**Private Key Name (optional)**

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>X'10', section identifier, private key name.</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>X'00', version.</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length, X'0044' (68 decimal).</td>
</tr>
<tr>
<td>004</td>
<td>064</td>
<td>Private key name (in ASCII), left-justified, padded with space characters (X'20'). An access control system can use the private key name to verify that the calling application is entitled to use the key.</td>
</tr>
</tbody>
</table>

**Internal Information Section (required)**

<table>
<thead>
<tr>
<th>Offset (Dec)</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>004</td>
<td>Eye catcher ‘PKTN’.</td>
</tr>
<tr>
<td>004</td>
<td>004</td>
<td>PKA token type.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Bit</strong></td>
</tr>
<tr>
<td>0</td>
<td>RSA key.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>DSS key.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Private key.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Public key.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Private key name section exists.</td>
<td></td>
</tr>
<tr>
<td>008</td>
<td>004</td>
<td>Address of token header.</td>
</tr>
<tr>
<td>012</td>
<td>002</td>
<td>Length of internal work area.</td>
</tr>
<tr>
<td>014</td>
<td>002</td>
<td>Count of number of sections.</td>
</tr>
<tr>
<td>016</td>
<td>016</td>
<td>PKA master key hash pattern.</td>
</tr>
<tr>
<td>032</td>
<td>016</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

**Trusted Block Key Token**

A trusted block key-token (trusted block) is an extension of CCA PKA key tokens using new section identifiers. A trusted block was introduced to CCA beginning with Release 3.25. They are an integral part of a remote key-loading process.
Trusted blocks contain various items, some of which are optional, and some of which can be present in different forms. Tokens are composed of concatenated sections that, unlike CCA PKA key tokens, occur in no prescribed order.

As with other CCA key-tokens, both internal and external forms are defined:

- An external trusted block contains a randomly generated confounder and a triple-length MAC key enciphered under a DES IMP-PKA transport key. The MAC key is used to calculate an ISO 16609 CBC mode TDES MAC of the trusted block contents. An external trusted block is created by the Trusted_Block_Create verb. This verb can:
  1. Create an inactive external trusted block
  2. Change an external trusted block from inactive to active

- An internal trusted block contains a confounder and triple-length MAC key enciphered under a variant of the PKA master key. The MAC key is used to calculate a TDES MAC of the trusted block contents. A PKA master key verification pattern is also included to enable determination that the proper master key is available to process the key. The Remote_Key_Export verb only operates on trusted blocks that are internal. An internal trusted block must be imported from an external trusted block that is active using the PKA_Key_Import verb.

Note: Trusted blocks do not contain a private key section.

**Trusted block sections**

A trusted block is a concatenation of a header followed by an unordered set of sections. The data structures of these sections are summarized in the following table:

<table>
<thead>
<tr>
<th>Section</th>
<th>Reference</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>Table 258 on page 619</td>
<td>Trusted block token header</td>
</tr>
<tr>
<td>X'11'</td>
<td>Table 259 on page 620</td>
<td>Trusted block public key</td>
</tr>
<tr>
<td>X'12'</td>
<td>Table 260 on page 621</td>
<td>Trusted block rule</td>
</tr>
<tr>
<td>X'13'</td>
<td>Table 267 on page 628</td>
<td>Trusted block name (key label)</td>
</tr>
<tr>
<td>X'14'</td>
<td>Table 268 on page 629</td>
<td>Trusted block information</td>
</tr>
<tr>
<td>X'15'</td>
<td>Table 272 on page 632</td>
<td>Trusted block application-defined data</td>
</tr>
</tbody>
</table>

Every trusted block starts with a token header. The first byte of the token header determines the key form:

- An external header (first byte X'1E'), created by the Trusted_Block_Create verb
- An internal header (first byte X'1F'), imported from an active external trusted block by the PKA_Key_Import verb

Following the token header of a trusted block is an unordered set of sections. A trusted block is formed by concatenating these sections to a trusted block header:

- An optional public-key section (trusted block section identifier X'11')
  The trusted block trusted RSA public-key section includes the key itself in addition to a key-usage flag. No multiple sections are allowed.
- An optional rule section (trusted block section identifier X'12')
  A trusted block may have zero or more rule sections.
  1. A trusted block with no rule sections can be used by the PKA_Key_Token_Change and PKA_Key_Import callable services. A trusted
block with no rule sections can also be used by the Digital_Signature_Verify verb, provided there is an RSA public-key section that has its key-usage flag bits set to allow digital signature operations.

2. At least one rule section is required when the Remote_Key_Export verb is used to:
   - Generate an RKX key-token
   - Export an RKX key-token
   - Export a CCA DES key-token
   - Encrypt the clear generated or exported key using the provided vendor certificate

3. If a trusted block has multiple rule sections, each rule section must have a unique 8-character Rule ID.

   • An optional name (key label) section (trusted block section identifier X’13’)
     The trusted block name section provides a 64-byte variable to identify the trusted block, just as key labels are used to identify other CCA keys. This name, or label, enables a host access-control system such as RACF to use the name to verify that the application has authority to use the trusted block. No multiple sections are allowed.

   • A required information section (trusted block section identifier X’14’)
     The trusted block information section contains control and security information related to the trusted block. The information section is required while the others are optional. This section contains the cryptographic information that guarantees its integrity and binds it to the local system. No multiple sections are allowed.

   • An optional application-defined data section (trusted block section identifier X’15’)
     The trusted block application-defined data section can be used to include application-defined data in the trusted block. The purpose of the data in this section is defined by the application. CCA does not examine or use this data in any way. No multiple sections are allowed.

**Trusted block integrity**
An enciphered confounder and triple-length MAC key contained within the required information section of the trusted block is used to protect the integrity of the trusted block. The randomly generated MAC key is used to calculate an ISO 16609 CBC mode TDES MAC of the trusted block contents. Together, the MAC key and MAC value provide a way to verify that the trusted block originated from an authorized source, and binds it to the local system.

An external trusted block has its MAC key enciphered under an IMP-PKA key-encrypting key. An internal trusted block has its MAC key enciphered under a variant of the PKA master key, and the master key verification pattern is stored in the information section.

**Number representation in trusted blocks**

   • All length fields are in binary
   • All binary fields (exponents, lengths, and so forth) are stored with the high-order byte first (left, low-address, z/OS format); thus the least significant bits are to the right and preceded with zero-bits to the width of a field
   • In variable-length binary fields that have an associated field-length value, leading bytes that would otherwise contain X’00’ can be dropped and the field shortened to contain only the significant bits
Format of trusted block sections
At the beginning of every trusted block is a trusted block header. The header contains the following information:

- A token identifier, which specifies if the token contains an external or internal key-token
- A token version number to allow for future changes
- A length in bytes of the trusted block, including the length of the header

The trusted block header is defined in the following table:

Table 258. Trusted block header

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Token identifier (a flag that indicates token type) X'1E' External trusted block token X'1F' Internal trusted block token</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Token version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Length of the key-token structure in bytes.</td>
</tr>
<tr>
<td>004</td>
<td>004</td>
<td>Reserved, binary zero.</td>
</tr>
</tbody>
</table>

Note: See “Number representation in trusted blocks” on page 618.

Following the header, in no particular order, are trusted block sections. There are five different sections defined, each identified by a one-byte section identifier (X'11' - X'15'). Two of the five sections have subsections defined. A subsection is a tag-length-value (TLV) object, identified by a two-byte subsection tag.

Only sections X'12' and X'14' have subsections defined; the other sections do not. A section and its subsections, if any, are one contiguous unit of data. The subsections are concatenated to the related section, but are otherwise in no particular order. Section X'12' has five subsections defined (X'0001' - X'0005'), and section X'14' has two (X'0001' and X'0002'). Of all the subsections, only subsection X'0001' of section X'14' is required. Section X'14' is also required.

The trusted block sections and subsections are described in detail in the following sections.

Trusted block section X'11'

Trusted block section X'11' contains the trusted RSA public key in addition to a key-usage flag indicating whether the public key is usable in key-management operations, digital signature operations, or both.

Section X'11' is optional. No multiple sections are allowed. It has no subsections defined.
This section is defined in the following table:

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Section identifier: X'11' (Trusted block trusted RSA public key)</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Section version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length (16+xxx+yyy).</td>
</tr>
<tr>
<td>004</td>
<td>002</td>
<td>Reserved, must be binary zero.</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>RSA public-key exponent field length in bytes, xxx.</td>
</tr>
<tr>
<td>008</td>
<td>002</td>
<td>RSA public-key modulus length in bits.</td>
</tr>
<tr>
<td>010</td>
<td>002</td>
<td>RSA public-key modulus field length in bytes, yyy.</td>
</tr>
<tr>
<td>012</td>
<td>xxx</td>
<td>Public-key exponent, e (this field length is typically 1, 3, or 64 - 512 bytes), e must be odd and 1≤e&lt;n. (e is frequently valued to 3 or 2(^{16}+1) (=65537), otherwise e is of the same order of magnitude as the modulus). <strong>Note:</strong> Although the current product implementation does not generate such a public key, you can import an RSA public key having an exponent valued to two (2). Such a public key (a Rabin key) can correctly validate an ISO 9796-1 digital signature.</td>
</tr>
<tr>
<td>012+xxx</td>
<td>yyy</td>
<td>RSA public-key modulus, n, n=pq, where p and q are prime and (p^{12}≠2^{512}). The field length is 64 - 512 bytes.</td>
</tr>
<tr>
<td>012+xxx+yyy</td>
<td>004</td>
<td>Flags: X'00000000' (Trusted block public key can be used in digital signature operations only) X'80000000' (Trusted block public key can be used in both digital signature and key management operations) X'C0000000' (Trusted block public key can be used in key management operations only)</td>
</tr>
</tbody>
</table>

**Note:** See "Number representation in trusted blocks" on page 618.

### Trusted block section X'12'

Trusted block section X'12' contains information that defines a rule. A trusted block may have zero or more rule sections.

1. A trusted block with no rule sections can be used by the PKA_Key_Token_Change and PKA_Key_Import callable services. A trusted block with no rule sections can be used by the Digital_Signature_Verify verb, provided there is an RSA public-key section that has its key-usage flag set to allow digital signature operations.

2. At least one rule section is required when the Remote_Key_Export verb is used to:
   - Generate an RKX key-token
   - Export an RKX key-token
   - Export a CCA DES key-token
   - Generate or export a key encrypted by a public key. The public key is contained in a vendor certificate (section X'11'), and is the root certification key for the ATM vendor. It is used to verify the digital signature on public-key certificates for specific individual ATMs.

3. If a trusted block has multiple rule sections, each rule section must have a unique 8-character Rule ID.
Section X'12' is the only section allowed to have multiple sections. Section X'12' is optional. Multiple sections are allowed.

**Note:** The overall length of the trusted block may not exceed its maximum size of 3500 bytes.

Five subsections (TLV objects) are defined.

This section is defined in the following table:

*Table 260. Trusted block rule section (X'12')*

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Section identifier: X'12' Trusted block rule</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Section version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length in bytes (20+yyy).</td>
</tr>
<tr>
<td>004</td>
<td>008</td>
<td>Rule ID (in ASCII). An 8-byte character string that uniquely identifies the rule within the trusted block. Valid ASCII characters are: A..Z, a...z, 0...9, - (hyphen), and _ (underscore), left justified and padded on the right with space characters.</td>
</tr>
<tr>
<td>012</td>
<td>004</td>
<td>Flags (undefined flag bits are reserved and must be zero). X'0000000000' Generate new key X'0000000001' Export existing key</td>
</tr>
<tr>
<td>016</td>
<td>001</td>
<td>Generated key length. Length in bytes of key to be generated when flags value (offset 012) is set to generate a new key; otherwise ignore this value. Valid values are 8, 16, or 24; return an error if not valid.</td>
</tr>
<tr>
<td>017</td>
<td>001</td>
<td>Key-check algorithm identifier (all others are reserved and must not be used): Value Meaning X'00' Do not compute key-check value. In a call to CSNDRKX, set the key_check_length variable to zero. X'01' Encrypt an 8-byte block of binary zeros with the key. In a call to CSNDRKX, set the key_check_length variable to 8. X'02' Compute the MDC-2 hash of the key. In a call to CSNDRKX, set the key_check_length variable to 16.</td>
</tr>
<tr>
<td>018</td>
<td>001</td>
<td>Symmetric encrypted output key format flag (all other values are reserved and must not be used). Return the indicated symmetric key-token using the sym_encrypted_key_identifier parameter. Value Meaning X'00' Return an RKX key-token encrypted under a variant of the MAC key. <strong>Note:</strong> This is the only key format permitted when the flags value (offset 012) is set to generate a new key. X'01' Return a CCA DES key-token encrypted under a transport key. <strong>Note:</strong> This is the only key format permitted when the flags value (offset 012) is set to export an existing key.</td>
</tr>
</tbody>
</table>
Table 260. Trusted block rule section (X'12') (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>019</td>
<td>001</td>
<td>Asymmetric encrypted output key format flag (all other values are reserved and must not be used). Return the indicated asymmetric key-token in the asym_encrypted_key variable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'01'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'02'</td>
</tr>
</tbody>
</table>

020 yyy Rule section subsections (tag-length-value objects). A series of 0 - 5 objects in TLV format.

**Note:** See [Number representation in trusted blocks](#) on page 618.

**Trusted block section X'12' subsections**

Section X'12' has five rule subsections (tag-length-value objects) defined. These subsections are summarized in the following table:

**Table 261. Summary of trusted block rule subsection**

<table>
<thead>
<tr>
<th>Rule subsection tag</th>
<th>TLV object</th>
<th>Optional or required</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0001'</td>
<td>Transport key variant</td>
<td>Optional</td>
<td>Contains variant to be exclusive-ORed into the cleartext transport key.</td>
</tr>
<tr>
<td>X'0002'</td>
<td>Transport key rule reference</td>
<td>Optional; required to use an RKX key-token as a transport key</td>
<td>Contains the rule ID for the rule that must have been used to create the transport key.</td>
</tr>
<tr>
<td>X'0003'</td>
<td>Common export key parameters</td>
<td>Optional; required for key export of an existing key</td>
<td>Contains the export key and source key minimum and maximum lengths, an output key variant length and variant, a CV length, and a CV to be exclusive-ORed with the cleartext transport key to control usage of the key.</td>
</tr>
<tr>
<td>X'0004'</td>
<td>Source key reference</td>
<td>Optional; required if the source key is an RKX key-token</td>
<td>Contains the rule ID for the rule used to create the source key. <strong>Note:</strong> Include all rules that will ever be needed when a trusted block is created. A rule cannot be added to a trusted block after it has been created.</td>
</tr>
<tr>
<td>X'0005'</td>
<td>Export key CCA token parameters</td>
<td>Optional; used for export of CCA DES key tokens only</td>
<td>Contains mask length, mask, and CV template to limit the usage of the exported key. Also contains the template length and template which defines which source key labels are allowed. The key type of a source key input parameter can be &quot;filtered&quot; by using the export key CV limit mask (offset 005) and limit template (offset 005+yyy) in this subsection.</td>
</tr>
</tbody>
</table>

**Note:** See [Number representation in trusted blocks](#) on page 618.

**Trusted block section X'12' subsection X'0001'**
Subsection X'0001' of the trusted block rule section (X'12') is the transport key variant TLV object. This subsection is optional. It contains a variant to be exclusive-ORed into the cleartext transport key.

This subsection is defined in the following table:

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>002</td>
<td>Subsection tag: X'0001' Transport key variant TLV object</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Subsection length in bytes (8+nnn).</td>
</tr>
<tr>
<td>004</td>
<td>001</td>
<td>Subsection version number (X'00').</td>
</tr>
<tr>
<td>005</td>
<td>002</td>
<td>Reserved, must be binary zero.</td>
</tr>
<tr>
<td>007</td>
<td>001</td>
<td>Length of variant field in bytes (nnn). This length must be greater than or equal to the length of the transport key that is identified by the transport_key_identifier parameter. If the variant is longer than the key, truncate it on the right to the length of the key prior to use.</td>
</tr>
<tr>
<td>008</td>
<td>nnn</td>
<td>Transport key variant. Exclusive-OR this variant into the cleartext transport key, provided: (1) the length of the variant field value (offset 007) is not zero, and (2) the symmetric encrypted output key format flag (offset 018 in section X'12') is X'01'.</td>
</tr>
</tbody>
</table>

**Note:** A transport key is not used when the symmetric encrypted output key is in RKX key-token format.

**Note:** See "Number representation in trusted blocks" on page 618.

**Trusted block section X'12' subsection X'0002'**

Subsection X'0002' of the trusted block rule section (X'12') is the transport key rule reference TLV object. This subsection is optional. It contains the rule ID for the rule that must have been used to create the transport key. This subsection must be present to use an RKX key-token as a transport key.
This subsection is defined in the following table:

Table 263. Transport key rule reference subsection (X'0002') of trusted block rule section (X'12')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>002</td>
<td>Subsection tag: X'0002' Transport key rule reference TLV object</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Subsection length in bytes (14).</td>
</tr>
<tr>
<td>004</td>
<td>001</td>
<td>Subsection version number (X'00').</td>
</tr>
<tr>
<td>005</td>
<td>001</td>
<td>Reserved, must be binary zero.</td>
</tr>
<tr>
<td>006</td>
<td>008</td>
<td>Rule ID. Contains the rule identifier for the rule that must have been used to create the RKX key-token used as the transport key. The Rule ID is an 8-byte string of ASCII characters, left justified and padded on the right with space characters. Acceptable characters are A...Z, a...z, 0...9, - (X'2D'), and _ (X'5F'). All other characters are reserved for future use.</td>
</tr>
</tbody>
</table>

Trusted block section (X'12') subsection X'0003'

Subsection X'0003' of the trusted block rule section (X'12') is the common export key parameters TLV object. This subsection is optional, but is required for the key export of an existing source key (identified by the source_key_identifier parameter) in either RKX key-token format or CCA DES key-token format. For new key generation, this subsection applies the output key variant to the cleartext generated key, if such an option is desired. It contains the input source key and output export key minimum and maximum lengths, an output key variant length and variant, a CV length, and a CV to be exclusive-ORed with the cleartext transport key.

This subsection is defined in the following table:

Table 264. Common export key parameters subsection (X'0003') of trusted block rule section (X'12')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>002</td>
<td>Subsection tag: X'0003' Common export key parameters TLV object</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Subsection length in bytes (12+xxx+yyy).</td>
</tr>
<tr>
<td>004</td>
<td>001</td>
<td>Subsection version number (X'00').</td>
</tr>
<tr>
<td>005</td>
<td>002</td>
<td>Reserved, must be binary zero.</td>
</tr>
<tr>
<td>007</td>
<td>001</td>
<td>Flags (must be set to binary zero).</td>
</tr>
<tr>
<td>008</td>
<td>001</td>
<td>Export key minimum length in bytes. Length must be 8, 16, or 24. Also applies to the source key.</td>
</tr>
<tr>
<td>009</td>
<td>001</td>
<td>Export key maximum length in bytes (yyy). Length must be 8, 16, or 24. Also applies to the source key.</td>
</tr>
</tbody>
</table>
Table 264. Common export key parameters subsection (X'0003') of trusted block rule section (X'12') (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td>001</td>
<td>Output key variant length in bytes (xxx). Valid values are 0 or 8 - 255. If greater than 0, the length must be at least as long as the longest key ever to be exported using this rule. If the variant is longer than the key, truncate it on the right to the length of the key prior to use. <strong>Note:</strong> The output key variant (offset 011) is not used if this length is zero.</td>
</tr>
<tr>
<td>011</td>
<td>xxx</td>
<td>Output key variant. The variant can be any value. Exclusive-OR this variant into the cleartext value of the output.</td>
</tr>
<tr>
<td>011+xxx</td>
<td>001</td>
<td>CV length in bytes (yyy). If the length is not 0, 8, or 16, return an error. If the length is 0, and if the source key is a CCA DES key-token, preserve the CV in the symmetric encrypted output if the output is to be in the form of a CCA DES key-token. If a non-zero length is less than the length of the key identified by the source_key_identifier parameter, return an error. If the length is 16, and if the CV (offset 012+xxx) is valued to 16 bytes of X'00' (ignoring the key-part bit), then: 1. Ignore all CV bit definitions 2. If CCA DES key-token format, set the flag byte of the symmetric encrypted output key to indicate a CV value is present. 3. If the source key is 8 bytes in length, do not replicate the key to 16 bytes.</td>
</tr>
<tr>
<td>012+xxx</td>
<td>yyy</td>
<td>CV. Place this CV into the output exported key-token, provided that the symmetric encrypted output key format selected (offset 018 in rule section) is CCA DES key-token. If the symmetric encrypted output key format flag (offset 018 in section X'12') indicates return an RKX key-token (X'00'), then ignore this CV. Otherwise, exclusive-OR this CV into the cleartext transport key. Exclusive-OR the CV of the source key into the cleartext transport key if the CV length (offset 011+xxx) is set to 0. If a transport key to encrypt a source key has equal left and right key halves, return an error. Replicate the key halves of the key identified by the source_key_identifier parameter whenever all of these conditions are met: 1. The Replicate Key command (offset X'00DB') is enabled in the active role 2. The CV length (offset 011+xxx) is 16, and both CV halves are non-zero 3. The source_key_identifier parameter (contained in either a CCA DES key-token or RKX key-token) identifies an 8-byte key 4. The key-form bits (40 - 42) of this CV do not indicate a single-length key (are not set to zero) 5. Key-form bit 40 of this CV does not indicate the key is to have guaranteed unique halves (is not set to 1). <strong>Note:</strong> A transport key is not used when the symmetric encrypted output key is in RKX key-token format.</td>
</tr>
</tbody>
</table>

**Note:** See "Number representation in trusted blocks" on page 618.
**Trusted block section X'12' subsection X'0004'**

Subsection X'0004' of the trusted block rule section (X'12') is the source key rule reference TLV object. This subsection is optional, but is required if using an RKX key-token as a source key (identified by `source_key_identifier` parameter). It contains the rule ID for the rule used to create the export key. If this subsection is not present, an RKX key-token format source key will not be accepted for use.

This subsection is defined in the following table:

*Table 265. Source key rule reference subsection (X'0004' of trusted block rule section (X'12'))*

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>002</td>
<td>Subsection tag: X'0004' Source key rule reference TLV object</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Subsection length in bytes (14).</td>
</tr>
<tr>
<td>004</td>
<td>001</td>
<td>Subsection version number (X'00').</td>
</tr>
<tr>
<td>005</td>
<td>001</td>
<td>Reserved, must be binary zero.</td>
</tr>
<tr>
<td>006</td>
<td>008</td>
<td>Rule ID.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rule identifier for the rule that must have been used to create the source key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The Rule ID is an 8-byte string of ASCII characters, left justified and padded on the right with space characters. Acceptable characters are A...Z, a...z, 0...9, - (X'2D'), and _ (X'5F'). All other characters are reserved for future use.</td>
</tr>
</tbody>
</table>

**Note:** See "Number representation in trusted blocks" on page 618.

**Trusted block section X'12' subsection X'0005'**

Subsection X'0005' of the trusted block rule section (X'12') is the export key CCA token parameters TLV object. This subsection is optional. It contains a mask length, mask, and template for the export key CV limit. It also contains the template length and template for the source key label. When using a CCA DES key-token as a source key input parameter, its key type can be “filtered” by using the export key CV limit mask (offset 005) and limit template (offset 005+yyy) in this subsection.

This subsection is defined in the following table:

*Table 266. Export key CCA token parameters subsection (X'0005') of trusted block rule section (X'12')*

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>002</td>
<td>Subsection tag: X'0005' Export key CCA token parameters TLV object</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Subsection length in bytes (10+yyy+yyy+zzz).</td>
</tr>
<tr>
<td>004</td>
<td>001</td>
<td>Subsection version number (X'00').</td>
</tr>
<tr>
<td>005</td>
<td>002</td>
<td>Reserved, must be binary zero.</td>
</tr>
<tr>
<td>007</td>
<td>001</td>
<td>Flags (must be set to binary zero).</td>
</tr>
</tbody>
</table>
Table 266. Export key CCA token parameters subsection (X’0005’) of trusted block rule section (X’12’) (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>008</td>
<td>001</td>
<td>Export key CV limit mask length in bytes (yyy). Do not use CV limits if this CV limit mask length (yyy) is zero. Use CV limits if yyy is non-zero, in which case yyy:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Must be 8 or 16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Must not be less than the export key minimum length (offset 008 in subsection X’0003’)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Must be equal in length to the actual source key length of the key</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Example:</strong> An export key minimum length of 16 and an export key CV limit mask length of 8 returns an error.</td>
</tr>
<tr>
<td>009</td>
<td>yyy</td>
<td>Export key CV limit mask (does not exist if yyy=0). Indicates which CV bits to check against the source key CV limit template (offset 009+yyy). <strong>Examples:</strong> A mask of X’FF’ means check all bits in a byte. A mask of X’FE’ ignores the parity bit in a byte.</td>
</tr>
<tr>
<td>009+yyy</td>
<td>yyy</td>
<td>Export key CV limit template (does not exist if yyy=0). Specifies the required values for those CV bits that are checked based on the export key CV limit mask (offset 009). The export key CV limit mask and template have the same length, yyy. This is because these two variables work together to restrict the acceptable CVs for CCA DES key tokens to be exported. The checks work as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. If the length of the key to be exported is less than yyy, return an error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Logical AND the CV for the key to be exported with the export key CV limit mask</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Compare the result to the export key CV limit template</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. Return an error if the comparison is not equal</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Examples:</strong> An export key CV limit mask of X’FF’ for CV byte 1 (key type) along with an export key CV limit template of X’3F’ (key type CVARENC) for byte 1 filters out all key types except CVARENC keys. <strong>Note:</strong> Using the mask and template to permit multiple key types is possible, but cannot consistently be achieved with one rule section. For example, setting bit 10 to 1 in the mask and the template permits PIN processing keys and cryptographic variable encrypting keys, and only those keys. However, a mask to permit PIN-processing keys and key-encrypting keys, and only those keys, is not possible. In this case, multiple rule sections are required, one to permit PIN-processing keys and the other to permit key-encrypting keys.</td>
</tr>
<tr>
<td>009+yyy+yyy</td>
<td>001</td>
<td>Source key label template length in bytes (zzz). Valid values are 0 and 64. Return an error if the length is 64 and a source key label is not provided.</td>
</tr>
</tbody>
</table>
Table 266. Export key CCA token parameters subsection (X'0005') of trusted block rule section (X'12') (continued)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>010+yyy+yyy</td>
<td>zzz</td>
<td>Source key label template (does not exist if zzz=0).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If a key label is identified by the <code>source_key_identifier</code> parameter, verify that the key label name matches this template. If the comparison fails, return an error. The source key label template must conform to the following rules:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The key label template must be 64 bytes in length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The first character cannot be in the range X'00' - X'1F', nor can it be X'FF'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The first character cannot be numeric (X'30' - X'39')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• A key label name is terminated by a space character (X'20') on the right and must be padded on the right with space characters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The only special characters permitted are #, $, @, and * (X'23', X'24', X'40', and X'2A')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The wildcard X'2A' (*) is only permitted as the first character, the last character, or the only character in the template</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Only alphanumeric characters (a...z, A...Z, 0...9), the four special characters (X'23', X'24', X'40', and X'2A'), and the space character (X'20') are allowed</td>
</tr>
</tbody>
</table>

Note: See “Number representation in trusted blocks” on page 618.

Trusted block section X'13'

Trusted block section X'13' contains the name (key label). The trusted block name section provides a 64-byte variable to identify the trusted block, just as key labels are used to identify other CCA keys. This name, or label, enables a host access-control system such as RACF to use the name to verify that the application has authority to use the trusted block.

Section X'13' is optional. No multiple sections are allowed. It has no subsections defined. This section is defined in the following table:

Table 267. Trusted block key label (name) section X'13'

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Section identifier: X'13' Trusted block name (key label)</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Section version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length in bytes (68).</td>
</tr>
<tr>
<td>004</td>
<td>064</td>
<td>Name (key label).</td>
</tr>
</tbody>
</table>

Note: See “Number representation in trusted blocks” on page 618.

Trusted block section X'14'

Trusted block section X'14' contains control and security information related to the trusted block. This information section is separate from the public key and other sections because this section is required while the others are optional. This section contains the cryptographic information that guarantees its integrity and binds it to the local system.

Note: See “Number representation in trusted blocks” on page 618.
Section X'14' is required. No multiple sections are allowed. Two subsections are defined. This section is defined in the following table:

Table 268. Trusted block information section X'14'

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Section identifier: X'14' Trusted block information</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Section version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length in bytes (10+xxx).</td>
</tr>
<tr>
<td>004</td>
<td>002</td>
<td>Reserved, binary zero.</td>
</tr>
<tr>
<td>006</td>
<td>004</td>
<td>Flags: X'00000000' Trusted block is in the inactive state X'00000001' Trusted block is in the active state</td>
</tr>
<tr>
<td>010</td>
<td>xxx</td>
<td>Information section subsections (tag-length-value objects). One or two objects in TLV format.</td>
</tr>
</tbody>
</table>

Note: See "Number representation in trusted blocks" on page 618.

Trusted block section X'14' subsections

Section X'14' has two information subsections (tag-length-value objects) defined. These subsections are summarized in the following table:

Table 269. Summary of trusted block information subsections

<table>
<thead>
<tr>
<th>Rule subsection tag</th>
<th>TLV object</th>
<th>Optional or required</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0001'</td>
<td>Protection information</td>
<td>Required</td>
<td>Contains the encrypted 8-byte confounder and triple-length (24-byte) MAC key, the ISO 16609 TDES CBC MAC value, and the MKVP of the PKA master key (computed using MDC4).</td>
</tr>
<tr>
<td>X'0002'</td>
<td>Activation and expiration dates</td>
<td>Optional</td>
<td>Contains flags indicating whether or not the coprocessor is to validate dates, and contains the activation and expiration dates that are considered valid for the trusted block.</td>
</tr>
</tbody>
</table>

Note: See "Number representation in trusted blocks" on page 618.

Trusted block section X'14' subsection X'0001'

Subsection X'0001' of the trusted block information section (X'14') is the protection information TLV object. This subsection is required. It contains the encrypted 8-byte confounder and triple-length (24-byte) MAC key, the ISO-16609 TDES CBC MAC value, and the MKVP of the PKA master key (computed using MDC4).
This subsection is defined in the following table:

Table 270. Protection information subsection (X’0001’) of trusted block information section (X’14’)

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>002</td>
<td>Subsection tag: X’0001’ Trusted block information TLV object</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Subsection length in bytes (62).</td>
</tr>
<tr>
<td>004</td>
<td>001</td>
<td>Subsection version number (X’00’).</td>
</tr>
<tr>
<td>005</td>
<td>001</td>
<td>Reserved, must be binary zero.</td>
</tr>
<tr>
<td>006</td>
<td>032</td>
<td>Encrypted MAC key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Contains the encrypted 8-byte confounder and triple-length (24-byte) MAC key in the following format:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Offset Description</td>
</tr>
<tr>
<td>00 - 07</td>
<td>Confounder</td>
<td></td>
</tr>
<tr>
<td>08 - 15</td>
<td>Left key</td>
<td></td>
</tr>
<tr>
<td>16 - 23</td>
<td>Middle key</td>
<td></td>
</tr>
<tr>
<td>24 - 31</td>
<td>Right key</td>
<td></td>
</tr>
<tr>
<td>038</td>
<td>008</td>
<td>MAC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Contains the ISO-16609 TDES CBC message authentication code value.</td>
</tr>
<tr>
<td>046</td>
<td>016</td>
<td>MKVP.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Contains the PKA master key verification pattern, computed using MDC4, when the trusted block is in internal form, otherwise contains binary zero.</td>
</tr>
</tbody>
</table>

Note: See “Number representation in trusted blocks” on page 618.

Trusted block section X’14’ subsection X’0002’

Subsection X’0002’ of the trusted block information section (X’14’) is the activation and expiration dates TLV object. This subsection is optional. It contains flags indicating whether or not the coprocessor is to validate dates, and contains the activation and expiration dates that are considered valid for the trusted block.
This subsection is defined in the following table:

Table 271. Activation and expiration dates subsection (X'0002') of trusted block information section (X'14')

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>002</td>
<td>Subsection tag: X'0002' Activation and expiration dates TLV object</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Subsection length in bytes (16).</td>
</tr>
<tr>
<td>004</td>
<td>001</td>
<td>Subsection version number (X'00').</td>
</tr>
<tr>
<td>005</td>
<td>001</td>
<td>Reserved, must be binary zero.</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>Flags:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0000' The coprocessor does not check dates.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0001' The coprocessor checks dates.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compare the activation date (offset 008) and the expiration date (offset 012) to the coprocessor's internal real-time clock. Return an error if the coprocessor date is before the activation date or after the expiration date.</td>
</tr>
<tr>
<td>008</td>
<td>004</td>
<td>Activation date.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Contains the first date that the trusted block can be used for generating or exporting keys. Format of the date is YYMD, where:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YY Big-endian year (return an error if greater than 9999)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M Month (return an error if any value other than X'01' - X'0C')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D Day of month (return an error if any value other than X'01' - X'1F'; day must be valid for given month and year, including leap years)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Return an error if the activation date is after the expiration date or is not valid.</td>
</tr>
<tr>
<td>012</td>
<td>004</td>
<td>Expiration date.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Contains the last date that the trusted block can be used. Same format as activation date (offset 008). Return an error if date is not valid.</td>
</tr>
</tbody>
</table>

Note: See "Number representation in trusted blocks" on page 618.

**Trusted block section X'15'**

Trusted block section X'15' contains application-defined data. The trusted block application-defined data section can be used to include application-defined data in the trusted block. The purpose of the data in this section is defined by the application; it is neither examined nor used by CCA in any way.

Section X'15' is optional. No multiple sections are allowed. It has no subsections defined. This section is defined in the following table:
Table 272. Trusted block application-defined data section X'15’

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>Section identifier: X'15’ Application-defined data</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>Section version number (X'00').</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Section length (6+xxx)</td>
</tr>
</tbody>
</table>
| 004           | 002            | Application data length (xxx)  
The value of xxx can be from 0 bytes to a length that does not cause the trusted block to exceed its maximum size of 3500 bytes. |
| 006           | xxx            | Application-defined data  
May be used to hold a public-key certificate for the trusted public key. |

**Note:** See [Number representation in trusted blocks](https://www.ibm.com/support/knowledgecenter/SSS7H7_1.3.2/com.ibm.zos.v1r11.icsf.apilibr.daprguides_1.3.2/...on page 618].
Appendix C. Control Vectors and Changing Control Vectors with the CVT Callable Service

This section contains a control vector table which displays the default value of the control vector that is associated with each type of key. It also describes how to change control vectors with the control vector translate callable service.

Control Vector Table

**Note:** The Control Vectors used in ICSF are exactly the same as documented in CCA and the TSS documents.

The master key enciphers all keys operational on your system. A transport key enciphers keys that are distributed off your system. Before a master key or transport key enciphers a key, ICSF exclusive ORs both halves of the master key or transport key with a control vector. The same control vector is exclusive ORed to the left and right half of a master key or transport key.

Also, if you are entering a key part, ICSF exclusive ORs each half of the key part with a control vector before placing the key part into the CKDS.

Each type of key on ICSF (except the master key) has either one or two unique control vectors associated with it. The control vector that ICSF exclusive ORs the master key or transport key with depends on the type of key the master key or transport key is enciphering. For double-length keys, a unique control vector exists for each half of a specific key type. For example, there is a control vector for the left half of an input PIN-encrypting key, and a control vector for the right half of an input PIN-encrypting key.

If you are entering a key part into the CKDS, ICSF exclusive ORs the key part with the unique control vector(s) associated with the key type. ICSF also enciphers the key part with two master key variants for a key part. One master key variant enciphers the left half of the key part, and another master key variant enciphers the right half of the key part. ICSF creates the master key variants for a key part by exclusive ORing the master key with the control vectors for key parts. These procedures protect key separation.

Table **273** displays the default value of the control vector that is associated with each type of key. Some key types do not have a default control vector. For keys that are double-length, ICSF enciphers a unique control vector on each half. Control vectors indicated with an "*" are supported by the Cryptographic Coprocessor Feature.

**Table 273. Default Control Vector Values**

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Control Vector Value (Hex) Value for Single-length Key or Left Half of Double-length Key</th>
<th>Control Vector Value (Hex) Value for Right Half of Double-length Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>*AKEK</td>
<td>00 00 00 00 00 00 00 00 00 00</td>
<td></td>
</tr>
<tr>
<td>CIPHER</td>
<td>00 03 71 00 03 00 00 00</td>
<td></td>
</tr>
<tr>
<td>CIPHER (double length)</td>
<td>00 03 71 00 03 41 00 00 00 00 03 71 00 03 21 00 00</td>
<td></td>
</tr>
<tr>
<td>CVARDEC</td>
<td>00 3F 42 00 03 00 00 00 00 00</td>
<td></td>
</tr>
</tbody>
</table>
Table 273. Default Control Vector Values (continued)

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Control Vector Value (Hex) Value for Single-length Key or Left Half of Double-length Key</th>
<th>Control Vector Value (Hex) Value for Right Half of Double-length Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVARENC</td>
<td>00 3F 48 00 03 00 00 00</td>
<td></td>
</tr>
<tr>
<td>CVARPINE</td>
<td>00 3F 41 00 03 00 00 00</td>
<td></td>
</tr>
<tr>
<td>CVARXCVL</td>
<td>00 3F 44 00 03 00 00 00</td>
<td></td>
</tr>
<tr>
<td>CVARXCVR</td>
<td>00 3F 47 00 03 00 00 00</td>
<td></td>
</tr>
<tr>
<td>*DATA</td>
<td>00 00 00 00 00 00 00 00</td>
<td></td>
</tr>
<tr>
<td>DATAC</td>
<td>00 00 71 00 03 41 00 00</td>
<td>00 00 71 00 03 21 00 00</td>
</tr>
<tr>
<td>*DATAM generation key (external)</td>
<td>00 00 4D 00 03 41 00 00</td>
<td>00 00 4D 00 03 21 00 00</td>
</tr>
<tr>
<td>*DATAM key (internal)</td>
<td>00 05 4D 00 03 00 00 00</td>
<td>00 05 4D 00 03 00 00 00</td>
</tr>
<tr>
<td>*DATAMV MAC verification key (external)</td>
<td>00 00 44 00 03 41 00 00</td>
<td>00 00 44 00 03 21 00 00</td>
</tr>
<tr>
<td>*DATAMV MAC verification key (internal)</td>
<td>00 05 44 00 03 00 00 00</td>
<td>00 05 44 00 03 00 00 00</td>
</tr>
<tr>
<td>*DATAXLAT</td>
<td>00 06 71 00 03 00 00 00</td>
<td></td>
</tr>
<tr>
<td>DECRYPTER</td>
<td>00 03 50 00 03 00 00 00</td>
<td></td>
</tr>
<tr>
<td>DECRYPTER (double-length)</td>
<td>00 03 50 00 03 41 00 00</td>
<td>00 03 50 00 03 21 00 00</td>
</tr>
<tr>
<td>DKEYGENKY</td>
<td>00 71 44 00 03 41 00 00</td>
<td>00 71 44 00 03 21 00 00</td>
</tr>
<tr>
<td>ENCRYPTER</td>
<td>00 03 60 00 03 00 00 00</td>
<td></td>
</tr>
<tr>
<td>ENCRYPTER (double-length)</td>
<td>00 03 60 00 03 41 00 00</td>
<td>00 03 60 00 03 21 00 00</td>
</tr>
<tr>
<td>*EXPORTER</td>
<td>00 41 7D 00 03 41 00 00</td>
<td>00 41 7D 00 03 21 00 00</td>
</tr>
<tr>
<td>IKEYXLAT</td>
<td>00 42 42 00 03 41 00 00</td>
<td>00 42 42 00 03 21 00 00</td>
</tr>
<tr>
<td>*IMP-PKA</td>
<td>00 42 05 00 03 41 00 00</td>
<td>00 42 05 00 03 21 00 00</td>
</tr>
<tr>
<td>*IMPORTER</td>
<td>00 42 7D 00 03 41 00 00</td>
<td>00 42 7D 00 03 21 00 00</td>
</tr>
<tr>
<td>*IPINENC</td>
<td>00 21 5F 00 03 41 00 00</td>
<td>00 21 5F 00 03 21 00 00</td>
</tr>
<tr>
<td>*MAC</td>
<td>00 05 4D 00 03 00 00 00</td>
<td></td>
</tr>
<tr>
<td>MAC (double-length)</td>
<td>00 05 4D 00 03 41 00 00</td>
<td>00 05 4D 00 03 21 00 00</td>
</tr>
<tr>
<td>*MACVER</td>
<td>00 05 44 00 03 00 00 00</td>
<td></td>
</tr>
<tr>
<td>MACVER (double-length)</td>
<td>00 05 44 00 03 41 00 00</td>
<td>00 05 44 00 03 21 00 00</td>
</tr>
<tr>
<td>OKEYXLAT</td>
<td>00 41 42 00 03 41 00 00</td>
<td>00 41 42 00 03 21 00 00</td>
</tr>
<tr>
<td>*OPINENC</td>
<td>00 24 77 00 03 41 00 00</td>
<td>00 24 77 00 03 21 00 00</td>
</tr>
<tr>
<td>*PINGEN</td>
<td>00 22 7E 00 03 41 00 00</td>
<td>00 22 7E 00 03 21 00 00</td>
</tr>
<tr>
<td>*PINVER</td>
<td>00 22 42 00 03 41 00 00</td>
<td>00 22 42 00 03 21 00 00</td>
</tr>
</tbody>
</table>

**Note:** The external control vectors for DATAC, DATAM MAC generation and DATAMV MAC verification keys are also referred to as data compatibility control vectors.
Figure 9. Control Vector Base Bit Map (Common Bits and Key-Encrypting Keys)
**Control-Vector Base Bits**

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

**Data Operation Keys**

- **DATA**: 00000000 00000000 0Eedmv0P 00000000 00000011 fff0K00P 00000000 00000000
- **DATAC**: 00000000 00000000 0E11000P 00000000 00000011 fff0K00P 00000000 00000000
- **DATAM**: 00000000 00000000 0E10100P 00000000 00000011 fff0K00P 00000000 00000000
- **DATAMV**: 00000000 00000000 0E00110P 00000000 00000011 fff0K00P 00000000 00000000
- **CIPHER**: 00000000 00000000 0E00111P 00000000 00000011 fff0K00P 00000000 00000000
- **DECIPHER**: 00000000 00000000 0E01000P 00000000 00000011 fff0K00P 00000000 00000000
- **ENCIPHER**: 00000000 00000000 0E10000P 00000000 00000011 fff0K00P 00000000 00000000
- **SECMSG**: 00000000 00000000 0E00010P 00000000 00000011 fff0K00P 00000000 00000000
- **MAC**: ccccccc0 00000101 0E00110P 00000000 00000011 fff0K00P 00000000 00000000
- **MACVER**: ccccccc0 00000101 0E00010P 00000000 00000011 fff0K00P 00000000 00000000

**Figure 10. Control Vector Base Bit Map (Data Operation Keys)**
Appendix C. Control Vectors and Changing Control Vectors with the CVT Callable Service 637
Key Form Bits, 'fff'

- The key form bits, 40-42, and for a double-length key, bits 104-106, are designated 'fff' in the preceding illustration. These bits can have these values:

  - **000**: Single length key
  - **010**: Double length key, left half
  - **001**: Double length key, right half

  The following values may exist in some CCA implementations:
  - **110**: Double-length key, left half, halves guaranteed unique
  - **101**: Double-length key, right half, halves guaranteed unique

**Specifying a Control-Vector-Base Value**

You can determine the value of a control vector by working through the following series of questions:

1. Begin with a field of 64 bits (eight bytes) set to B'0'. The most significant bit is referred to as bit 0. Define the key type and subtype (bits 8 to 14), as follows:
   - The main key type bits (bits 8 to 11), Set bits 8 to 11 to one of the following values:
### Bits 8 to 11  Main Key Type

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Data operation keys</td>
</tr>
<tr>
<td>0010</td>
<td>PIN keys</td>
</tr>
<tr>
<td>0011</td>
<td>Cryptographic variable-encrypting keys</td>
</tr>
<tr>
<td>0100</td>
<td>Key-encrypting keys</td>
</tr>
<tr>
<td>0101</td>
<td>Key-generating keys</td>
</tr>
<tr>
<td>0111</td>
<td>Diversified key-generating keys</td>
</tr>
</tbody>
</table>

- The key subtype bits (bits 12 to 14). Set bits 12 to 14 to one of the following values:

**Note:** For Diversified Key Generating Keys, the subtype field specifies the hierarchical level of the DKYGENKY. If the subtype is non-zero, then the DKYGENKY can only generate another DKYGENKY key with the hierarchy level decremented by one. If the subtype is zero, the DKYGENKY can only generate the final diversified key (a non-DKYGENKY key) with the key type specified by the usage bits.

### Bits 12 to 14  Key Subtype

**Data Operation Keys**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Compatibility key (DATA)</td>
</tr>
<tr>
<td>001</td>
<td>Confidentiality key (CIPHER, DECIPHER, or ENCIPHER)</td>
</tr>
<tr>
<td>010</td>
<td>MAC key (MAC or MACVER)</td>
</tr>
<tr>
<td>101</td>
<td>Secure messaging keys</td>
</tr>
</tbody>
</table>

**Key-Encrypting Keys**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Transport-sending keys (EXPORTER and OKEYXLAT)</td>
</tr>
<tr>
<td>001</td>
<td>Transport-receiving keys (IMPORTER and IKEYXLAT)</td>
</tr>
</tbody>
</table>

**PIN Keys**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>PIN-generating key (PINGEN, PINVER)</td>
</tr>
<tr>
<td>000</td>
<td>Inbound PIN-block decrypting key (IPINENC)</td>
</tr>
<tr>
<td>010</td>
<td>Outbound PIN-block encrypting key (OPINENC)</td>
</tr>
</tbody>
</table>

**Cryptographic Variable-Encrypting Keys**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>Cryptographic variable-encrypting key (CVAR....)</td>
</tr>
</tbody>
</table>

**Diversified Key Generating Keys**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>DKY Subtype 0</td>
</tr>
<tr>
<td>001</td>
<td>DKY Subtype 1</td>
</tr>
<tr>
<td>010</td>
<td>DKY Subtype 2</td>
</tr>
<tr>
<td>011</td>
<td>DKY Subtype 3</td>
</tr>
<tr>
<td>100</td>
<td>DKY Subtype 4</td>
</tr>
<tr>
<td>101</td>
<td>DKY Subtype 5</td>
</tr>
<tr>
<td>110</td>
<td>DKY Subtype 6</td>
</tr>
<tr>
<td>111</td>
<td>DKY Subtype 7</td>
</tr>
</tbody>
</table>

2. For key-encrypting keys, set the following bits:
   - The key-generating usage bits (gks, bits 18 to 20). Set the gks bits to B’111’ to indicate that the Key Generate callable service can use the associated...
key-encrypting key to encipher generated keys when the Key Generate callable service is generating various key-pair key-form combinations (see the Key-Encrypting Keys section of Figure 9). Without any of the gks bits set to 1, the Key Generate callable service cannot use the associated key-encrypting key. The Key Token Build callable service can set the gks bits to 1 when you supply the OPIM, IMEX, IMIM, OPEX, and EXEX keywords.

- The IMPORT and EXPORT bit and the XLATE bit (ix, bits 21 and 22). If the 'i' bit is set to 1, the associated key-encrypting key can be used in the Data Key Import, Key Import, Data Key Export, and Key Export callable services. If the 'x' bit is set to 1, the associated key-encrypting key can be used in the Key Translate callable service.

- The key-form bits (fff, bits 40 to 42). The key-form bits indicate how the key was generated and how the control vector participates in multiple-enciphering. To indicate that the parts can be the same value, set these bits to B'010'. For information about the value of the key-form bits in the right half of a control vector, see Step 8.

3. For MAC and MACVER keys, set the following bits:
   - The MAC control bits (bits 20 and 21). For a MAC-generate key, set bits 20 and 21 to B'11'. For a MAC-verify key, set bits 20 and 21 to B'01'.
   - The key-form bits (fff, bits 40 to 42). For a single-length key, set the bits to B'000'. For a double-length key, set the bits to B'010'.

4. For PINGEN and PINVER keys, set the following bits:
   - The PIN calculation method bits (aaaa, bits 0 to 3). Set these bits to one of the following values:

<table>
<thead>
<tr>
<th>Bits 0 to 3</th>
<th>Calculation Method Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>NO-SPEC</td>
<td>A key with this control vector can be used with any PIN calculation method.</td>
</tr>
<tr>
<td>0001</td>
<td>IBM-PIN or IBM-PINO</td>
<td>A key with this control vector can be used only with the IBM PIN or PIN Offset calculation method.</td>
</tr>
<tr>
<td>0010</td>
<td>VISA-PVV</td>
<td>A key with this control vector can be used only with the VISA-PVV calculation method.</td>
</tr>
<tr>
<td>0100</td>
<td>GBP-PIN or GBP-PINO</td>
<td>A key with this control vector can be used only with the German Banking Pool PIN or PIN Offset calculation method.</td>
</tr>
<tr>
<td>0011</td>
<td>INBK-PIN</td>
<td>A key with this control vector can be used only with the Interbank PIN calculation method.</td>
</tr>
<tr>
<td>0101</td>
<td>NL-PIN-1</td>
<td>A key with this control vector can be used only with the NL-PIN-1, Netherlands PIN calculation method.</td>
</tr>
</tbody>
</table>

- The prohibit-offset bit (o, bit 37) to restrict operations to the PIN value. If set to 1, this bit prevents operation with the IBM 3624 PIN Offset calculation method and the IBM German Bank Pool PIN Offset calculation method.
5. For PINGEN, IPINENC, and OPINENC keys, set bits 18 to 22 to indicate whether the key can be used with the following callable services

<table>
<thead>
<tr>
<th>Service Allowed</th>
<th>Bit Name</th>
<th>Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear PIN Generate</td>
<td>CPINGEN</td>
<td>18</td>
</tr>
<tr>
<td>Encrypted PIN Generate Alternate</td>
<td>EPINGENA</td>
<td>19</td>
</tr>
<tr>
<td>Encrypted PIN Generate</td>
<td>EPINGEN</td>
<td>20 for PINGEN 19 for OPINENC</td>
</tr>
<tr>
<td>Clear PIN Generate Alternate</td>
<td>CPINGENA</td>
<td>21 for PINGEN 20 for IPINENC</td>
</tr>
<tr>
<td>Encrypted Pin Verify</td>
<td>EPINVER</td>
<td>19</td>
</tr>
<tr>
<td>Clear PIN Encrypt</td>
<td>CPINENC</td>
<td>18</td>
</tr>
</tbody>
</table>

6. For the IPINENC (inbound) and OPINENC (outbound) PIN-block ciphering keys, do the following:
   - Set the TRANSLAT bit (t, bit 21) to 1 to permit the key to be used in the PIN Translate callable service. The Control Vector Generate callable service can set the TRANSLAT bit to 1 when you supply the TRANSLAT keyword.
   - Set the REFORMAT bit (r, bit 22) to 1 to permit the key to be used in the PIN Translate callable service. The Control Vector Generate callable service can set the REFORMAT bit and the TRANSLAT bit to 1 when you supply the REFORMAT keyword.

7. For the cryptographic variable-encrypting keys (bits 18 to 22), set the variable-type bits (bits 18 to 22) to one of the following values:

<table>
<thead>
<tr>
<th>Bits 18 to 22</th>
<th>Generic Key Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>CVARPINE</td>
<td>Used in the Encrypted PIN Generate Alternate service to encrypt a clear PIN.</td>
</tr>
<tr>
<td>00010</td>
<td>CVARXCVL</td>
<td>Used in the Control Vector Translate callable service to decrypt the left mask array.</td>
</tr>
<tr>
<td>00011</td>
<td>CVARXCVR</td>
<td>Used in the Control Vector Translate callable service to decrypt the right mask array.</td>
</tr>
<tr>
<td>00100</td>
<td>CVARENC</td>
<td>Used in the Cryptographic Variable Encipher callable service to encrypt an unformatted PIN.</td>
</tr>
</tbody>
</table>

8. For key-generating keys, set the following bits:
   - For KEYGENKY, set bit 18 for UKPT usage and bit 19 for CLR8-ENC usage.
   - For DKYGENKY, bits 12–14 will specify the hierarchical level of the DKYGENKY key. If the subtype CV bits are non-zero, then the DKYGENKY can only generate another DKYGENKY key with the hierarchical level decremented by one. If the subtype CV bits are zero, the DKYGENKY can only generate the final diversified key (a non-DKYGENKY key) with the key type specified by usage bits.
To specify the subtype values of the DKYGENKY, keywords DKYL0, DKYL1, DKYL2, DKYL3, DKYL4, DKYL5, DKYL6 and DKYL7 will be used.

- For DKYGENKY, bit 18 is reserved and must be zero.
- Usage bits 18-22 for the DKYGENKY key type are defined as follows. They will be encoded as the final key type that the DKYGENKY key generates.

<table>
<thead>
<tr>
<th>Bits 19 to 22</th>
<th>Keyword</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>DDATA</td>
<td>DATA, DATAC, single or double length</td>
</tr>
<tr>
<td>0010</td>
<td>DMAC</td>
<td>MAC, DATAM</td>
</tr>
<tr>
<td>0011</td>
<td>DMV</td>
<td>MACVER, DATAMV</td>
</tr>
<tr>
<td>0100</td>
<td>DIMP</td>
<td>IMPORTER, IKEYXLAT</td>
</tr>
<tr>
<td>0101</td>
<td>DEXP</td>
<td>EXPORTER, OKEYXLAT</td>
</tr>
<tr>
<td>0110</td>
<td>DPVR</td>
<td>PINVER</td>
</tr>
<tr>
<td>1000</td>
<td>DMKEY</td>
<td>Secure message key for encrypting keys</td>
</tr>
<tr>
<td>1001</td>
<td>DMPIN</td>
<td>Secure message key for encrypting PINs</td>
</tr>
<tr>
<td>1111</td>
<td>DALL</td>
<td>All key types may be generated except DKYGENKY and KEYGENKY keys. Usage of the DALL keyword is controlled by a separate access control point.</td>
</tr>
</tbody>
</table>

9. For secure messaging keys, set the following bits:
   - Set bit 18 to 1 if the key will be used in the secure messaging for PINs service. Set bit 19 to 1 if the key will be used in the secure messaging for keys service.

10. For all keys, set the following bits:
   - The export bit (E, bit 17). If set to 0, the export bit prevents a key from being exported. By setting this bit to 0, you can prevent the receiver of a key from exporting or translating the key for use in another cryptographic subsystem. Once this bit is set to 0, it cannot be set to 1 by any service other than Control Vector Translate. The Prohibit Export callable service can reset the export bit.
   - The key-part bit (K, bit 44). Set the key-part bit to 1 in a control vector associated with a key part. When the final key part is combined with previously accumulated key parts, the key-part bit in the control vector for the final key part is set to 0. The Control Vector Generate callable service can set the key-part bit to 1 when you supply the KEY-PART keyword.
   - The anti-variant bits (bit 30 and bit 38). Set bit 30 to 0 and bit 38 to 1. Many cryptographic systems have implemented a system of variants where a 7-bit value is exclusive-ORed with each 7-bit group of a key-encrypting key before enciphering the target key. By setting bits 30 and 38 to opposite values, control vectors do not produce patterns that can occur in variant-based systems.
   - Control vector bits 64 to 127. If bits 40 to 42 are B'000' (single-length key), set bits 64 to 127 to 0. Otherwise, copy bits 0 to 63 into bits 64 to 127 and set bits 105 and 106 to B'01'.
• Set the parity bits (low-order bit of each byte, bits 7, 15, ..., 127). These bits contain the parity bits (P) of the control vector. Set the parity bit of each byte so the number of zero-value bits in the byte is an even number.
• For secure messaging keys, usage bit 18 on will enable the encryption of keys in a secure message and usage bit 19 on will enable the encryption of PINs in a secure message.

Changing Control Vectors with the Control Vector Translate Callable Service

Do the following when using the Control Vector Translate callable service:
• Provide the control information for testing the control vectors of the source, target, and key-encrypting keys to ensure that only sanctioned changes can be performed
• Select the key-half processing mode.

Providing the Control Information for Testing the Control Vectors

To minimize your security exposure, the Control Vector Translate callable service requires control information (mask array information) to limit the range of allowable control vector changes. To ensure that this service is used only for authorized purposes, the source-key control vector, target-key control vector, and key-encrypting key (KEK) control vector must pass specific tests. The tests on the control vectors are performed within the secured cryptographic engine.

The tests consist of evaluating four logic expressions, the results of which must be a string of binary zeros. The expressions operate bitwise on information that is contained in the mask arrays and in the portions of the control vectors associated with the key or key-half that is being processed. If any of the expression evaluations do not result in all zero bits, the callable service is ended with a control vector violation return and reason code (8/39). See Figure 13. Only the 56 bit positions that are associated with a key value are evaluated. The low-order bit that is associated with key parity in each key byte is not evaluated.

Mask Array Preparation

A mask array consists of seven 8-byte elements: A1, B1, A2, B2, A3, B3, and B4. You choose the values of the array elements such that each of the following four expressions evaluates to a string of binary zeros. (See Figure 13 on page 645.) Set the A bits to the value that you require for the corresponding control vector bits. In expressions 1 through 3 set the B bits to select the control vector bits to be evaluated. In expression 4 set the B bits to select the source and target control vector bits to be evaluated. Also, use the following control vector information:

- C1 is the control vector associated with the left half of the KEK.
- C2 is the control vector associated with the source key, or selected source-key half/halves.
- C3 is the control vector associated with the target key or selected target-key half/halves.

1. (C1 exclusive-OR A1) logical-AND B1
   This expression tests whether the KEK used to encipher the key meets your criteria for the desired translation.
2. (C2 exclusive-OR A2) logical-AND B2
   This expression tests whether the control vector associated with the source key meets your criteria for the desired translation.
3. \((C_3 \text{ exclusive-OR } A_3) \text{ logical-AND } B_3\)

This expression tests whether the control vector associated with the target key meets your criteria for the desired translation.

4. \((C_2 \text{ exclusive-OR } C_3) \text{ logical-AND } B_4\)

This expression tests whether the control vectors associated with the source key and the target key meet your criteria for the desired translation.

Encipher two copies of the mask array, each under a different cryptographic-variable key (key type CVARENC). To encipher each copy of the mask array, use the Cryptographic Variable Encipher callable service. Use two different keys so that the enciphered-array copies are unique values. When using the Control Vector Translate callable service, the mask_array_left parameter and the mask_array_right parameter identify the enciphered mask arrays. The array_key_left parameter and the array_key_right parameter identify the internal keys for deciphering the mask arrays. The array_key_left key must have a key type of CVARXCVL and the array_key_right key must have a key type of CVARXCVR. The cryptographic process decipheres the arrays and compares the results; for the service to continue, the deciphered arrays must be equal. If the results are not equal, the service returns the return and reason code for data that is not valid (8/385).

Use the Key Generate callable service to create the key pairs CVARENC-CVARXCVL and CVARENC-CVARXCVR. Each key in the key pair must be generated for a different node. The CVARENC keys are generated for, or imported into, the node where the mask array will be enciphered. After enciphering the mask array, you should destroy the enciphering key. The CVARXCVL and CVARXCVR keys are generated for, or imported into, the node where the Control Vector Translate callable service will be performed.

If using the BOTH keyword to process both halves of a double-length key, remember that bits 41, 42, 104, and 105 are different in the left and right halves of the CCA control vector and must be ignored in your mask-array tests (that is, make the corresponding \(B_2\) and/or \(B_3\) bits equal to zero).

When the control vectors pass the masking tests, the verb does the following:

- Deciphers the source key. In the decipher process, the service uses a key that is formed by the exclusive-OR of the KEK and the control vector in the key token variable the source_key_token parameter identifies.
- Enciphers the deciphered source key. In the encipher process, the service uses a key that is formed by the exclusive-OR of the KEK and the control vector in the key token variable the target_key_token parameter identifies.
- Places the enciphered key in the key field in the key token variable the target_key_token parameter identifies.
Selecting the Key-Half Processing Mode

Use the Control Vector Translate callable service to change a control vector associated with a key. Rule-array keywords determine which key halves are processed in the call, as shown in Figure 14 on page 646.
**Keyword Meaning**

**SINGLE**

This keyword causes the control vector of the left half of the source key to be changed. The updated key half is placed into the left half of the target key in the target key token. The right half of the target key is unchanged.

The **SINGLE** keyword is useful when processing a single-length key, or when first processing the left half of a double-length key (to be followed by processing the right half).

**RIGHT**

This keyword causes the control vector of the right half of the source key to be changed. The updated key half is placed into the right half of the target key of the target key token. The left half of the source key is copied unchanged into the left half of the target key in the target key token.

**BOTH**

This keyword causes the control vector of both halves of the source key to be changed. The updated key is placed into the target key in the target key token.

A single set of control information must permit the control vector changes applied to each key half. Normally, control vector bit positions 41, 42, 105, and 106 are different for each key half. Therefore, set bits 41 and 42 to B'00' in mask array elements B₁, B₂, and B₃.

You can verify that the source and target key tokens have control vectors with matching bits in bit positions 40-42 and 104-106, the “form field” bits. Ensure that bits 40-42 of mask array B₄ are set to B'111'.

**LEFT**

This keyword enables you to supply a single-length key and obtain a double-length key. The source key token must contain:

- The KEK-enciphered single-length key
- The control vector for the single-length key (often this is a null value)
- A control vector, stored in the source token where the right-half control vector is normally stored, used in decrypting the single-length source key when the key is being processed for the target right half of the key.

The service first processes the source and target tokens as with the **SINGLE** keyword. Then the source token is processed using the single-length enciphered key and the source token right-half control.
vector to obtain the actual key value. The key value is then enciphered using the KEK and the control vector in the target token for the right-half of the key.

This approach is frequently of use when you must obtain a double-length CCA key from a system that only supports a single-length key, for example when processing PIN keys or key-encrypting keys received from non-CCA systems.

To prevent the service from ensuring that each key byte has odd parity, you can specify the NOADJUST keyword. If you do not specify the NOADJUST keyword, or if you specify the ADJUST keyword, the service ensures that each byte of the target key has odd parity.

When the Target Key Token CV Is Null

When you use any of the LEFT, BOTH, or RIGHT keywords, and when the control vector in the target key token is null (all B'0'), then bit 3 in byte 59 will be set to B'1' to indicate that this is a double-length DATA key.

Control Vector Translate Example

As an example, consider the case of receiving a single-length PIN-block encrypting key from a non-CCA system. Often such a key will be encrypted by an unmodified transport key (no control vector or variant is used). In a CCA system, an inbound PIN encrypting key is double-length.

First use the Key Token Build callable service to insert the single-length key value into the left-half key-space in a key token. Specify USE-CV as a key type and a control vector value set to 16 bytes of X'00'. Also specify EXTERNAL, KEY, and CV keywords in the rule array. This key token will be the source key key token.

Second, the target key token can also be created using the Key Token Build callable service. Specify a key type of IPINENC and the NO-EXPORT rule array keyword.

Then call the Control Vector Translate callable service and specify a rule-array keyword of LEFT. The mask arrays can be constructed as follows:

- **A1** is set to the value of the KEK's control vector, most likely the value of an IMPORTER key, perhaps with the NO-EXPORT bit set. **B1** is set to eight bytes of X'FF' so that all bits of the KEK's control vector will be tested.
- **A2** is set to eight bytes of X'00', the (null) value of the source key control vector. **B2** is set to eight bytes of X'FF' so that all bits of the source-key “control vector” will be tested.
- **A3** is set to the value of the target key’s left-half control vector. **B3** is set to X'FFFF FFFF FF9F FFFF'. This will cause all bits of the control vector to be tested except for the two (“fff”) bits used to distinguish between the left-half and right-half target-key control vector.
- **B4** is set to eight bytes of X'00' so that no comparison is made between the source and target control vectors.
Appendix D. Coding Examples

This appendix provides sample routines using the ICSF callable services for these languages:

- C
- COBOL
- Assembler
- PL/1

The C, COBOL and Assembler H examples that follow use the key generate, encipher, and decipher callable services to determine whether the deciphered text matches the starting text.

For C programs, the header file cfsbext.h, which contains stubs for calling the ICSF services, is installed in the HFS directory /usr/include and is copied to SYS1.SIEAHDR.H(CSFBEXT). The cfsbext.h header file is required by C applications.

Incidentally, information on creating C applications that call ICSF services is available in the book "z/OS Cryptographic Services ICSF Writing PKCS #11 Applications".

---

```c
#include <stdio.h>
#include "csfbext.h"

#include <stdio.h>
#include "csfbext.h"

/* Prototypes for functions in this example */

/* Utility for printing hex strings */
void printHex(unsigned char *, unsigned int);

/* Main Function */
int main(void) {

    /* Constant inputs to ICSF services */
    static int textLen = 24;
    static unsigned char clearText[24] = "ABCDEFHIJKLMNOPQRSTUVWXYZ";
    static unsigned char cipherProcessRule[8] = "CUSB ";
    static unsigned char keyForm[4] = "OP " ;
    static unsigned char keyLength[8] = "SINGLE " ;
    static unsigned char dataKeyType[8] = "DATA " ;
    static unsigned char nullKeyType[8] = " " ;
    static unsigned char ICV[8] = {0};
    static int *pad = 0;
    static int exitDataLength = 0;
    static unsigned char exitData[4] = {0};
    static int ruleArrayCount = 1;
```
* Variable inputs/outputs for ICSF services *
*------------------------------------------------------------------*/
unsigned char cipherText[24]={0};
unsigned char compareText[24]={0};
unsigned char dataKeyId[64]={0};
unsigned char nullKeyId[64]={0};
unsigned char dummyKEKKeyId1[64]={0};
unsigned char dummyKEKKeyId2[64]={0};
int returnCode = 0;
int reasonCode = 0;
unsigned char OCV[18]={0};
/*------------------------------------------------------------------*
* Begin executable code *
*------------------------------------------------------------------*/
do {
  /*------------------------------------------------------------------*
  * Call key generate *
  *------------------------------------------------------------------*/
  if ((returnCode = CSNBKGN(&returnCode,
           &reasonCode,
           &exitDataLength,
           exitData,
           keyForm,
           keyLength,
           dataKeyType,
           nullKeyType,
           dummyKEKKeyId1,
           dummyKEKKeyId2,
           dataKeyId,
           nullKeyId)) != 0) {
    printf("Key Generate failed:\n");
    printf(" Return Code = %04d\n",returnCode);
    printf(" Reason Code = %04d\n",reasonCode);
    break;
  }

  /*------------------------------------------------------------------*
  * Call encipher *
  *------------------------------------------------------------------*/
  printf("Clear Text\n");
  printHex(clearText,sizeof(clearText));
  if ((returnCode = CSNBENC(&returnCode,
           &reasonCode,
           &exitDataLength,
           exitData,
           dataKeyId,
           &textLen,
           clearText,
           ICV,
           &ruleArrayCount,
           cipherProcessRule,
           &pad,
           OCV,
           cipherText)) != 0) {
    printf("Return from Encipher:\n");
    printf(" Return Code = %04d\n",returnCode);
    printf(" Reason Code = %04d\n",reasonCode);
    if (returnCode > 4)
      break;
  }

  /*------------------------------------------------------------------*
  * Call decipher *
  *------------------------------------------------------------------*/
  printf("Cipher Text\n");
  printHex(cipherText,sizeof(cipherText));

/* Variable inputs/outputs for ICSF services */
unsigned char cipherText[24]=0;
unsigned char compareText[24]=0;
unsigned char dataKeyId[64]=0;
unsigned char nullKeyId[64]=0;
unsigned char dummyKEKKeyId1[64]=0;
unsigned char dummyKEKKeyId2[64]=0;
int returnCode = 0;
int reasonCode = 0;
unsigned char OCV[18]=0;
if ((returnCode = CSNBDEC(&returnCode,
        &reasonCode,
        &exitDataLength,
        exitData,
        dataKeyId,
        &textLen,
        cipherText,
        ICV,
        &ruleArrayCount,
        cipherProcessRule,
        OCV,
        compareText)) != 0) {
    printf("\nReturn from Decipher: \n");
    printf(" Return Code = %04d\n",returnCode);
    printf(" Reason Code = %04d\n",reasonCode);
    if (returnCode > 4)
        break;
}
/*---------------------------------------------------------------*
* End                                                          *
*---------------------------------------------------------------*/
printf("\nClear Text after decipher\n");
printHex(compareText,sizeof(compareText));
} while(0);

return returnCode;
} /* end main */

void printHex (unsigned char * text, unsigned int len)
/*------------------------------------------------------------------*
* Prints a string as hex characters                                  *
*------------------------------------------------------------------*/
{
    unsigned int i;
    for (i = 0; i < len; ++i)
        if ((i & 7) == 7 || (i == (len - 1)))
            printf (" %02x\n", text[i]);
        else
            printf (" %02x", text[i]);
    printf ("\n");
} /* end printHex */

COBOL

**************************
IDENTIFICATION DIVISION.
**************************
PROGRAM-ID. COBOLXMP.
****************************
ENVIRONMENT DIVISION.
*****************************************************************
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370.
OBJECT-COMPUTER. IBM-370.
***************************
DATA DIVISION.
*****************************************************************
FILE SECTION.
WORKING-STORAGE SECTION.
77 INPUT-TEXT PIC X(24)
    VALUE 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321'.
************* DEFINE SAPI INPUT/OUTPUT PARAMETERS *************
01 SAPI-REC.
   05 RETURN-CODE-S PIC 9(08) COMP.
   05 REASON-CODE-S PIC 9(08) COMP.
   05 EXIT-DATA-LENGTH-S PIC 9(08) COMP.
   05 EXIT-DATA-S PIC X(04).
   05 KEK-KEY-ID-1-S PIC X(64)
      VALUE LOW-VALUES.
   05 KEK-KEY-ID-2-S PIC X(64)
      VALUE LOW-VALUES.
   05 DATA-KEY-ID-S PIC X(64)
      VALUE LOW-VALUES.
   05 NULL-KEY-ID-S PIC X(64)
      VALUE LOW-VALUES.
   05 KEY-FORM-S PIC X(08).
   05 KEY-LENGTH-S PIC X(08).
   05 DATA-KEY-TYPE-S PIC X(08).
   05 NULL-KEY-TYPE-S PIC X(08).
   05 TEXT-LENGTH-S PIC 9(08) COMP.
   05 TEXT-S PIC X(24).
   05 ICV-S PIC X(08).
   05 PAD-S PIC X(01).
   05 CPHR-TEXT-S PIC X(24).
   05 COMP-TEXT-S PIC X(24).
   05 RULE-ARRAY-COUNT-S PIC 9(08) COMP.
   10 RULE-ARRAY PIC X(08).
      10 RULE-ARRAY PIC X(08).
      10 CHAINING-VECTOR-S PIC X(18).
********************************************************************************
PROCEDURE DIVISION.
********************************************************************************
MAIN-RTN.
************* CALL KEY GENERATE ***************************************
   MOVE 0 TO EXIT-DATA-LENGTH-S.
   MOVE KEY-FORM TO KEY-FORM-S.
   MOVE KEY-LENGTH TO KEY-LENGTH-S.
   MOVE KEY-TYPE-1 TO DATA-KEY-TYPE-S.
   MOVE KEY-TYPE-2 TO NULL-KEY-TYPE-S.
   CALL 'CSNBKGN' USING RETURN-CODE-S
      REASON-CODE-S
      EXIT-DATA-LENGTH-S
      EXIT-DATA-S
      KEY-FORM-S
      KEY-LENGTH-S
      DATA-KEY-TYPE-S
      NULL-KEY-TYPE-S
      KEK-KEY-ID-1-S
      KEK-KEY-ID-2-S
      DATA-KEY-ID-S
NULL-KEY-ID-S.
IF RETURN-CODE-S NOT = 0 OR
REASON-CODE-S NOT = 0 THEN
DISPLAY '*** KEY-GENERATE ***'
DISPLAY '*** RETURN-CODE = ' RETURN-CODE-S
DISPLAY '*** REASON-CODE = ' REASON-CODE-S
ELSE
MOVE 24 TO TEXT-LENGTH-S
MOVE INPUT-TEXT TO TEXT-S
MOVE 1 TO RULE-ARRAY-COUNT-S
MOVE CIPHER-PROCESSING-RULE TO RULE-ARRAY-S
MOVE LOW-VALUES TO CHAINING-VECTOR-S
MOVE ICV TO ICV-S.
MOVE PAD TO PAD-S.
************* CALL ENCIPHER *******************************
CALL 'CSNBENC' USING RETURN-CODE-S
REASON-CODE-S
EXIT-DATA-LENGTH-S
EXIT-DATA-S
DATA-KEY-ID-S
TEXT-LENGTH-S
TEXT-S
ICV-S
RULE-ARRAY-COUNT-S
RULE-ARRAY-S
PAD-S
CHAINING-VECTOR-S
CPHR-TEXT-S
IF RETURN-CODE-S NOT = 0 OR
REASON-CODE-S NOT = 0 THEN
DISPLAY '*** ENCIPHER ***'
DISPLAY '*** RETURN-CODE = ' RETURN-CODE-S
DISPLAY '*** REASON-CODE = ' REASON-CODE-S
ELSE
************* CALL DECIPHER *******************************
CALL 'CSNBDEC' USING RETURN-CODE-S
REASON-CODE-S
EXIT-DATA-LENGTH-S
EXIT-DATA-S
DATA-KEY-ID-S
TEXT-LENGTH-S
CPHR-TEXT-S
ICV-S
RULE-ARRAY-COUNT-S
RULE-ARRAY-S
CHAINING-VECTOR-S
COMP-TEXT-S
IF RETURN-CODE-S NOT = 0 OR
REASON-CODE-S NOT = 0 THEN
DISPLAY '*** DECIPHER ***'
DISPLAY '*** RETURN-CODE = ' RETURN-CODE-S
DISPLAY '*** REASON-CODE = ' REASON-CODE-S
ELSE
IF COMP-TEXT-S = TEXT-S THEN
DISPLAY '*** DECIPHERED TEXT = PLAIN TEXT ***'
ELSE
DISPLAY '*** DECIPHERED TEXT ê= PLAIN TEXT ***'.
DISPLAY '*** TEST PROGRAM ENDED ***'
STOP RUN.

Assembler H

TITLE 'SAMPLE ENCIPHER/DECIPHER S/370 PROGRAM.'
******************************************************************************
* SYSTEM/370 ASSEMBLER H EXAMPLE *
*                                *
******************************************************************************
SPACE

SAMPLE START 0
DS 0H
STM 14,12,12(13) SAVE REGISTERS
BALR 12,0 USE R12 AS BASE REGISTER
USING *,12 PROVIDE SAVE AREA FOR SUBROUTINE
LA 14,SAVE PERFORM SAVE AREA CHAINING
ST 13,4(14) 
ST 14,8(13) 
LR 13,14 

* CALL CSFKGN,(RETCD, 
  RESCD, 
  EXDATA, 
  EXDATA, 
  KEY_FORM, 
  KEY_LEN, 
  KEY_TYP1, 
  KEY_TYP2, 
  KEK_ID1, 
  KEK_ID2, 
  DATA_ID, 
  NULL_ID)
  CLC RETCD,=F'0' CHECK RETURN CODE
  BNE BACK OUTPUT RETURN/REASON CODE AND STOP
  CLC RESCD,=F'0' CHECK REASON CODE
  BNE BACK OUTPUT RETURN/REASON CODE AND STOP

* CALL ENCIIPHER WITH THE KEY JUST GENERATED
* OPERATIONAL FORM
*
  MVC RULEAC,=F'1' SET RULE ARRAY COUNT
  MVC RULEA,=CL8'CUSP ' BUILD RULE ARRAY
  CALL CSFENC,(RETCD, 
    RESCD, 
    EXDATA, 
    EXDATA, 
    DATA_ID, 
    TEXTL, 
    TEXT, 
    ICV, 
    RULEAC, 
    RULEA, 
    PAD_CHAR, 
    OCV, 
    CIPHER_TEXT)
  CLC RETCD,=F'0' CHECK RETURN CODE
  BNE BACK OUTPUT RETURN/REASON CODE AND STOP
  CLC RESCD,=F'0' CHECK REASON CODE
  BNE BACK OUTPUT RETURN/REASON CODE AND STOP
CALL CSFDEC,(RETCD, 
  RESCD, 
  EXDATA, 
  EXDATA, 
  DATA_ID, 
  TEXTL, 
  CIPHER_TEXT, 
  ICV, 
  RULEAC, 
  RULEA, 
  OCV, 
  NEW_TEXT)
  CLC RETCD,=F'0' CHECK RETURN CODE
  BNE BACK OUTPUT RETURN/REASON CODE AND STOP
  CLC RESCD,=F'0' CHECK REASON CODE
  BNE BACK OUTPUT RETURN/REASON CODE AND STOP

*
COMPARE EQU *  
CLC TEXT,NEW_TEXT
BE GOODENC
WTO 'DECRYPTED TEXT DOES NOT MATCH STARTING TEXT'
B BACK
GOODENC WTO 'DECRYPTED TEXT MATCHES STARTING TEXT'
*
*
WTO 'TEST PROGRAM TERMINATING'
B RETURN
*
*----------------------------------------------------
* CONVERT RETURN/REASON CODES FROM BINARY TO EBCDIC
*----------------------------------------------------
BACK DS 0F  OUTPUT RETURN & REASON CODE
L 5,RETCD  LOAD RETURN CODE
L 6,RESCD  LOAD REASON CODE
CVD 5,BCD1  CONVERT TO PACK-DECIMAL
CVD 6,BCD2
UNPK ORETCD,BCD1  CONVERT TO EBCDIC
UNPK ORESCD,BCD2
OI ORETCD+7,X'F0'  CORRECT LAST DIGIT
OI ORESCD+7,X'F0'
*
MVC ERROUT+21(4),ORETCD+4
MVC ERROUT+41(4),ORESCD+4
ERROUT WTO 'ERROR CODE = ', REASON CODE = '
RETURN EQU *
L 13,4(13)  SAVE AREA RESTORATION
LM 14,12,12(13)
BR 14  RETURN TO CALLER
*
BCD1 DS D  CONVERT TO BCD TEMP AREA
BCD2 DS D  CONVERT TO BCD TEMP AREA
ORETCD DS CL8'0'  OUTPUT RETURN CODE
ORESCD DS CL8'0'  OUTPUT REASON CODE
*
KEY_FORM DC CL8'OP '  KEY FORM
KEY_LEN DC CL8'SINGLE '  KEY LENGTH
KEY_TYP1 DC CL8'DATA '  KEY TYPE 1
KEY_TYP2 DC CL8' '  KEY TYPE 2
TEXT DC C'ABCDEFGHIJKLMNOPQRSTUV0987654321'
TEXTL DC F'32'  TEXT LENGTH
CIPHER_TEXT DC CL32'
NEW_TEXT DC CL32'
DATA_ID DC XL6'00'  DATA KEY TOKEN
NULL_ID DC XL6'00'  NULL KEY TOKEN - UNFILLED
KEK_ID1 DC XL6'00'  KEK1 KEY TOKEN
KEK_ID2 DC XL6'00'  KEK2 KEY TOKEN
RETCODE DS F'0'  RETURN CODE
RESCD DS F'0'  REASON CODE
EXDATAL DC F'0'  EXIT DATA LENGTH
EXDATA DS 0C  EXIT DATA
RULEA DS 1CL8  RULE ARRAY
RULEAC DS F'0'  RULE ARRAY COUNT
ICV DC XL8'00'  INITIAL CHAINING VECTOR
OCV DC XL18'00'  OUTPUT CHAINING VECTOR
PAD_CHAR DC F'0'  PAD CHARACTER
SAVE DS 18F  SAVE REGISTER AREA
END SAMPLE
Sample program to call the one-way hash service to generate
the SHA-1 hash of the input text and call digital signature
generate with an RSA key using the ISO 9796 text formatting. The
RSA key token is built from supplied data and imported for the
signature generate service to use.

INPUT: TEXT Message digest to be signed

OUTPUT: SIGNATURE_LENGTH Length of the signature in bytes
Written to a dataset.

SIGNATURE Signature for hash. Written to a dataset.

***************************************************************
DSIGEXP:PROCEDURE( TEXT ) OPTIONS( MAIN );

/* Declarations - Parameters */
DCL TEXT CHAR( 64 ) VARYING;

/* Declarations - API parameters */
DCL CHAINING_VECTOR_LENGTH FIXED BINARY( 31, 0 ) INIT( 128 );
DCL CHAINING_VECTOR CHAR( 128 );
DCL DUMMY_KEY CHAR( 64 );
DCL EXIT_DATA CHAR( 4 );
DCL EXIT_LEN FIXED BINARY( 31, 0 ) INIT( 0 );
DCL HASH CHAR( 20 );
DCL HASH_LENGTH FIXED BINARY( 31 , 0 ) INIT( 20 );
DCL INTERNAL_PKA_TOKEN CHAR( 1024 );
DCL INTERNAL_PKA_TOKEN_LENGTH FIXED BINARY( 31, 0 );
DCL KEY_VALUE_STRUCTURE CHAR(139)
INIT(( '02000040000300400000000000000000'X ||
 '01AE28DA4666D085EB7E034006BAC51'X ||
 '991C0C0DEAE835AD97E7E7E7EA741'X ||
 '41DAA0D2A431A16F970100010252BDAD42'X ||
 '52BDAD425AC6045041AF746EBD5F'X ||
 '085D574CF9C07F0838CC5017C2A1A'X ||
 'B919D2551350A76606BFA6AF2B1609A'X ||
 '000A4B0D19A559CA80'X )));
DCL KEY_VALUE_LENGTH FIXED BINARY( 31 , 0 ) INIT( 139 );
DCL OWH_TEXT CHAR( 64 );
DCL PKA_KEY_TOKEN CHAR( 1024 );
DCL PKA_TOKEN_LENGTH FIXED BINARY( 31, 0 );
DCL PRIVATE_NAME CHAR( 64 ) INIT( 'PL1.EXAMPLE.FOR.APG' );
DCL PRIVATE_NAME_LENGTH FIXED BINARY( 31, 0 ) INIT( 0 );
DCL RETURN_CODE FIXED BINARY( 31, 0 ) INIT( 0 );
DCL REASON_CODE FIXED BINARY( 31, 0 ) INIT( 0 );
DCL RESERVED_FIELD_LENGTH FIXED BINARY( 31 , 0 ) INIT( 0 );
DCL RESERVED_FIELD CHAR( 1 );
DCL RULE_ARY_CNT_DSG FIXED BINARY( 31, 0 ) INIT( 1 );
DCL RULE_ARY_CNT_PKB FIXED BINARY( 31, 0 ) INIT( 1 );
DCL RULE_ARY_CNT_PKI FIXED BINARY (31, 0) INIT(0);
DCL RULE_ARY_CNT_OWH FIXED BINARY (31, 0) INIT(2);
DCL RULE_ARY_DSG CHAR(8) INIT('ISO-9796');
DCL RULE_ARY_PKB CHAR(8) INIT('RSA-PRIV');
DCL RULE_ARY_PKB CHAR(8);
DCL RULE_ARY_No WH CHAR(16) INIT('SHA-1 ONLY ');
DCL SIGNATURE_LENGTH FIXED BINARY (31, 0);
DCL SIGNATURE CHAR (128);
DCL SIG_BIT_LENGTH FIXED BINARY (31, 0);
DCL TEXT_LENGTH FIXED BINARY (31, 0);

/* Declarations - Files and entry points */
DCL SYSPRINT FILE OUTPUT;
DCL SIGOUT FILE RECORD OUTPUT;
DCL CSNDPKB ENTRY EXTERNAL OPTIONS(ASM, INTER);
DCL CSNDPKI ENTRY EXTERNAL OPTIONS(ASM, INTER);
DCL CSNBOWH ENTRY EXTERNAL OPTIONS(ASM, INTER);
DCL CSNDDSG ENTRY EXTERNAL OPTIONS(ASM, INTER);

/* Declarations - Internal variables */
DCL DSG_HEADER CHAR (32)
   INIT('DIGITAL SIGNATURE GENERATION *');
DCL FILE_OUT_LINE CHAR (128);
DCL OWH_HEADER CHAR (16)
   INIT('ONE WAY HASH *');
DCL PKB_HEADER CHAR (16)
   INIT('PKA TOKEN BUILD *');
DCL PKI_HEADER CHAR (16)
   INIT('PKA TOKEN IMPORT *');
DCL RC_STRING CHAR (14)
   INIT('RETURN CODE =');
DCL RS_STRING CHAR (14)
   INIT('REASON CODE =');
DCL SIG_STRING CHAR (12)
   INIT('SIGNATURE =');
DCL SIG_LEN_STRING CHAR (26)
   INIT('SIGNATURE LENGTH (BYTES) =');

/* Declarations - Built-in functions */
DCL (SUBSTR, LENGTH) BUILTIN;

/************************************************************/
/* Call one-way hash to get the SHA-1 hash of the text. */
/* TEXT_LENGTH = LENGTH (TEXT ); */
/* OWH_TEXT = SUBSTR( TEXT, 1, TEXT_LENGTH ); */
CALL CSNBOWH (RETURN_CODE,
   REASON_CODE,
   EXIT_LEN,
   EXIT_DATA,
   RULE_ARY_CNT_OWH,
   RULE_ARY_OWH,
   TEXT_LENGTH,
   OWH_TEXT,
   CHAINING_VECTOR_LENGTH,
   CHAINING_VECTOR,
   HASH_LENGTH,
   HASH );

PUT SKIP LIST( OWH_HEADER );
PUT SKIP LIST( RC_STRING || RETURN_CODE );
PUT SKIP LIST( RS_STRING || REASON_CODE );

/************************************************************/
/* Create the PKA RSA private external token. */
/***************************************************************************/
IF RETURN_CODE = 0 THEN
  DO;
    PKA_TOKEN_LENGTH = 1024;
    CALL CSNDPKB( RETURN_CODE,
                  REASON_CODE,
                  EXIT_LEN,
                  EXIT_DATA,
                  RULE_ARY_CNT_PKB,
                  RULE_ARY_PKB,
                  KEY_VALUE_LENGTH,
                  KEY_VALUE_STRUCTURE,
                  PRIVATE_NAME_LENGTH,
                  PRIVATE_NAME,
                  RESERVED_FIELD_LENGTH,
                  RESERVED_FIELD,
                  RESERVED_FIELD_LENGTH,
                  RESERVED_FIELD,
                  RESERVED_FIELD_LENGTH,
                  RESERVED_FIELD,
                  RESERVED_FIELD_LENGTH,
                  RESERVED_FIELD,
                  RESERVED_FIELD_LENGTH,
                  RESERVED_FIELD,
                  RESERVED_FIELD_LENGTH,
                  RESERVED_FIELD,
                  PKA_TOKEN_LENGTH,
                  PKA_KEY_TOKEN );
    PUT SKIP LIST( PKB_HEADER);
    PUT SKIP LIST( RC_STRING || RETURN_CODE );
    PUT SKIP LIST( RS_STRING || REASON_CODE );
  END;
/***************************************************************************/
/* Import the clear RSA private external token. */
/***************************************************************************/
IF RETURN_CODE = 0 THEN
  DO;
    INTERNAL_PKA_TOKEN_LENGTH = 1024;
    CALL CSNDPKI( RETURN_CODE,
                 REASON_CODE,
                 EXIT_LEN,
                 EXIT_DATA,
                 RULE_ARY_CNT_PKI,
                 RULE_ARY_PKI,
                 PKA_TOKEN_LENGTH,
                 PKA_KEY_TOKEN,
                 DUMMY_KEK,
                 INTERNAL_PKA_TOKEN_LENGTH,
                 INTERNAL_PKA_TOKEN );
    PUT SKIP LIST( PKI_HEADER );
    PUT SKIP LIST( RC_STRING || RETURN_CODE );
    PUT SKIP LIST( RS_STRING || REASON_CODE );
  END;
/***************************************************************************/
/* Call digital signature generate. */
/***************************************************************************/
IF RETURN_CODE = 0 THEN
  DO;
SIGNATURE_LENGTH = 128;

CALL CSNDDSG( RETURN_CODE,
    REASON_CODE,
    EXIT_LEN,
    EXIT_DATA,
    RULE_ARY_CNT_DSG,
    RULE_ARY_DSG,
    INTERNAL_PKA_TOKEN_LENGTH,
    INTERNAL_PKA_TOKEN,
    HASH_LENGTH,
    HASH,
    SIGNATURE_LENGTH,
    SIG_BIT_LENGTH,
    SIGNATURE);

PUT SKIP LIST( DSG_HEADER );
PUT SKIP LIST( RC_STRING || RETURN_CODE );
PUT SKIP LIST( RS_STRING || REASON_CODE );

IF RETURN_CODE = 0 THEN
    DO;
        /****************************************************************/
        /* Write the signature and its length to the output file. */
        /****************************************************************/
        FILE_OUT_LINE = SIG_LEN_STRING || SIGNATURE_LENGTH;
        WRITE FILE(SIGOUT) FROM( FILE_OUT_LINE );
        FILE_OUT_LINE = SIG_STRING || SIGNATURE;
        WRITE FILE(SIGOUT) FROM( FILE_OUT_LINE );
        END;
    END;

END DSIGEXP;

Appendix D. Coding Examples  659
Appendix E. Using ICSF with BSAFE

ICSF works in conjunction with RSA Security, Inc.’s BSAFE toolkit (BSAFE 3.1 or later). If you are currently using applications developed with BSAFE, we strongly recommend you take advantage of the increased security and performance available with ICSF interfaces. The BHAPI interface has been stabilized since ICSF FMID HCR770B and may be removed in a future release.

Through BSAFE 3.1 you can access the ICSF services to:

- Compute message digests or hashes
- Generate random numbers
- Encipher and decipher data using the DES algorithm
- Generate and verify RSA digital signatures

Some BSAFE Basics

BSAFE has many algorithm information types (called AIs). Many of the AIs can perform several cryptographic functions. For this reason, you must specify the algorithmic method (AM) to be used by supplying a chooser. If the cryptographic function requires a key, you supply key information to the BSAFE application with a key information (KI) type. For the most current information on the BSAFE user interface and a complete description of algorithm information types, algorithm methods, choosers, and key information types, refer to *BSAFE User’s Manual* and *BSAFE Library Reference Manual*.

Computing Message Digests and Hashes

MD5 and SHA1 hashing are both available from ICSF via BSAFE. If your BSAFE application uses the AM_MD5 or the AM_SHA algorithm methods, you can add a couple of BSAFE function calls and the application will use ICSF and the Cryptographic Coprocessor Feature instead of the BSAFE algorithm method.

The following list shows BSAFE AI types with choosers that may include AM_MD5:
- AI_MD5
- AI_MD5_BER
- AI_MD5WithDES_CBCPad
- AI_MD5WithDES_CBCPadBER
- AI_MD5WithRC2_CBCPad
- AI_MD5WithRC2_CBCPadBER
- AI_MD5WithRSAEncryption
- AI_MD5WithRSAEncryptionBER
- AI_MD5WithXOR
- AI_MD5WithXOR_BER

The following list shows BSAFE AI types with choosers that may include AM_SHA:
- AI_SHA1
- AI_SHA1_BER
- AI_SHA1WithDES_CBCPad
- AI_SHA1WithDES_CBCPadBER

Generating Random Numbers

If your BSAFE application uses the algorithm method AM_MD5RANDOM, you can add a chooser definition containing the algorithm method AM_HWRANDOM (new...
with BSAFE 3.1) and a couple of BSAFE function calls and your program can use ICSF and the Cryptographic Coprocessor Feature to generate random numbers instead of the BSAFE algorithm method.

BSAFE 3.1 provides a new algorithm information type, AI_HWRandom. You need to set your random number generation object with AI_HWRandom, and initialize the object with a chooser containing AM_HW_RANDOM, in order to use ICSF with the Cryptographic Coprocessor Feature for generating random numbers. You do not, however, have to make a B_RandomUpdate call, since the S/390 and IBM zSeries cryptographic solution does not require a seed.

The only AI type with choosers that may include AM_HW_RANDOM is AI_HWRandom.

**Encrypting and Decrypting with DES**

If your BSAFE application uses either the AM_DES_CBC_ENCRYPT or the AM_DES_CBC_DECRYPT algorithm methods, you can add a chooser containing the algorithm methods AM_TOKEN_DES_CBC_ENCRYPT and/or AM_TOKEN_DES_CBC_DECRYPT (both new with BSAFE 3.1) and a couple of BSAFE function calls and your program can use ICSF and the Cryptographic Coprocessor Feature to encrypt and/or decrypt data using the DES algorithm.

For your encryption or decryption key, you can use either a clear key in the form of a KI_8Byte or KI_DES8 or KI_Item (8 bytes long), or a CCA DES Key Token in the form of a KI_TOKEN (64 bytes long). KI_TOKEN is a new key information type in BSAFE 3.1.

The following list shows BSAFE AI types with choosers that may include either AM_TOKEN_DES_CBC_ENCRYPT, AM_TOKEN_DES_CBC_DECRYPT, or both:

- AI_DES_CBC_BSAFE1
- AI_DES_CBC_IV8
- AI_DES_CBCPadBER
- AI_DES_CBCPadIV8
- AI_DES_CBCPadPEM
- AI_MD5WithDES_CBCPad
- AI_MD5WithDES_CBCPadBER
- AI_SHA1WithDES_CBCPad
- AI_SHA1WithDES_CBCPadBER

**Generating and Verifying RSA Digital Signatures**

You can use algorithm method AM_TOKEN_RSA_PRV_ENCRYPT with AM_MD5 or AM_SHA to have ICSF and the Cryptographic Coprocessor Feature generate RSA digital signatures. To verify the RSA digital signature using the S/390 or IBM zSeries cryptographic solution, you can use AM_TOKEN_RSA_PUB_DECRYPT (with AM_MD5 or AM_SHA). Your BSAFE application must contain a couple of new BSAFE function calls to access the S/390 and IBM zSeries services. AM_TOKEN_RSA_PRV_ENCRYPT and AM_TOKEN_RSA_PUB_DECRYPT are new in BSAFE 3.1. For more information, see “Using the New Function Calls in Your BSAFE Application” on page 663.

For signature generation, you can use either a clear private key in the form of a KI_PKCS_RSAPrivate or a CCA RSA private key token in the form of a KI_TOKEN. For signature verification, you can use either a public RSA key in the form of a KI_RSAPublic or a CCA RSA public key token in the form of a KI_TOKEN.
KI_TOKEN is a new key information type in BSAFE. For more information about KI_TOKEN, see "Using the BSAFE KI_TOKEN" on page 665.

The following list shows BSAFE AI types with choosers that may include AM_TOKEN_RSA_PRV_ENCRYPT:
- AI_MD5WithRSAEncryption
- AI_MD5WithRSAEncryptionBER
- AI_SHA1WithRSAEncryption
- AI_SHA1WithRSAEncryptionBER

The following list shows BSAFE AI types with choosers that may include AM_TOKEN_RSA_PUB_DECRYPT:
- AI_MD5WithRSAEncryption
- AI_SHA1WithRSAEncryption

Encrypting and Decrypting with RSA

You can use algorithm method AM_TOKEN_RSA_ENCRYPT to have ICSF encrypt a symmetric key (or other string of 48 bytes or fewer). To decrypt the string using ICSF, you can use AM_TOKEN_RSA_CRT_DECRYPT. You'll need a couple of new BSAFE function calls to access the S/390 and IBM @server zSeries services (see "Using the New Function Calls in Your BSAFE Application."

To encrypt a string, you can use either a public key in the form KI_RSAPublic or a CCA RSA public key token in the form of a KI_TOKEN.

To decrypt a string, you can use either a private key in the form KI_PKCS_RSAPrivate or a CCA RSA private key token in the form of a KI_TOKEN.

Using the New Function Calls in Your BSAFE Application

To have your BSAFE application access the ICSF, S/390, and IBM @server zSeries Cryptographic Coprocessor Feature services, you need to add several new elements to your program. These elements are explained with examples in the steps that follow.

1. At the beginning of your program, declare one or more session choosers and also the hardware table list. For information about choosers and the hardware table list, see BSAFE User's Manual.

   /*-------------------------------------------------------------*/
   /* SESSION_CHOOSER will replace OLD_CHOOSER. */
   /**SESSION_CHOOSER = NULL_PTR;*/
   /*----------------------------------------------------------------*/
   B_ALGORITHM_METHOD **SESSION_CHOOSER = NULL_PTR;

   /*----------------------------------------------------------------*/
   /* CCA_VTABLE is a vector table of functions that will be */
   /* substituted for BSAFE equivalents. It is supplied by IBM */
   /* and will be loaded into your application when you invoke */
   /* QueryCrypto. */
   /*----------------------------------------------------------------*/
   HW_TABLE_LIST CCA_VTABLE = (HW_TABLE_LIST)NULL_PTR;

2. Declare a tag list. The content of the tag list is supplied by BSAFE at the B_CreateSessionChooser call, which is discussed in a later step.

   unsigned char **taglist = (unsigned char **)NULL_PTR;

3. For random number generation, DES encryption or decryption or RSA encryption or decryption, you need to define and declare an additional chooser
wherever your current chooser is defined and declared. For instance, suppose your application is doing an RSA encryption, and OLD_CHOOSER is defined as follows:

```c
/*--------------------------------------------------------------*
* OLD_CHOOSER is used for this application when ICSF and the crypto hardware is not available. *
*--------------------------------------------------------------*/
B_ALGORITHM_METHOD *OLD_CHOOSER[] = {
  &AM_SHA,
  &AM_RSA_ENCRYPT,
  (B_ALGORITHM_METHOD *)NULL_PTR
};
/*--------------------------------------------------------------*
* ICSF_CHOOSER is a 'skeleton' for SESSION_CHOOSER. *
* SESSION_CHOOSER will be used for this application if ICSF and the crypto hardware are not available. *
*--------------------------------------------------------------*/
B_ALGORITHM_METHOD *ICSF_CHOOSER[] = {
  &AM_SHA,
  &AM_TOKEN_RSA_PUB_ENCRYPT,
  (B_ALGORITHM_METHOD *)NULL_PTR
};
```

4. At the beginning of the main function in your application, add a call to the ICSF QueryCrypto function followed by a conditional call to the BSAFE B_CreateSessionChooser function.

```c
if ((status = QueryCrypto(CRYPTO_Q_DES_AND_RSA,&CCA_VTABLE)) == 0)
/*-------------------------------------------------------------*
* B_CreateSessionChooser will replace the BSAFE software functions with their CCA equivalents. *
* Note that the last three parameters are not used with CCA *
*-------------------------------------------------------------*/
if ((status = B_CreateSessionChooser(ICSF_CHOOSER,
  &SESSION_CHOOSER,
  CCA_VTABLE,
  (ITEM *)NULL_PTR,
  (POINTER *)NULL_PTR,
  &taglist)) != 0)
break;
```

5. Set up the conditions under which any alternate choosers are used to initialize the appropriate algorithm object. For information about initializing algorithm objects, see BSAFE User's Manual.

```c
/*-------------------------------------------------------------*
* Initialize the algorithm object with the appropriate chooser. *
*-------------------------------------------------------------*/
if (SESSION_CHOOSER != NULL_PTR)
if ((status = B_xxxxxxInit
  (xxxxxxObject,SESSION_CHOOSER,
   (A_SURRENDER_CTX *)NULL_PTR)) != 0)
  break;
else ;
```
6. When your application no longer needs the session chooser, program a call to the BSAFE B_FreeSessionChooser function.

```c
if (SESSION_CHOOSER != NULL_PTR)
    B_FreeSessionChooser(&SESSION_CHOOSER,&taglist);
```

### Using the BSAFE KI_TOKEN

Those ICSF functions that require a key, like encipher and decipher, expect the key in the form of a CCA token. If you already have a CCA token, perform the following steps before you try to set your algorithm object. For information about how to perform the following tasks, see *BSAFE User's Manual* and *BSAFE Library Reference Manual*.

1. Create a key object.
2. Declare a KEY_TOKEN_INFO and fill it in.
   - `KEY_TOKEN_INFO` is defined as follows in the *BSAFE User's Manual*:
     ```c
typedef struct {
    ITEM manufacturerID;
    ITEM internalKeyInfo;
} KEY_TOKEN_INFO;
```
   - The first ITEM is the address and length of one of the following three strings, depending on the CCA key token type you are using:
     - `com.ibm.CCADES`
     - `com.ibm.CCARSAPublic`
     - `com.ibm.CCARSAPrivate`
   - The second ITEM is the address and length of your CCA key token.
3. Set the key information (B_SetKeyInfo) into the key object using the item and a key information type of KI_TOKEN as input.

If you don't already have a CCA token, you can supply a clear key to the function using one of the key information types mentioned in the section discussing the function you are using. BSAFE will convert the key to a CCA token. If you supply a clear BSAFE KI type to one of the ICSF functions, and the function is performed successfully, you can retrieve the key as a CCA token by invoking B_GetKeyInfo with KI_TOKEN as the key information type. A KEY_TOKEN_INFO struct is returned.

### ICSF Triple DES via BSAFE

ICSF performs single, double, or triple DES depending on the length of the DES key; if you're using BSAFE to access ICSF triple DES, you should use the algorithm methods `AM_TOKEN_DES_CBC_ENCRYPT` and `AM_TOKEN_DES_CBC_DECRYPT`.

If you've already have an ICSF token, follow the instructions in the section titled "Using the BSAFE KI_TOKEN."

If you're using a clear key, follow the same procedure, except use your clear key padded on the right with binary zeroes to a length of 64 as the internalKeyInfo part of your KI_TOKEN_INFO. ICSF will convert your clear key to an internal ICSF key token.
Here's an example:

```c
B_KEY_OBJ desKey = (B_KEY_OBJ)NULL_PTR;
KEY_TOKEN_INFO myTokenInfo;
unsigned char myToken[64] = {0};
unsigned char * myTokenP;
unsigned char myDoubleKey[16]; /* Input to this function */
unsigned char mfgID[] = "com.ibm.CCADES";
unsigned char * mfgIDP;

myTokenP = myToken;
mfgIDP = mfgID;
T_memcpy(myToken,myDoubleKey,sizeof(myDoubleKey));
myTokenInfo.manufacturerID.len = strlen(mfgID);
myTokenInfo.manufacturerID.data = mfgIDP;
myTokenInfo.internalKeyInfo.len = sizeof(myToken);
myTokenInfo.internalKeyInfo.data = myTokenP;

/* Create a key object. */
if ((status = B_CreateKeyObject (&desKey)) != 0)
    break;

/* Set the key object. */
if ((status = B_SetKeyInfo(
    desKey, KI_TOKEN, myTokenInfo )) != 0)
    break;
```

### Retrieving ICSF Error Information

When using the ICSF and Cryptographic Coprocessor Feature, Init, Update, and Final calls can result in BSAFE returning a status of BE_HARDWARE (0x020B). When this occurs, you can derive the ICSF return and reason codes by using a new BSAFE operation, B_GetExtendedErrorInfo. For an explanation of the return codes and reason codes, see Appendix A, "ICSF and TSS Return and Reason Codes," on page 557.

A coding example follows.

```c
#include "balg.h"
#include "algobj.h"
#include "cca.h"

{
    B_ALGORITHM_OBJECT * aop;
    ITEM * errp;
    unsigned char * algorithmMethod;
    CCA_ERROR_DATA * edp;
    unsigned int CCAreturnCode=0;
    unsigned int CCAreasonCode=0;
    unsigned char algorithmName[40]= {0x00};

    if (status==BE_HARDWARE) {
        B_GetExtendedErrorInfo(aop, errp, algorithmMethod);
    }
```
edp = errp->data;
CCAreturnCode = (unsigned int) edp->returnCode;
CCAreasonCode = (unsigned int) edp->reasonCode;
}

The prototype for B_GetExtendedErrorInfo is in balg.h, as shown in the example that follows.

B_GetExtendedErrorInfo(
B_ALGORITHM_OBJ algorithmObject, /* in--algorithm object */
ITEM * errorData, /* out--address and length of error data */
POINTER algorithmMethod /* out--address of faulting AM */
);

Appendix E. Using ICSF with BSAFE
Appendix F. Cryptographic Algorithms and Processes

This appendix describes the personal identification number (PIN) formats and algorithms.

PIN Formats and Algorithms

For PIN calculation procedures, see IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface Reference.

PIN Notation

This section describes various PIN block formats. The following notations describe the contents of PIN blocks:

- **P** = A 4-bit decimal digit that is one digit of the PIN value.
- **C** = A 4-bit hexadecimal control value. The valid values are X'0', X'1', and X'2'.
- **L** = A 4-bit hexadecimal value that specifies the number of PIN digits. The value ranges from 4 to 12, inclusive.
- **F** = A 4-bit field delimiter of value X'F'.
- **f** = A 4-bit delimiter filler that is either P or F, depending on the length of the PIN.
- **D** = A 4-bit decimal padding value. All pad digits in the PIN block have the same value.
- **X** = A 4-bit hexadecimal padding value. All pad digits in the PIN block have the same value.
- **x** = A 4-bit hexadecimal filler that is either P or X, depending on the length of the PIN.
- **R** = A 4-bit hexadecimal random digit. The sequence of R digits can each take a different value.
- **r** = A 4-bit random filler that is either P or R, depending on the length of the PIN.
- **Z** = A 4-bit hexadecimal zero (X'0').
- **z** = A 4-bit zero filler that is either P or Z, depending on the length of the PIN.
- **S** = A 4-bit hexadecimal digit that constitutes one digit of a sequence number.
- **A** = A 4-bit decimal digit that constitutes one digit of a user-specified constant.

PIN Block Formats

This section describes the PIN block formats and assigns a code to each format.

**ANSI X9.8**

This format is also named ISO format 0, VISA format 1, VISA format 4, and ECI format 1.

\[
P1 = \text{CLPPPPPPPPPPPPPFF} \\
P2 = \text{ZZZZZZZZZZZZZZZZZZZZZZZZZZZZ} \\
\]
PIN Block = P1 XOR P2

where C = X'0'
L = X'4' to X'C'

Programming Note: The rightmost 12 digits (excluding the check digit) in P2 are the rightmost 12 digits of the account number for all formats except VISA format 4. For VISA format 4, the rightmost 12 digits (excluding the check digit) in P2 are the leftmost 12 digits of the account number.

ISO Format 1
This format is also named ECI format 4.

PIN Block = CLPPPPrrrrrrrrRR

where C = X'1'
L = X'4' to X'C'

ISO Format 2

PIN Block = CLPPPPffffffffFF

where C = X'2'
L = X'4' to X'C'

VISA Format 2

PIN Block = LPPPPzzDDDDDDDD

where L = X'4' to X'6'

VISA Format 3
This format specifies that the PIN length can be 4-12 digits, inclusive. The PIN starts from the leftmost digit and ends by the delimiter ('F'), and the remaining digits are padding digits.

An example of a 6-digit PIN:

PIN Block = PPPPPPPXXXXXXXXX

IBM 4700 Encrypting PINPAD Format
This format uses the value X'F' as the delimiter for the PIN.

PIN Block = LPPPPPPPPPPPPPPFSS

where L = X'4' to X'C'

IBM 3624 Format
This format requires the program to specify the delimiter, X, for determining the PIN length.

PIN Block = PPPxxxxxxxxxxxx

IBM 3621 Format
This format requires the program to specify the delimiter, X, for determining the PIN length.

PIN Block = SSSSSPPPxxxxxxxx

ECI Format 2
This format defines the PIN to be 4 digits.

PIN Block = PPPPPPPPPPPPPPPPP

670 z/OS V1R11 ICSF Application Programmer's Guide
ECI Format 3

PIN Block = LPPPPzzRRRRRRRRR

where \( L = \text{X}'4' \) to \( \text{X}'6' \)

PIN Extraction Rules

This section describes the PIN extraction rules for the Encrypted PIN verify and Encrypted PIN translate callable services.

Encrypted PIN Verify Callable Service

The service extracts the customer-entered PIN from the input PIN block according to the following rules:

- If the input PIN block format is ANSI X9.8, ISO format 0, VISA format 1, VISA format 4, ECI format 1, ISO format 1, ISO format 2, VISA format 2, IBM Encrypting PINPAD format, or ECI format 3, the service extracts the PIN according to the length specified in the PIN block.
- If the input PIN block format is VISA format 3, the specified delimiter (padding) determines the PIN length. The search starts at the leftmost digit in the PIN block. If the input PIN block format is 3624, the specification of a PIN extraction method for the 3624 is supported through rule array keywords. If no PIN extraction method is specified in the rule array, the specified delimiter (padding) determines the PIN length.
- If the input PIN block format is 3621, the specification of a PIN extraction method for the 3621 is supported through rule array keywords. If no PIN extraction method is specified in the rule array, the specified delimiter (padding) determines the PIN length.
- If the input PIN block format is ECI format 2, the PIN is the leftmost 4 digits. For the VISA algorithm, if the extracted PIN length is less than 4, the service sets a reason code that indicates that verification failed. If the length is greater than or equal to 4, the service uses the leftmost 4 digits as the referenced PIN.
- For the IBM German Banking Pool algorithm, if the extracted PIN length is not 4, the service sets a reason code that indicates that verification failed.
- For the IBM 3624 algorithm, if the extracted PIN length is less than the PIN check length, the service sets a reason code that indicates that verification failed.

Clear PIN Generate Alternate Callable Service

The service extracts the customer-entered PIN from the input PIN block according to the following rules:

- This service supports the specification of a PIN extraction method for the 3624 and 3621 PIN block formats through the use of the rule array keyword. Rule array points to an array of one or two 8-byte elements. The first element in the rule array specifies the PIN calculation method. The second element in the rule array (if specified) indicates the PIN extraction method. Refer to the “Clear PIN Generate Alternate (CSNBCPA)” on page 317 for an explanation of PIN extraction method keywords.

Encrypted PIN Translate Callable Service

The service extracts the customer-entered PIN from the input PIN block according to the following rules:

- If the input PIN block format is ANSI X9.8, ISO format 0, VISA format 1, VISA format 4, ECI format 1, ISO format 1, ISO format 2, VISA format 2, IBM Encrypting PINPAD format, or ECI format 3, the service extracts the PIN according to the length specified in the PIN block.

IBM Encrypting PINPAD format, or ECI format 3, the service extracts the PIN according to the length specified in the PIN block.
Encrypting PINPAD format, or ECI format 3, and if the specified PIN length is less than 4, the service sets a reason code to reject the operation. If the specified PIN length is greater than 12, the operation proceeds to normal completion with unpredictable contents in the output PIN block. Otherwise, the service extracts the PIN according to the specified length.

- If the input PIN block format is VISA format 3, the specified delimiter (padding) determines the PIN length. The search starts at the leftmost digit in the PIN block. If the input PIN block format is 3624, the specification of a PIN extraction method for the 3624 is supported through rule array keywords. If no PIN extraction method is specified in the rule array, the specified delimiter (padding) determines the PIN length.

- If the input PIN block format is 3621, the specification of a PIN extraction method for the 3621 is supported through rule array keywords. If no PIN extraction method is specified in the rule array, the specified delimiter (padding) determines the PIN length.

- If the input block format is ECI format 2, the PIN is always the leftmost 4 digits.

If the maximum PIN length allowed by the output PIN block is shorter than the extracted PIN, only the leftmost digits of the extracted PIN that form the allowable maximum length are placed in the output PIN block. The PIN length field in the output PIN block, if it exists, specifies the allowable maximum length.

**PIN Change/Unblock (CSNBPCU) Callable Service**

The PIN Block calculation PIN Change/Unblock (CSNBPCU):

1. Form three 8-byte, 16-digit blocks, -1, -2, and -3, and set all digits to X’0’
2. Replace the rightmost four bytes of block-1 with the authentication code described in the previous section.
3. Set the second digit of block-2 to the length of the new PIN (4 to 12), followed by the new PIN, and padded to the right with X’F’
4. Include any current PIN by placing it into the leftmost digits of block-3.
5. Exclusive-OR blocks -1, -2, and -3 to form the 8-byte PIN block.
6. Pad the PIN block with other portions of the message for the smart card:
   - Prepend X’08’
   - Append X’80’
   - Append an additional six bytes of X’00’

The resulting message is ECB-mode triple-encrypted with an appropriate session key.

**IBM PIN Algorithms**

This section describes the IBM PIN generation algorithms, IBM PIN offset generation algorithm, and IBM PIN verification algorithms.

**3624 PIN Generation Algorithm**

This algorithm generates a n-digit PIN based on an account-related data or person-related data, namely the validation data. The assigned PIN length parameter specifies the length of the generated PIN.

The algorithm requires the following input parameters:

- A 64-bit validation data
- A 64-bit decimalization table
- A 4-bit assigned PIN length
A 128-bit PIN-generation key

The service uses the PIN generation key to encipher the validation data. Each digit of the enciphered validation data is replaced by the digit in the decimalization table whose displacement from the leftmost digit of the table is the same as the value of the digit of the enciphered validation data. The result is an intermediate PIN. The leftmost n digits of the intermediate PIN are the generated PIN, where n is specified by the assigned PIN length.

Figure 15 illustrates the 3624 PIN generation algorithm.

German Banking Pool PIN Generation Algorithm

This algorithm generates a 4-digit PIN based on an account-related data or person-related data, namely the validation data.

The algorithm requires the following input parameters:
- A 64-bit validation data
- A 64-bit decimalization table
- A 128-bit PIN-generation key

The validation data is enciphered using the PIN generation key. Each digit of the enciphered validation data is replaced by the digit in the decimalization table whose displacement from the leftmost digit of the table is the same as the value of the digit of the enciphered validation data. The result is an intermediate PIN. The rightmost 4 digits of the leftmost 6 digits of the intermediate PIN are extracted. The leftmost digit of the extracted 4 digits is checked for zero. If the digit is zero, the digit is changed to one; otherwise, the digit remains unchanged. The resulting four digits is the generated PIN.
PIN Offset Generation Algorithm

To allow the customer to select his own PIN, a PIN offset is used by the IBM 3624 and GBP PIN generation algorithms to relate the customer-selected PIN to the generated PIN.

The PIN offset generation algorithm requires two parameters in addition to those used in the 3624 PIN generation algorithm. They are a customer-selected PIN and a 4-bit PIN check length. The length of the customer-selected PIN is equal to the assigned-PIN length, n.

The 3624 PIN generation algorithm described in the previous section is performed. The offset data value is the result of subtracting (modulo 10) the leftmost n digits of the intermediate PIN from the customer-selected PIN. The modulo 10 subtraction ignores carries. The rightmost m digits of the offset data form the PIN offset, where m is specified by the PIN check length. Note that n cannot be less than m.

To generate a PIN offset for a GBP PIN, m is set to 4 and n is set to 6.

Figure 16 illustrates the German Banking Pool (GBP) PIN generation algorithm.

Figure 16. GBP PIN Generation Algorithm

If A = 0, then Z = 1; otherwise, Z = A.

Figure 16. GBP PIN Generation Algorithm

PIN Offset Generation Algorithm

To allow the customer to select his own PIN, a PIN offset is used by the IBM 3624 and GBP PIN generation algorithms to relate the customer-selected PIN to the generated PIN.

The PIN offset generation algorithm requires two parameters in addition to those used in the 3624 PIN generation algorithm. They are a customer-selected PIN and a 4-bit PIN check length. The length of the customer-selected PIN is equal to the assigned-PIN length, n.

The 3624 PIN generation algorithm described in the previous section is performed. The offset data value is the result of subtracting (modulo 10) the leftmost n digits of the intermediate PIN from the customer-selected PIN. The modulo 10 subtraction ignores carries. The rightmost m digits of the offset data form the PIN offset, where m is specified by the PIN check length. Note that n cannot be less than m.

To generate a PIN offset for a GBP PIN, m is set to 4 and n is set to 6.

Figure 17 illustrates the PIN offset generation algorithm.

Figure 17. PIN Offset Generation Algorithm
3624 PIN Verification Algorithm

This algorithm generates an intermediate PIN based on the specified validation data. A part of the intermediate PIN is adjusted by adding an offset data. A part of the result is compared with the corresponding part of the customer-entered PIN.

The algorithm requires the following input parameters:
- A 64-bit validation data
- A 64-bit decimalization table
- A 128-bit PIN-verification key
- A 4-bit PIN check length
An offset data
A customer-entered PIN

The rightmost m digits of the offset data form the PIN offset, where m is the PIN check length.
1. The validation data is enciphered using the PIN verification key. Each digit of the enciphered validation data is replaced by the digit in the decimalization table whose displacement from the leftmost digit of the table is the same as the value of the digit of enciphered validation data.
2. The leftmost n digits of the result is added (modulo 10) to the offset data value, where n is the length of the customer-entered PIN. The modulo 10 addition ignores carries.
3. The rightmost m digits of the result of the addition operation form the PIN check number. The PIN check number is compared with the rightmost m digits of the customer-entered PIN. If they match, PIN verification is successful; otherwise, verification is unsuccessful.

When a nonzero PIN offset is used, the length of the customer-entered PIN is equal to the assigned PIN length.

Figure 18 illustrates the PIN verification algorithm.
German Banking Pool PIN Verification Algorithm

This algorithm generates an intermediate PIN based on the specified validation data. A part of the intermediate PIN is adjusted by adding an offset data. A part of the result is extracted. The extracted value may or may not be modified before it compares with the customer-entered PIN.

The algorithm requires the following input parameters:

- Length of CE PIN
- PIN Check Length
- PIN CN: PIN Check Number
- CE PIN: Customer-entered PIN

Figure 18. PIN Verification Algorithm

Appendix F. Cryptographic Algorithms and Processes   677
The rightmost 4 digits of the offset data form the PIN offset.

1. The validation data is enciphered using the PIN verification key. Each digit of the enciphered validation data is replaced by the digit in the decimalization table whose displacement from the leftmost digit of the table is the same as the value of the digit of enciphered validation data.

2. The leftmost 6 digits of the result is added (modulo 10) to the offset data. The modulo 10 addition ignores carries.

3. The rightmost 4 digits of the result of the addition (modulo 10) are extracted.

4. The leftmost digit of the extracted value is checked for zero. If the digit is zero, the digit is set to one; otherwise, the digit remains unchanged. The resulting four digits are compared with the customer-entered PIN. If they match, PIN verification is successful; otherwise, verification is unsuccessful.

Figure 19 illustrates the GBP PIN verification algorithm.

**VISA PIN Algorithms**

The VISA PIN verification algorithm performs a multiple encipherment of a value, called the transformed security parameter (TSP), and a extraction of a 4-digit PIN verification value (PVV) from the ciphertext. The calculated PVV is compared with the referenced PVV and stored on the plastic card or data base. If they match, verification is successful.

**PVV Generation Algorithm**

The algorithm generates a 4-digit PIN verification value (PVV) based on the transformed security parameter (TSP).

The algorithm requires the following input parameters:

- A 64-bit TSP
- A 128-bit PVV generation key
1. A multiple encipherment of the TSP using the double-length PVV generation key is performed.

2. The ciphertext is scanned from left to right. Decimal digits are selected during the scan until four decimal digits are found. Each selected digit is placed from left to right according to the order of selection. If four decimal digits are found, those digits are the PVV.

3. If, at the end of the first scan, less than four decimal digits have been selected, a second scan is performed from left to right. During the second scan, all decimal digits are skipped and only nondecimal digits can be processed. Nondecimal digits are converted to decimal digits by subtracting 10. The process proceeds until four digits of PVV are found.

Figure 20 illustrates the PVV generation algorithm.

Programming Note: For VISA PVV algorithms, the leftmost 11 digits of the TSP are the personal account number (PAN), the leftmost 12th digit is a key table index to select the PVV generation key, and the rightmost 4 digits are the PIN. The key table index should have a value between 1 and 6, inclusive.

PVV Verification Algorithm
The algorithm requires the following input parameters:

- A 64-bit TSP
A 16-bit referenced PVV
A 128-bit PVV verification key

A PVV is generated using the PVV generation algorithm, except a PVV verification key rather than a PVV generation key is used. The generated PVV is compared with the referenced PVV. If they match, verification is successful.

**Interbank PIN Generation Algorithm**

The Interbank PIN calculation method consists of the following steps:

1. Let X denote the transaction_security parameter element converted to an array of 16 4-bit numeric values. This parameter consists of (in the following sequence) the 11 rightmost digits of the customer PAN (excluding the check digit), a constant of 6, a 1-digit key indicator, and a 3-digit validation field.
2. Encrypt X with the double-length PINGEN (or PINVER) key to get 16 hexadecimal digits (64 bits).
3. Perform decimalization on the result of the previous step by scanning the 16 hexadecimal digits from left to right, skipping any digit greater than X'9' until 4 decimal digits (for example, digits that have values from X'0' to X'9') are found. If all digits are scanned but 4 decimal digits are not found, repeat the scanning process, skipping all digits that are X'9' or less and selecting the digits that are greater than X'9'. Subtract 10 (X'A') from each digit selected in this scan. If the 4 digits that were found are all zeros, replace the 4 digits with 0100.
4. Concatenate and use the resulting digits for the Interbank PIN. The 4-digit PIN consists of the decimal digits in the sequence in which they are found.

**Cipher Processing Rules**

DES defines operations on 8-byte data strings. Although the fundamental concepts of ciphering (enciphering and deciphering) and data verification are simple, there are different approaches to processing data strings that are not a multiple of 8 bytes in length. These approaches are defined in various standards and IBM products.

**CBC and ANSI X3.106**

ANSI standard X3.106 defines four methods of operation for ciphering. One of these modes, cipher block chaining (CBC), defines the basic method for performing ciphering on multiple 8-byte data strings. A plaintext data string, which must be a multiple of 8 bytes, is processed as a series of 8-byte groups. The ciphered result from processing an 8-byte group is exclusive ORed with the next group of 8 input bytes. The last 8-byte ciphered result is defined as an output chaining vector (OCV). ICSF stores the output chaining vector value in the chaining_vector parameter.

An initial chaining vector is exclusive ORed with the first group of 8 input bytes.

In summary:
- An input chaining vector (ICV) is required.
- If the text_length is not an exact multiple of 8 bytes, the request fails.
- The plaintext is not padded, for example, the output text length is not increased.

**ANSI X9.23 and IBM 4700**

An enhancement to the basic cipher block chaining mode of ANSI X3.106 is defined so the data lengths that are not an exact multiple of 8 bytes can be processed. The
ANSI X9.23 method always adds from 1 byte to 8 bytes to the plaintext before encipherment. The last added byte is the count of the added bytes and is in the range of X'01' to X'08'. The standard defines that the other added bytes, the pad characters, are random.

When ICSF enciphers the plaintext, the resulting ciphertext is always 1 to 8 bytes longer than the plaintext.

When ICSF deciphers the ciphertext, ICSF uses the last byte of the deciphered data as the number of bytes to be removed (the pad bytes and the count byte). The resulting plaintext is the same as the original plaintext.

The output chaining vector can be used as feedback with this method in the same way as with the X3.106 method.

In summary, for the ANSI X9.23 method:
- X9.23 processing requires the caller to supply an ICV.
- X9.23 encipher does not allow specification of a pad character.

The 4700 padding rule is similar to the X9.23 rule. The only difference is that in the X9.23 method, the padding character is not user-selected, but the padding string is selected by the encipher process.

**Segmenting**

The callable services can operate on large data objects. Segmenting is the process of dividing the function into more than one processing step. Your application can divide the process into multiple steps without changing the final outcome.

To provide segmenting capability, the MAC generation, MAC verification, and MDC generation callable services require an 18-byte system work area in the application address space that is provided as the chaining vector parameter to the callable service. The application program must not change the system work area.

**Cipher Last-Block Rules**

The DES defines cipher-block chaining as operating on multiples of 8 bytes. Various algorithms are used to process strings that are multiples of 8 bytes. The algorithms are generically named “last-block rules”. You select the supported last-block rules by using these keywords:
- X9.23
- IPS
- CUSP (also used with PCF)
- 4700-PAD

You specify which cipher last-block rule you want to use in the rule_array parameter of the callable service.

**CUSP**

If the length of the data to be enciphered is an exact multiple of 8 bytes, the ICV is exclusive ORed with the first 8-byte block of plaintext, and the resulting 8 bytes are passed to the DES with the specified key. The resulting 8-byte block of ciphertext is then exclusive ORed with the second 8-byte block of plaintext, and the value is enciphered. This process continues until the last 8-byte block of plaintext is to be enciphered. Because the length of this last block is exactly 8 bytes, the last block is processed in an identical manner to all the preceding blocks.
To produce the OCV, the last block of ciphertext is enciphered again (thus producing a double-enciphered block). The user can pass this value of the OCV as the ICV in his next encipher call to produce chaining between successive calls. The caller can alternatively pass the same ICV on every call to the callable service.

If the length of data to be enciphered is greater than 7 bytes, and is not an exact multiple of 8 bytes, the process is the same as that above, until the last partial block of 1 to 7 bytes is reached. To encipher the last short block, the previous 8-byte block of ciphertext is passed to the DES with the specified key. The first 1 to 7 bytes of this double-enciphered block has two uses. The first use is to exclusive OR this block with the last short block of plaintext to form the last short block of the ciphertext. The second use is to pass it back as the OCV. Thus, the OCV is the last complete 8-byte block of plaintext, doubly enciphered.

If the length of the data to be enciphered is less than 8 bytes, the ICV is enciphered under the specified key. The first 1 to 7 bytes of the enciphered ICV is exclusive ORed with the plaintext to form the ciphertext. The OCV is the enciphered ICV.

**The Information Protection System (IPS)**

The Information Protection System (IPS) offers two forms of chaining: block and record. Under record chaining, the OCV for each enciphered data string becomes the ICV for the next. Under block chaining, the same ICV is used for each encipherment.

Files that are enciphered directly with the ICSF encipher callable service cannot be properly deciphered using the IPS/CMS CIPHER command or the IPS/CMS subroutines. Both IPS/CMS CIPHER and AMS REPRO ENCIPHER write headers to their files that contain information (principally the ICV and chaining method) needed for decipherment. The encipher callable service does not generate these headers. Specialized techniques are described in IPS/CMS documentation to overcome some, if not all, of these limitations, depending on the chaining mode. As a rough test, you can attempt a decipherment with the CIPHER command HDWARN option, which causes CIPHER to continue processing even though the header is absent.

The encipher callable service returns an OCV used by IPS for record chaining. This allows cryptographic applications using ICSF to be compatible with IPS record chaining.

Record chaining provides a superior method of handling successive short blocks, and has better error recovery features when the caller passes successive short blocks.

The principle used by record chaining is that the OCV is the last 8 bytes of ciphertext. This is handled as follows:

- If the length of the data to be enciphered is an exact multiple of 8 bytes, the ICV is exclusive ORed with the first 8-byte block of plaintext, and the resulting 8 bytes are passed to DES with the specified key. The resulting 8-byte block of ciphertext is then exclusive ORed with the second 8-byte block of plaintext, and the resulting value is enciphered. This process continues until the last 8-byte block of plaintext is to be enciphered. Because the length of this last block is exactly 8 bytes, the last block is processed in an identical manner to all the preceding blocks. The OCV is the last 8 bytes of ciphertext.

The user can pass this value as the ICV in the next encipher call to produce chaining between successive calls.
• If the length of data to be enciphered is greater than 7 bytes, and is not an exact multiple of 8 bytes, the process is the same as that above, until the last partial block of 1 to 7 bytes is reached. To encipher the last short block, the previous 8-byte block of ciphertext is passed to the DES with the specified key. The first 1 to 7 bytes of this doubly enciphered block is then exclusive ORed with the last short block of plaintext to form the last short block of the ciphertext. The OCV is the last 8 bytes of ciphertext.

• If the length of the data to be enciphered is less than 8 bytes, then the ICV is enciphered under the specified key. The first 1 to 7 bytes of the enciphered ICV is exclusive ORed with the plaintext to form the ciphertext. The OCV is the rightmost 8 bytes of the plaintext ICV concatenated with the short block of ciphertext. For example:

\[
\begin{align*}
\text{ICV} & = \text{ABCDEFGH} \\
\text{ciphertext} & = \text{XYZ} \\
\text{OCV} & = \text{DEFGXYZ}
\end{align*}
\]

PKCS Padding Method

This section describes the algorithm used to pad clear text when the PKCS-PAD method is specified. Padding is applied before encryption when this keyword is specified with the Symmetric Algorithm Encipher callable service, and it is removed from decrypted data when the keyword is specified with the Symmetric Algorithm Decipher callable service.

The rules for PKCS padding are very simple:
• Padding bytes are always added to the clear text before it is encrypted.
• Each padding byte has a value equal to the total number of padding bytes that are added. For example, if 6 padding bytes must be added, each of those bytes will have the value 0x06.
• The total number of padding bytes is at least one, and is the number that is required in order to bring the data length up to a multiple of the cipher algorithm block size.

The callable services described in this document use AES, which has a cipher block size of 16 bytes. The total number of padding bytes added to the clear text will always be between 1 and 16. The table below indicates exactly how many padding bytes are added according to the data length, and also shows the value of the padding bytes that are applied.

<table>
<thead>
<tr>
<th>Value of clear text length (mod 16)</th>
<th>Number of padding bytes added</th>
<th>Value of each padding byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16</td>
<td>0x10</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>0x0F</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>0x0E</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>0x0D</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>0x0C</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>0x0B</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>0x0A</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>0x09</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>0x08</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>0x07</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>0x06</td>
</tr>
</tbody>
</table>
PKCS Padding Method

<table>
<thead>
<tr>
<th>Value of clear text length (mod 16)</th>
<th>Number of padding bytes added</th>
<th>Value of each padding byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>5</td>
<td>0x05</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>0x04</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>0x03</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>0x02</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>0x01</td>
</tr>
</tbody>
</table>

Note that the PKCS standards that define this padding method describe it in a way that limits the maximum padding length to 8 bytes. This is a consequence of the fact that the algorithms at that time used 8-byte blocks. We extend the definition to apply to 16-byte AES cipher blocks.

EXAMPLE 1

Clear text consists of the following 18 bytes:
F14ADBDA019D6DB7 EFD91546E3FF8444 9BCB

In order to make this a multiple of 16 bytes (the AES block size), we must add 14 bytes. Each byte will contain the value 0x0E, which is 14, the total number of padding bytes added. The result is that the padded clear text is as follows:
F14ADBDA019D6DB7 EFD91546E3FF8444 9BCB0E0E0E0E0E0E0E
0E0E0E0E0E0E0E0E

The padded value is 32 bytes in length, which is two AES blocks. This padded string is encrypted in CBC mode, and the resulting ciphertext will also be 32 bytes in length.

EXAMPLE 1

Clear text consists of the following 16 bytes:
971ACD01C9C7ADEA CC83257926F490FF

This is already a multiple of the AES block size, but PKCS padding rules say that padding is always applied. Thus, we add 16 bytes of padding to bring the total length to 32, the next multiple of the AES block size. Each pad byte has the value 0x10, which is 16, the total number of padding bytes added. The result is that the padded clear text is as follows:
971ACD01C9C7ADEA CC83257926F490FF 1010101010101010
1010101010101010

The padded value is 32 bytes in length, which is two AES blocks. This padded string is encrypted in CBC mode, and the resulting cipher text will also be 32 bytes in length.

Multiple Decipherment and Encipherment

This appendix explains multiple encipherment and decipherment and their equations.

The Cryptographic Coprocessor Feature uses multiple encipherment whenever it enciphers a key under a key-encrypting key like the master key or the transport key and in triple-DES encipherment for data privacy. Multiple encipherment is superior
to single encipherment because multiple encipherment increases the work needed to “break” a key. ICSF provides extra protection for a key by enciphering it under an enciphering key multiple times rather than once. The multiple encipherment method for keys enciphered under a key-encrypting key uses a double-length (128 bit) key split into two 64-bit halves. Like single encipherment, multiple encipherment uses a DES based on the electronic code book (ECB) mode of encipherment.

Keys can either be double-length or single-length depending on the installation and their cryptographic function. When a single-length key is encrypted under a double-length key, multiple encipherment is performed on the key. In the multiple encipherment method, the key is encrypted under the left half of the enciphering key. The result is then decrypted under the right half of the enciphering key. Finally, this result is encrypted under the left half of the enciphering key again.

When a double-length key is encrypted with multiple encipherment, the method is similar, except ICSF uses two enciphering keys. One enciphering key encrypts each half of the double-length key. Double-length keys active on the system have two master key variants used when enciphering them.

Multiple encipherment and decipherment is not only used to protect or retrieve a cryptographic key, but they are also used to protect or retrieve 64-bit data in the area of PIN applications. For example, the following two sections use a double-length *KEK as an example to cipher a single-length key even though the same algorithms apply to cipher 64-bit data by a double-length PIN-related cryptographic key.

ICSF also supports triple-DES encipherment for data privacy using double-length and triple-length DATA keys. For this procedure the data is first enciphered using the first DATA key. The result is then deciphered using the second DATA key. This second result is then enciphered using the third DATA key when a triple-length key is provided, or reusing the first DATA key when a double-length key is provided.

Note that an asterisk (*) preceding the key means that the key is double-length. Notations in this chapter have the following meaning:
- eK(x), where x is enciphered under K
- dK(y) represents plaintext, where K is the key and y is the ciphertext

Therefore, dK(eK(x)) equals x for any 64-bit key K and any 64-bit plaintext x.

When a key (*K) to be protected is double-length, two double-length *KEKs are used. One *KEK is used for protecting the left half of the key (*K); another is for the right half. Multiple encipherment is used with the appropriate *KEK for protecting each half of the key.

**Multiple Encipherment of Single-length Keys**

The multiple encipherment of a single-length key (K) using a double-length *KEK is defined as follows:

\[ e^{*KEK}(K) = e^{KEKL}(d^{KEKR}(e^{KEKL}(K))) \]

where KEKL is the left 64 bits of *KEK and KEKR is the right 64 bits of *KEK.

*Figure 21* illustrates the definition.
Multiple Decipherment of Single-length Keys

The multiple encipherment of an encrypted single-length key \( Y = e^{*\text{KEK}}(K) \) using a double-length \( *\text{KEK} \) is defined as follows:

\[
\begin{align*}
  d^{*\text{KEK}}(Y) &= d\text{KEKL}(e\text{KEKR}(d\text{KEKL}(Y))) \\
  &= d^{*\text{KEK}}(e^{*\text{KEK}}(K)) \\
  &= K
\end{align*}
\]

where KEKL is the left 64 bits of \( *\text{KEK} \) and KEKR is the right 64 bits of \( *\text{KEK} \).

Figure 22 illustrates the definition.

---

Figure 21. Multiple Encipherment of Single-length Keys
Multiple Encipherment of Double-length Keys

The multiple encipherment of a double-length key (*K) using two double-length *KEKs, *KEKa and *KEKb is defined as follows:

\[
e^{*KEK}(K) = e^{*KEKa}(d^{KEKa}(e^{KEKa}(K_L))) || e^{KEKb}(d^{KEKb}(e^{KEKb}(K_R)))
\]

where:

- \( K_L \) is the left 64 bits of *K.
- \( K_R \) is the right 64 bits of *K.
- \( KEKaL \) is the left 64 bits of *KEKa.
- \( KEKaR \) is the right 64 bits of *KEKa.
- \( KEKbL \) is the left 64 bits of *KEKb.
- \( KEKbR \) is the right 64 bits of *KEKb.
- || means concatenation.

*Figure 23* illustrates the definition.
Multiple Decipherment of Double-length Keys

The multiple decipherment of an encrypted double-length key, *Y = e*KEKa(KL) || e*KEKb(KR), using two double-length *KEKs, *KEKa and *KEKb, is defined as follows:

\[
D^{*KEKa}(YL) || d^{*KEKb}(YR) = d^{KEKaL}(e^{KEKaR}(d^{KEKaL}(YL))) || d^{KEKbL}(e^{KEKbR}(d^{KEKbL}(YR)))
\]

\[
= d^{*KEKa}(e^{*KEKa}(KL)) || d^{*KEKb}(e^{*KEKb}(KR)) = *K
\]

where

- YL is the left 64 bits of *Y.
- YR is the right 64 bits of *Y.
- KEKaL is the left 64 bits of *KEKa.
- KEKaR is the right 64 bits of *KEKa.
- KEKbL is the left 64 bits of *KEKb.
- KEKbR is the right 64 bits of *KEKb.
- || means concatenation.

*Figure 24* illustrates the definition.
Multiple Encipherment of Triple-length Keys

The multiple encipherment of a triple-length key (**K) using two double-length KEKs, *KEKa and *KEKb is defined as follows:

\[ e^{*\text{KEKa}}(KL) \oplus e^{*\text{KEKb}}(KR) = e^{\text{KEKaL}}(d^{\text{KEKaR}}(e^{\text{KEKaL}}(KL))) \oplus e^{\text{KEKbL}}(d^{\text{KEKbR}}(e^{\text{KEKbL}}(KM))) \oplus e^{\text{KEKaL}}(d^{\text{KEKaR}}(e^{\text{KEKaL}}(KR))) \]

where:
- KL is the left 64 bits of **K
- KM is the next 64 bits of **K
- KR is the right 64 bits of **K
- KEKaL is the left 64 bits of *KEKa
- KEKaR is the right 64 bits of *KEKa
- KEKbL is the left 64 bits of *KEKb
- KEKbR is the right 64 bits of *KEKb
- \( \oplus \) means concatenation

Figure 25 on page 690 illustrates the definition.

**Figure 24. Multiple Decipherment of Double-length Keys**
Multiple Decipherment of Triple-length Keys

The multiple decipherment of an encrypted triple-length key $**Y = e^{*KEKa}(KL) \ || e^{*KEKb}(KM) \ || e^{*KEKa}(KR)$, using two double-length *KEKs, *KEKa and *KEKb, is defined as follows:

$$
\begin{align*}
\text{d*KEKa}(YL) & \ || \ d\text{KEKb}(YM) & \ || & \ d\text{KEKa}(YR) \\
\text{dKEKaL}(e\text{KEKaR}(\text{dKEKaL}(YL))) & \ || & \ d\text{KEKbL}(e\text{KEKbR}(\text{dKEKbL}(YM))) & \ || & \ d\text{KEKaL}(e\text{KEKaR}(\text{dKEKaL}(YR))) \\
\text{dKEKaL}(e\text{KEKaR}(e\text{KEKaL}(KL))) & \ || & \ d\text{KEKbL}(e\text{KEKbR}(e\text{KEKbL}(KM))) & \ || & \ d\text{KEKaL}(e\text{KEKaR}(e\text{KEKaL}(KR))) \\
\text{**K} & & & & \\
\end{align*}
$$

where:

- YL is the left 64 bits of **Y
- YM is the next 64 bits of **Y
- YR is the right 64 bits of **Y
- KEKaL is the left 64 bits of *KEKa
- KEKaR is the right 64 bits of *KEKa
- KEKbL is the left 64 bits of *KEKb
- KEKbR is the right 64 bits of *KEKb
- || means concatenation

*Figure 26 on page 691* illustrates the definition.
PKA92 Key Format and Encryption Process

The PKA Symmetric Key Generate and the PKA Symmetric Key Import callable services optionally support a **PKA92** method of encrypting a DES or CDMF key with an RSA public key. This format is adapted from the IBM Transaction Security System (TSS) 4753 and 4755 product's implementation of "PKA92". The callable services do not create or accept the complete PKA92 AS key token as defined for the TSS products. Rather, the callable services only support the actual RSA-encrypted portion of a TSS PKA92 key token, the **AS External Key Block**.

**Forming an AS External Key Block** - The PKA96 implementation forms an AS External Key Block by RSA-encrypting a key block using a public key. The key block is formed by padding the key record detailed in Table 274 with zero bits on the left, high-order end of the key record. The process completes the key block with three sub-processes: masking, overwriting, and RSA encrypting.

**Table 274. PKA96 Clear DES Key Record**

<table>
<thead>
<tr>
<th>Offset (Bytes)</th>
<th>Length (Bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>005</td>
<td>Header and flags: X'01 0000 0000'</td>
</tr>
<tr>
<td>005</td>
<td>016</td>
<td>Environment Identifier (EID), encoded in ASCII</td>
</tr>
<tr>
<td>021</td>
<td>008</td>
<td>Control vector base for the DES key</td>
</tr>
</tbody>
</table>

Zero-bit padding to form a structure as long as the length of the public key modulus. The implementation constrains the public key modulus to a multiple of 64 bits in the range of 512 to 1024 bits. Note that government export or import regulations can impose limits on the modulus length. The maximum length is validated by a check against a value in the Function Control Vector.
Table 274. PKA96 Clear DES Key Record (continued)

<table>
<thead>
<tr>
<th>Offset (Bytes)</th>
<th>Length (Bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>029</td>
<td>008</td>
<td>Repeat of the CV data at offset 021</td>
</tr>
<tr>
<td>037</td>
<td>008</td>
<td>The single-length DES key or the left half of a double-length DES key</td>
</tr>
</tbody>
</table>
| 045            | 008            | The right half of a double-length DES key or a random number. This value is locally designated "K."
| 053            | 008            | Random number, "IV" |
| 061            | 001            | Ending byte, X'00' |

Masking Sub-process
1. Form the initial key block by padding the PKR with zero bits on the left, high-order end to the length of the modulus.
2. Create a mask by CBC encrypting a multiple of 8 bytes of binary zeros using K as the key and the length of the modulus, and IV as the initialization vector as defined in the key record at offsets 45 and 53. Exclusive-OR the mask with the key record and call the result PKR.
3. Exclusive-OR the mask with the key block.

Overwriting Sub-process
1. Set the high-order bits of PKR to B'01', and set the low-order bits to B'0110'.
2. Exclusive-OR K and IV and write the result at offset 45 in PKR.
3. Write IV at offset 53 in PKR. This causes the masked and overwritten PKR to have IV at its original position.

Encrypting Sub-process - RSA encrypt the overwritten PKR masked key record using the public key of the receiving node. This is the last step in creating an AS external key block.

Recovering a Key from an AS External Key Block - Recover the encrypted DES key from an AS External Key Block by performing decrypting, validating, unmasking, and extraction sub-processes.

Decrypting Sub-process - RSA decrypt the AS External Key Block using an RSA private key and call the result of the decryption PKR. The private key must be usable for key management purposes.

Validating Sub-process - Verify that the high-order two bits of the decrypted key block are valued to B'01' and that the low-order four bits of the PKR record are valued to B'0110'.

Unmasking Sub-process - Set IV to the value of the 8 bytes at offset 53 of the PKR record. Note that there is a variable quantity of padding prior to offset 0. See Table 274 on page 691

Set K to the exclusive-OR of IV and the value of the 8 bytes at offset 45 of the PKR record.

Create a mask that is equal in length to the key block by CBC encrypting a multiple of 8 bytes of binary zeros using K as the key and IV as the initialization vector. Exclusive-OR the mask with PKR and call the result the key record.
Copy K to offset 45 in the PKR record.

*Extraction Sub-process.* Confirm that:
- The four bytes at offset 1 in the PKR are valued to X'0000 0000'
- The two control vector fields at offsets 21 and 29 are identical
- If the control vector is an IMPORTER or EXPORTER key class, that the EID in the key record is not the same as the EID stored in the cryptographic engine.

The control vector base of the recovered key is the value at offset 21. If the control vector base bits 40 to 42 are valued to B'010' or B'110', the key is double length. Set the right half of the received key’s control vector equal to the left half and reverse bits 41 and 42 in the right half.

The recovered key is at offset 37 and is either 8 or 16 bytes long based on the control vector base bits 40 to 42. If these bits are valued to B'000', the key is single length. If these bits are valued to B'010' or B'110', the key is double length.

**ANSI X9.17 Partial Notarization Method**

The ANSI X9.17 notarization process can be divided into two procedures:

1. **Partial notarization,** in which the ANSI key-encrypting key (AKEK) is cryptographically combined with the origin and destination identifiers.

   **Note:** IBM defines this step as partial notarization. The ANSI X9.17 standard does not use the term partial notarization.

2. **Offsetting,** in which the result of the first step is exclusive-ORed with a counter value. ICSF performs the offset procedure to complete the notarization process when you use a partially notarized AKEK.

This appendix describes partial notarization for the ANSI X9.17 notarization process.

**Partial Notarization**

Partial notarization improves performance when you use an AKEK for many cryptographic service messages, each with a different counter value.

This section describes the steps in partial notarization. For more information about partial notarization, see [ANSI X9.17 Key Management Services](#) on page 46. For a description of the steps ICSF uses to complete the notarization of an AKEK or to notarize a key in one process, see ANSI X9.17 - 1985, Financial Institution Key Management (Wholesale).

**Notations Used in the Calculations**

- **KK** The 16-byte AKEK to be partially notarized
- **KKL** The leftmost 8 bytes of **KK**
- **KKR** The rightmost 8 bytes of **KK**
- **KK** The 8-byte AKEK to be partially notarized
- **KK1** An 8-byte intermediate result
- **KK2** An 8-byte intermediate result
- **FMID** The 16-byte origin identifier
- **FMID1** The leftmost 8 bytes of **FMID**
- **FMID2** The rightmost 8 bytes of **FMID**
**TOID**  The 16-byte destination identifier
**TOID1**  The leftmost 8 bytes of TOID
**TOID2**  The rightmost 8 bytes of TOID

**NSL**  An 8-byte intermediate result
**NSL1**  The leftmost 4 bytes of NSL

**NSR**  An 8-byte intermediate result
**NSR2**  The rightmost 4 bytes of NSR

***KKNI**  The 16-byte partially notarized AKEK
**KKNIL**  The leftmost 8 bytes of *KKNI
**KKNIR**  The rightmost 8 bytes of *KKNI
**KKNI**  The 8-byte partially notarized AKEK

**XOR**  Denotes the exclusive-OR operation
**TOID1<<1**  Denotes the ASCII TOID1 left-shifted one bit
**FMID1<<1**  Denotes the ASCII FMID1 left-shifted one bit
**eK(X)**  Denotes DES encryption of plaintext X using key K

 Denotes the concatenation operation

### Partial Notarization Calculation for a Double-Length AKEK
For a double-length AKEK, the partial notarization calculation consists of the following steps:
1. Set KK1 = KKL XOR TOID1<<1
2. Set KK2 = KKR XOR FMID1<<1
3. Set NSL = eKK2(TOID2)
4. Set NSR = eKK1(FMID2)
5. Set KKNI = KKL XOR NSL
6. Set KKNIR = KKR XOR NSR
7. Set *KKNI = KKNI || KKNIR

### Partial Notarization Calculation for a Single-Length AKEK
For a single-length AKEK, the partial notarization calculation consists of the following steps:
1. Set KK1 = KK XOR TOID1<<1
2. Set KK2 = KK XOR FMID1<<1
3. Set NSL = eKK2(TOID2)
4. Set NSR = eKK1(FMID2)
5. Set NSL = NSL1 || NSR2
6. Set KKNI = KK XOR NSL

### Transform CDMF Key Algorithm
The CDMF key transformation algorithm uses a 64-bit cryptographic key.
1. Set parity bits of the key to zero by ANDing the key with
   X'FEFEFEFEFEFEFEFE' to produce Kx.
2. Using DES, encipher Kx under the constant K1.
3. XOR this value with Kx to produce Ky.
4. AND Ky with X'0EFE0EFE0EFE0EFE' to produce Kz.
5. Using DES, encipher Kz under K2 to produce eK2(Kz).
6. Adjust $eK2(Kz)$ to odd parity in each byte. The result is the transformed key. The following figure illustrates these steps. ($e$ indicates DES encryption.)

![Diagram of the CDMF Key Transformation Algorithm]

**Figure 27. The CDMF Key Transformation Algorithm**

**Formatting Hashes and Keys in Public-Key Cryptography**

The digital signature generate and digital signature verify callable services support several methods for formatting a hash, and in some cases a descriptor for the hashing method, into a bit-string to be processed by the cryptographic algorithm. This topic discusses the ANSI X9.31 and PKCS #1 methods. The ISO 9796-1 method can be found in the ISO standard.

This topic also describes the PKCS #1, version 1, 1.5, and 2.0, methods for placing a key in a bit string for RSA ciphering in a key exchange.
ANSI X9.31 Hash Format

With ANSI X9.31, the string that is processed by the RSA algorithm is formatted by the concatenation of a header, padding, the hash and a trailer, from the most significant bit to the least significant bit, such that the resulting string is the same length as the modulus of the key. For the ICSF implementation, the modulus length must be a multiple of 8 bits.

- The header consists of X'6B'
- The padding consists of X'BB', repeated as many times as required, and terminated by X'BA'
- The hash value follows the padding
- The trailer consists of a hashing mechanism specifier and final byte. These specifiers are defined:
  - X'31': RIPEMD-160
  - X'33': SHA-1
- A final byte of X'CC'.

PKCS #1 Formats

Version 2.0 of the PKCS #1 standard defines methods for formatting keys and hashes prior to RSA encryption of the resulting data structures. The lower versions of the PKCS #1 standard defined block types 0, 1, and 2, but in the current standard that terminology is dropped.

ICSF implemented these processes using the terminology of the Version 2.0 standard:

- For formatting keys for secured transport (CSNDSYX, CSNDSYG, CSNDSYI):
  - RSAES-OAEP, the preferred method for key-encipherment when exchanging DATA keys between systems. Keyword PKCSOAEP is used to invoke this formatting technique. The parameter described in the standard is not used and its length is set to zero.
  - RSAES-PKCS1-v1_5, is an older method for formatting keys. Keyword PKCS-1.2 is used to invoke this formatting technique.
- For formatting hashes for digital signatures (CSNDSSG and CSNDSSV):
  - RSASSA-PKCS1-v1_5, the newer name for the block-type 1 format. Keyword PKCS-1.1 is used to invoke this formatting technique.
  - The PKCS #1 specification no longer discusses use of block-type 0. Keyword PKCS-1.0 is used to invoke this formatting technique. Use of block-type 0 is discouraged.

Using the terminology from older versions of the PKCS #1 standard, block types 0 and 1 are used to format a hash and block type 2 is used to format a DES key. The blocks consist of (|| means concatenation): X'00' || BT || PS || X'00' D where:

- BT is the block type, X'00', X'01', X'02'.
- PS is the padding of as many bytes as required to make the block the same length as the modulus of the RSA key, and is bytes of X'00' for block type 0, X'01' for block type 1, and random and non-X'00' for block type 2. The length of PS must be at least 8 bytes.

---

5. PKCS standards can be retrieved from http://www.rsasecurity.com/rsalabs/pkcs.
6. The PKA 92 method and the method incorporated into the SET standard are other examples of the Optimal Asymmetric Encryption Padding (OAEP) technique. The OAEP technique is attributed to Bellare and Rogaway.
D is the key, or the concatenation of the BER-encoded hash identifier and the hash.

You can create the BER encoding of an MD5 or SHA-1 value by prepending these strings to the 16 or 20-byte hash values, respectively:

MD5 \(\text{X}'3020300C~06082A86~4886F70D~02050500~0410'\)
SHA-1 \(\text{X}'30213009~06052B0E~03021A05~000414'\)

**Visa and EMV-related smart card formats and processes**

The VISA and EMV specifications for performing secure messaging with an EMV compliant smart card are covered in these documents:

- **EMV 2000 Integrated Circuit Card Specification for Payment Systems Version 4.0 (EMV4.0) Book 2**
- **Integrated Circuit Card Specification (VIS) 1.4.0 Corrections**

Book 2, Annex A1.3, describes how a smart-card, card-specific authentication code is derived from a card-issuer-supplied encryption key (ENC-MDK). The *Integrated Circuit Card Specification (VIS) 1.4.0 Corrections* indicates that the key used should be an authentication key (MAC-MDK).

Book 2, Annex A1.3 describes how a smart-card, card-specific session key is derived from a card-issuer-supplied PIN-block-encryption key (ENC-MDK). The encryption key is derived using a "tree-based-derivation" technique. IBM CCA offers two variations of the tree-based technique (TDESEMV2 and TDESEMV4), and a third technique CCA designates TDES-XOR.

In addition, Book 2 describes construction of the PIN block sent to an EMV card to initialize or update the user’s PIN.


Augmented by the above-mentioned documentation, the relevant processes are described in these sections:

- "Deriving the smart-card-specific authentication code"
- "Constructing the PIN-block for transporting an EMV smart-card PIN"
- "Deriving the CCA TDES-XOR session key" on page 698
- "Deriving the EMV TDESEMVn tree-based session key" on page 698
- "PIN-block self-encryption" on page 699

**Deriving the smart-card-specific authentication code**

To ensure that an original or replacement PIN is received from an authorized source, the EMV PIN-transport PIN-block incorporates an authentication code. The authentication code is the rightmost four bytes resulting from the ECB-mode triple-DES encryption of (the first) eight bytes of card-specific data (that is, the rightmost four bytes of the Unique DEA Key A).

**Constructing the PIN-block for transporting an EMV smart-card PIN**

The PIN block is used to transport a new PIN value. The PIN block also contains an authentication code, and optionally the "current" PIN value, enabling the smart
card to further ensure receipt of a valid PIN value. To enable incorporation of the
PIN block into the a message for an EMV smart-card, the PIN block is padded to
16 bytes prior to encryption.

PINs of length 4 - 12 digits are supported.

PIN-block construction:
1. Form three 8-byte, 16-digit blocks, block-1, block-2, and block-3, and set all
digits to X'0'.
2. Replace the rightmost four bytes of block-1 with the authentication code
described in the previous section.
3. Set the second digit of block-2 to the length of the new PIN (4 to 12), followed
by the new PIN, and padded to the right with X'F'.
4. Include any current PIN by placing it into the leftmost digits of block-3.
5. Exclusive-OR block-1, block-2, and block-3 to form the 8-byte PIN block.
6. Pad the PIN block with other portions of the message for the smart card:
   • Prepend X'08' (the length of the PIN block)
   • Append X'80', followed by 6 bytes of X'00'

The resulting message is ECB-mode triple-encrypted with an appropriate session
key.

Deriving the CCA TDES-XOR session key
In the diversified key generate and PIN change/unblock services, the TDES-XOR
process first derives a smart-card-specific intermediate key from the issuer-supplied
ENC-MDK key and card-specific data. (This intermediate key is also used in the
TDESEMV2 and TDESEMV4 processes. See the next section.) The intermediate
key is then modified using the application transaction counter (ATC) value supplied
by the smart card.

The double-length session-key creation steps:
1. Obtain the left-half of an intermediate key by ECB-mode triple-DES encrypting
   the (first) eight bytes of card specific data using the issuer-supplied ENC-MDK
   key.
2. Again using the ENC-MDK key, obtain the right-half of the intermediate key by
   ECB-mode triple-DES encrypting:
   • The second 8 bytes of card-specific derivation data when 16 bytes have been
     supplied
   • The exclusive-OR of the supplied 8 bytes of derivation data with X'FFFFFFFF
     FFFFFFFF'
3. Pad the ATC value to the left with six bytes of X'00' and exclusive-OR the result
   with the left-half of the intermediate key to obtain the left-half of the session key.
4. Obtain the one’s complement of the ATC by exclusive-ORing the ATC with
   X'FFFF'. Pad the result on the left with six bytes of X'00'. Exclusive-OR the
   8-byte result with the right-half of the intermediate key to obtain the right-half of
   the session key.

Deriving the EMV TDESEMVn tree-based session key
In the diversified key generate and PIN change/unblock services, the TDESEMV2
and TDESEMV4 keywords call for the creation of the session key with this process:
1. The intermediate key is obtained as explained above for the TDES-XOR process.

2. Combine the intermediate key with the two-byte Application Transaction Counter (ATC) and an optional Initial Value. The process is defined in the EMV 2000 Integrated Circuit Card Specification for Payment Systems Version 4.0 (EMV4.0) Book 2 Book 2, Annex A1.3.
   - TDESEMV2 causes processing with a branch factor of 2 and a height of 16.
   - TDESEMV4 causes processing with a branch factor of 4 and a height of 8.

PIN-block self-encryption

In the secure messaging for PINs (CSNBSPN) service, you can use the SELFENC rule-array keyword to specify that the 8-byte PIN block shall be used as a DES key to encrypt the PIN block. The verb appends the self-encrypted PIN block to the clear PIN-block in the output message.

Key Test Verification Pattern Algorithm for DES Keys

For DES keys, the key test callable service uses this algorithm to generate and verify the verification pattern.

\[
KK = \text{e}C(\ KL \ ) \ XOR \ KL \\
VP = \text{e}KK(\ KR \ XOR \ RN \ ) \ XOR \ RN
\]

where:

\(\text{e}K(x)\) - \(x\) is encrypted by key \(K\) using the DES algorithm

- \(KL\) is the left 128-bit clear key value of the key
- \(KR\) is the right 128-bit clear key value of the key (will be hex zero for a single length key)
- \(C\) is \(X'4545454545454545'\)
- \(KK\) is a 128-bit intermediate value
- \(RN\) is a 128-bit pseudo-random number
- \(VP\) is the 128-bit verification pattern
Appendix G. EBCDIC and ASCII Default Conversion Tables

This section presents tables showing EBCDIC to ASCII and ASCII to EBCDIC conversion tables. In the table headers, EBC refers to EBCDIC and ASC refers to ASCII.

Table 275 shows the EBCDIC to ASCII default conversion table.

Table 275. EBCDIC to ASCII Default Conversion Table

<table>
<thead>
<tr>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>20</td>
<td>81</td>
<td>40</td>
<td>2D</td>
<td>80</td>
<td>F8</td>
<td>A0</td>
<td>C8</td>
<td>C0</td>
<td>7B</td>
<td>E0</td>
<td>5C</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>21</td>
<td>82</td>
<td>41</td>
<td>A6</td>
<td>61</td>
<td>2F</td>
<td>81</td>
<td>61</td>
<td>A1</td>
<td>7E</td>
<td>C1</td>
<td>41</td>
</tr>
<tr>
<td>02</td>
<td>02</td>
<td>22</td>
<td>1C</td>
<td>42</td>
<td>E1</td>
<td>62</td>
<td>DF</td>
<td>82</td>
<td>62</td>
<td>A2</td>
<td>73</td>
<td>C2</td>
<td>42</td>
</tr>
<tr>
<td>03</td>
<td>CF</td>
<td>24</td>
<td>86</td>
<td>44</td>
<td>EB</td>
<td>64</td>
<td>9A</td>
<td>84</td>
<td>64</td>
<td>A4</td>
<td>75</td>
<td>C4</td>
<td>44</td>
</tr>
<tr>
<td>04</td>
<td>05</td>
<td>25</td>
<td>0A</td>
<td>45</td>
<td>90</td>
<td>65</td>
<td>DD</td>
<td>85</td>
<td>65</td>
<td>A5</td>
<td>76</td>
<td>C5</td>
<td>45</td>
</tr>
<tr>
<td>06</td>
<td>D3</td>
<td>26</td>
<td>17</td>
<td>46</td>
<td>9F</td>
<td>66</td>
<td>DE</td>
<td>86</td>
<td>66</td>
<td>A6</td>
<td>77</td>
<td>C6</td>
<td>46</td>
</tr>
<tr>
<td>07</td>
<td>7F</td>
<td>27</td>
<td>1B</td>
<td>47</td>
<td>E2</td>
<td>67</td>
<td>98</td>
<td>87</td>
<td>67</td>
<td>A7</td>
<td>78</td>
<td>C7</td>
<td>47</td>
</tr>
<tr>
<td>08</td>
<td>D4</td>
<td>28</td>
<td>89</td>
<td>48</td>
<td>AB</td>
<td>68</td>
<td>9D</td>
<td>88</td>
<td>68</td>
<td>A8</td>
<td>79</td>
<td>C8</td>
<td>48</td>
</tr>
<tr>
<td>09</td>
<td>D5</td>
<td>29</td>
<td>91</td>
<td>49</td>
<td>8B</td>
<td>69</td>
<td>AC</td>
<td>89</td>
<td>69</td>
<td>A9</td>
<td>7A</td>
<td>C9</td>
<td>49</td>
</tr>
<tr>
<td>0A</td>
<td>C3</td>
<td>2A</td>
<td>92</td>
<td>4A</td>
<td>9B</td>
<td>6A</td>
<td>BA</td>
<td>8A</td>
<td>9E</td>
<td>C0</td>
<td>CB</td>
<td>EA</td>
<td>A0</td>
</tr>
<tr>
<td>0B</td>
<td>0B</td>
<td>2B</td>
<td>95</td>
<td>4B</td>
<td>2E</td>
<td>6B</td>
<td>8B</td>
<td>AB</td>
<td>C0</td>
<td>CB</td>
<td>CA</td>
<td>CB</td>
<td>EB</td>
</tr>
<tr>
<td>0C</td>
<td>0C</td>
<td>2C</td>
<td>A2</td>
<td>4C</td>
<td>3C</td>
<td>6B</td>
<td>2C</td>
<td>8B</td>
<td>A4</td>
<td>AB</td>
<td>C0</td>
<td>CB</td>
<td>CA</td>
</tr>
<tr>
<td>0D</td>
<td>0D</td>
<td>2D</td>
<td>05</td>
<td>4D</td>
<td>28</td>
<td>6D</td>
<td>5F</td>
<td>8D</td>
<td>AF</td>
<td>AD</td>
<td>5B</td>
<td>CD</td>
<td>E8</td>
</tr>
<tr>
<td>0E</td>
<td>0E</td>
<td>2E</td>
<td>06</td>
<td>4E</td>
<td>2B</td>
<td>6E</td>
<td>3E</td>
<td>8E</td>
<td>AE</td>
<td>AE</td>
<td>F2</td>
<td>CE</td>
<td>EE</td>
</tr>
<tr>
<td>0F</td>
<td>0F</td>
<td>2F</td>
<td>07</td>
<td>4F</td>
<td>7C</td>
<td>6F</td>
<td>3F</td>
<td>8F</td>
<td>C5</td>
<td>AF</td>
<td>F9</td>
<td>CF</td>
<td>ED</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>30</td>
<td>E0</td>
<td>50</td>
<td>26</td>
<td>70</td>
<td>D7</td>
<td>90</td>
<td>8C</td>
<td>B0</td>
<td>B5</td>
<td>D0</td>
<td>7D</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>31</td>
<td>EE</td>
<td>51</td>
<td>A9</td>
<td>71</td>
<td>88</td>
<td>91</td>
<td>6A</td>
<td>B1</td>
<td>B6</td>
<td>D1</td>
<td>4A</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>32</td>
<td>16</td>
<td>52</td>
<td>AA</td>
<td>72</td>
<td>94</td>
<td>92</td>
<td>6B</td>
<td>B2</td>
<td>FD</td>
<td>D2</td>
<td>4B</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>33</td>
<td>E5</td>
<td>53</td>
<td>9C</td>
<td>73</td>
<td>B0</td>
<td>93</td>
<td>6C</td>
<td>B3</td>
<td>B7</td>
<td>D3</td>
<td>4C</td>
</tr>
<tr>
<td>14</td>
<td>C7</td>
<td>34</td>
<td>D0</td>
<td>54</td>
<td>DB</td>
<td>74</td>
<td>B1</td>
<td>94</td>
<td>6D</td>
<td>B4</td>
<td>B8</td>
<td>D4</td>
<td>4D</td>
</tr>
<tr>
<td>15</td>
<td>B4</td>
<td>35</td>
<td>1E</td>
<td>55</td>
<td>A5</td>
<td>75</td>
<td>B2</td>
<td>95</td>
<td>6E</td>
<td>B5</td>
<td>B9</td>
<td>D5</td>
<td>4E</td>
</tr>
<tr>
<td>16</td>
<td>08</td>
<td>36</td>
<td>EA</td>
<td>56</td>
<td>99</td>
<td>76</td>
<td>FC</td>
<td>96</td>
<td>6F</td>
<td>B6</td>
<td>E6</td>
<td>D6</td>
<td>4F</td>
</tr>
<tr>
<td>17</td>
<td>C9</td>
<td>37</td>
<td>04</td>
<td>57</td>
<td>E3</td>
<td>77</td>
<td>D6</td>
<td>97</td>
<td>70</td>
<td>B7</td>
<td>BB</td>
<td>D7</td>
<td>50</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>38</td>
<td>8A</td>
<td>58</td>
<td>A8</td>
<td>78</td>
<td>FB</td>
<td>98</td>
<td>71</td>
<td>B8</td>
<td>BC</td>
<td>D8</td>
<td>51</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>39</td>
<td>F6</td>
<td>59</td>
<td>9E</td>
<td>79</td>
<td>60</td>
<td>99</td>
<td>72</td>
<td>B9</td>
<td>BD</td>
<td>D9</td>
<td>52</td>
</tr>
<tr>
<td>1A</td>
<td>CC</td>
<td>3A</td>
<td>C6</td>
<td>5A</td>
<td>21</td>
<td>7A</td>
<td>3A</td>
<td>9A</td>
<td>97</td>
<td>BA</td>
<td>8D</td>
<td>DA</td>
<td>A1</td>
</tr>
<tr>
<td>1B</td>
<td>CD</td>
<td>3B</td>
<td>C2</td>
<td>5B</td>
<td>24</td>
<td>7B</td>
<td>23</td>
<td>9B</td>
<td>87</td>
<td>BB</td>
<td>D9</td>
<td>DB</td>
<td>AD</td>
</tr>
<tr>
<td>1C</td>
<td>83</td>
<td>3C</td>
<td>14</td>
<td>5C</td>
<td>2A</td>
<td>7C</td>
<td>40</td>
<td>9C</td>
<td>CE</td>
<td>BC</td>
<td>BF</td>
<td>DC</td>
<td>F5</td>
</tr>
<tr>
<td>1D</td>
<td>1D</td>
<td>3D</td>
<td>15</td>
<td>5D</td>
<td>29</td>
<td>7D</td>
<td>27</td>
<td>9D</td>
<td>93</td>
<td>BD</td>
<td>5D</td>
<td>DD</td>
<td>F4</td>
</tr>
<tr>
<td>1E</td>
<td>D2</td>
<td>3E</td>
<td>C1</td>
<td>5E</td>
<td>3B</td>
<td>7E</td>
<td>3D</td>
<td>9E</td>
<td>F1</td>
<td>BE</td>
<td>D8</td>
<td>DE</td>
<td>A3</td>
</tr>
<tr>
<td>1F</td>
<td>1F</td>
<td>3F</td>
<td>1A</td>
<td>5F</td>
<td>5E</td>
<td>7F</td>
<td>22</td>
<td>9F</td>
<td>FE</td>
<td>BF</td>
<td>C4</td>
<td>DF</td>
<td>8F</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 1997, 2009
Table 276 shows the ASCII to EBCDIC default conversion table.

<table>
<thead>
<tr>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
<th>ASC</th>
<th>EBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>20</td>
<td>40</td>
<td>40</td>
<td>7C</td>
<td>60</td>
<td>79</td>
<td>80</td>
<td>43</td>
<td>A0</td>
<td>EA</td>
<td>C0</td>
<td>AB</td>
<td>E0</td>
<td>30</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>21</td>
<td>5A</td>
<td>41</td>
<td>C1</td>
<td>61</td>
<td>81</td>
<td>81</td>
<td>20</td>
<td>A1</td>
<td>DA</td>
<td>C1</td>
<td>3E</td>
<td>E1</td>
<td>42</td>
</tr>
<tr>
<td>02</td>
<td>02</td>
<td>22</td>
<td>7F</td>
<td>42</td>
<td>C2</td>
<td>62</td>
<td>82</td>
<td>82</td>
<td>21</td>
<td>A2</td>
<td>2C</td>
<td>C2</td>
<td>3B</td>
<td>E2</td>
<td>47</td>
</tr>
<tr>
<td>03</td>
<td>03</td>
<td>23</td>
<td>7B</td>
<td>43</td>
<td>C3</td>
<td>63</td>
<td>83</td>
<td>83</td>
<td>1C</td>
<td>A3</td>
<td>DE</td>
<td>C3</td>
<td>0A</td>
<td>E3</td>
<td>57</td>
</tr>
<tr>
<td>04</td>
<td>37</td>
<td>24</td>
<td>5B</td>
<td>44</td>
<td>C4</td>
<td>64</td>
<td>84</td>
<td>84</td>
<td>23</td>
<td>A4</td>
<td>8B</td>
<td>C4</td>
<td>BF</td>
<td>E4</td>
<td>EE</td>
</tr>
<tr>
<td>05</td>
<td>2D</td>
<td>25</td>
<td>6C</td>
<td>45</td>
<td>C5</td>
<td>65</td>
<td>85</td>
<td>85</td>
<td>EB</td>
<td>A5</td>
<td>55</td>
<td>C5</td>
<td>8F</td>
<td>E5</td>
<td>33</td>
</tr>
<tr>
<td>06</td>
<td>2E</td>
<td>26</td>
<td>50</td>
<td>46</td>
<td>C6</td>
<td>66</td>
<td>86</td>
<td>86</td>
<td>24</td>
<td>A6</td>
<td>41</td>
<td>C6</td>
<td>3A</td>
<td>E6</td>
<td>B6</td>
</tr>
<tr>
<td>07</td>
<td>2F</td>
<td>27</td>
<td>7D</td>
<td>47</td>
<td>C7</td>
<td>67</td>
<td>87</td>
<td>87</td>
<td>9B</td>
<td>A7</td>
<td>FE</td>
<td>C7</td>
<td>14</td>
<td>E7</td>
<td>E1</td>
</tr>
<tr>
<td>08</td>
<td>16</td>
<td>28</td>
<td>4D</td>
<td>48</td>
<td>C8</td>
<td>68</td>
<td>88</td>
<td>88</td>
<td>71</td>
<td>A8</td>
<td>58</td>
<td>C8</td>
<td>A0</td>
<td>E8</td>
<td>CD</td>
</tr>
<tr>
<td>09</td>
<td>05</td>
<td>29</td>
<td>5D</td>
<td>49</td>
<td>C9</td>
<td>69</td>
<td>89</td>
<td>89</td>
<td>28</td>
<td>A9</td>
<td>51</td>
<td>C9</td>
<td>17</td>
<td>E9</td>
<td>ED</td>
</tr>
<tr>
<td>0A</td>
<td>25</td>
<td>2A</td>
<td>5C</td>
<td>4A</td>
<td>D1</td>
<td>6A</td>
<td>91</td>
<td>8A</td>
<td>38</td>
<td>AA</td>
<td>52</td>
<td>CA</td>
<td>CB</td>
<td>EA</td>
<td>36</td>
</tr>
<tr>
<td>0B</td>
<td>0B</td>
<td>2B</td>
<td>4E</td>
<td>4B</td>
<td>D2</td>
<td>6B</td>
<td>92</td>
<td>8B</td>
<td>49</td>
<td>AB</td>
<td>48</td>
<td>CB</td>
<td>CA</td>
<td>EB</td>
<td>44</td>
</tr>
<tr>
<td>0C</td>
<td>0C</td>
<td>2C</td>
<td>6B</td>
<td>4C</td>
<td>D3</td>
<td>6C</td>
<td>93</td>
<td>8C</td>
<td>90</td>
<td>AC</td>
<td>69</td>
<td>CC</td>
<td>1A</td>
<td>EC</td>
<td>CE</td>
</tr>
<tr>
<td>0D</td>
<td>0D</td>
<td>2D</td>
<td>60</td>
<td>4D</td>
<td>D4</td>
<td>6D</td>
<td>94</td>
<td>8D</td>
<td>BA</td>
<td>AD</td>
<td>DB</td>
<td>CD</td>
<td>1B</td>
<td>ED</td>
<td>CF</td>
</tr>
<tr>
<td>0E</td>
<td>0E</td>
<td>2E</td>
<td>4B</td>
<td>4E</td>
<td>D5</td>
<td>6E</td>
<td>95</td>
<td>8E</td>
<td>EC</td>
<td>AE</td>
<td>8E</td>
<td>CE</td>
<td>9C</td>
<td>EE</td>
<td>31</td>
</tr>
<tr>
<td>0F</td>
<td>0F</td>
<td>2F</td>
<td>61</td>
<td>4F</td>
<td>D6</td>
<td>6F</td>
<td>96</td>
<td>8F</td>
<td>DF</td>
<td>AF</td>
<td>8D</td>
<td>CF</td>
<td>04</td>
<td>EF</td>
<td>AA</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>30</td>
<td>F0</td>
<td>50</td>
<td>D7</td>
<td>70</td>
<td>97</td>
<td>90</td>
<td>45</td>
<td>B0</td>
<td>73</td>
<td>D0</td>
<td>34</td>
<td>F0</td>
<td>FC</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>31</td>
<td>F1</td>
<td>51</td>
<td>D8</td>
<td>71</td>
<td>98</td>
<td>91</td>
<td>29</td>
<td>B1</td>
<td>74</td>
<td>D1</td>
<td>EF</td>
<td>F1</td>
<td>9E</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>32</td>
<td>F2</td>
<td>52</td>
<td>D9</td>
<td>72</td>
<td>99</td>
<td>92</td>
<td>2A</td>
<td>B2</td>
<td>75</td>
<td>D2</td>
<td>1E</td>
<td>F2</td>
<td>AE</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>33</td>
<td>F3</td>
<td>53</td>
<td>E2</td>
<td>73</td>
<td>A2</td>
<td>93</td>
<td>9D</td>
<td>B3</td>
<td>FA</td>
<td>D3</td>
<td>06</td>
<td>F3</td>
<td>8C</td>
</tr>
<tr>
<td>14</td>
<td>3C</td>
<td>34</td>
<td>F4</td>
<td>54</td>
<td>E3</td>
<td>74</td>
<td>A3</td>
<td>94</td>
<td>72</td>
<td>B4</td>
<td>15</td>
<td>D4</td>
<td>08</td>
<td>F4</td>
<td>DD</td>
</tr>
<tr>
<td>15</td>
<td>3D</td>
<td>35</td>
<td>F5</td>
<td>55</td>
<td>E4</td>
<td>75</td>
<td>A4</td>
<td>95</td>
<td>2B</td>
<td>B5</td>
<td>09</td>
<td>F5</td>
<td>DC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>32</td>
<td>36</td>
<td>F6</td>
<td>56</td>
<td>E5</td>
<td>76</td>
<td>A5</td>
<td>96</td>
<td>8A</td>
<td>B6</td>
<td>1D</td>
<td>D6</td>
<td>77</td>
<td>F6</td>
<td>39</td>
</tr>
<tr>
<td>17</td>
<td>26</td>
<td>37</td>
<td>F7</td>
<td>57</td>
<td>E6</td>
<td>77</td>
<td>A6</td>
<td>97</td>
<td>9A</td>
<td>B7</td>
<td>B3</td>
<td>D7</td>
<td>70</td>
<td>F7</td>
<td>FB</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>38</td>
<td>F8</td>
<td>58</td>
<td>E7</td>
<td>78</td>
<td>A7</td>
<td>98</td>
<td>67</td>
<td>B8</td>
<td>B4</td>
<td>D8</td>
<td>BE</td>
<td>F8</td>
<td>80</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>39</td>
<td>F9</td>
<td>59</td>
<td>E8</td>
<td>79</td>
<td>A8</td>
<td>99</td>
<td>56</td>
<td>B9</td>
<td>B5</td>
<td>D9</td>
<td>BB</td>
<td>F9</td>
<td>AF</td>
</tr>
<tr>
<td>1A</td>
<td>3F</td>
<td>3A</td>
<td>7A</td>
<td>5A</td>
<td>E9</td>
<td>7A</td>
<td>A9</td>
<td>9A</td>
<td>64</td>
<td>BA</td>
<td>6A</td>
<td>DA</td>
<td>AC</td>
<td>FA</td>
<td>FD</td>
</tr>
<tr>
<td>1B</td>
<td>27</td>
<td>3B</td>
<td>5E</td>
<td>5B</td>
<td>AD</td>
<td>7B</td>
<td>C0</td>
<td>9B</td>
<td>4A</td>
<td>BB</td>
<td>B7</td>
<td>DB</td>
<td>54</td>
<td>FB</td>
<td>78</td>
</tr>
<tr>
<td>1C</td>
<td>22</td>
<td>3C</td>
<td>4C</td>
<td>5C</td>
<td>E0</td>
<td>7C</td>
<td>4F</td>
<td>9C</td>
<td>53</td>
<td>BC</td>
<td>B8</td>
<td>DC</td>
<td>63</td>
<td>FC</td>
<td>76</td>
</tr>
<tr>
<td>1D</td>
<td>1D</td>
<td>3D</td>
<td>7E</td>
<td>5D</td>
<td>BD</td>
<td>7D</td>
<td>D0</td>
<td>9D</td>
<td>68</td>
<td>BD</td>
<td>B9</td>
<td>DD</td>
<td>65</td>
<td>FD</td>
<td>B2</td>
</tr>
<tr>
<td>1E</td>
<td>35</td>
<td>3E</td>
<td>6E</td>
<td>5E</td>
<td>5F</td>
<td>7E</td>
<td>A1</td>
<td>9E</td>
<td>59</td>
<td>BE</td>
<td>CC</td>
<td>DE</td>
<td>66</td>
<td>FE</td>
<td>9F</td>
</tr>
<tr>
<td>1F</td>
<td>1F</td>
<td>3F</td>
<td>6F</td>
<td>5F</td>
<td>6D</td>
<td>7F</td>
<td>07</td>
<td>9F</td>
<td>46</td>
<td>BF</td>
<td>BC</td>
<td>DF</td>
<td>62</td>
<td>FF</td>
<td>FF</td>
</tr>
</tbody>
</table>
Appendix H. Access Control Points and Callable Services

The TKE workstation allows you to enable or disable callable service access control points. For systems that do not use the optional TKE Workstation, all access control points (current and new) are enabled in the DEFAULT Role with the appropriate licensed internal code on the PCI Cryptographic Coprocessor, PCI X Cryptographic Coprocessor, Crypto Express2 Coprocessor, or Crypto Express3 Coprocessor.

TKE Version 4.0 and higher

Access to services that are executed on the PCIXCC, CEX2C, or CEX3C is through Access Control Points in the DEFAULT Role. To execute callable services on the PCI X Cryptographic Coprocessor/Crypto Express2 Coprocessor, access control points must be enabled for each service in the DEFAULT Role.

New TKE users and non-TKE users have all access control points enabled. This is also true for new TKE V5.x users. If you are migrating from TKE V4.0, V4.1, or V4.2 to TKE V5.0 and have a PCIXCC/CEX2C/CEX3C, all your current access control points will remain the same and any new access control points for ICSF will not be enabled.

Note: Access control points DKYGENKY-DALL and DSG ZERO-PAD unrestricted hash length and PTR enhanced PIN security are always disabled in the DEFAULT role for all customers (TKE and Non-TKE). A TKE Workstation is required to enable these access control points.

Access control points added in ICSF FMID HCR7770 are:
- PKA Key Token Change RTNMK
- PKA Key Translate - from CCA RSA to SC Visa Format
- PKA Key Translate - from CCA RSA to SC ME Format
- PKA Key Translate - from CCA RSA to SC CRT Format
- PKA Key Translate - from source EXP KEK to target EXP KEK
- PKA Key Translate - from source IMP KEK to target EXP KEK
- PKA Key Translate - from source IMP KEK to target IMP KEK
- Symmetric Key Encipher/Decipher - Encrypted DES keys
- Symmetric Key Encipher/Decipher - Encrypted AES keys

In addition, access control point PKA Key Token Change was renamed to PKA Key Token Change RTCMK.

Access Control Points for HCR7751 are:
- Clear New AES Master Key Register (ISPF ACP)
- Load First AES Master Key Part (ISPF ACP)
- Combine AES Master Key Parts (ISPF ACP)
- Set AES Master Key (ISPF ACP)
- Multiple Clear Key Import/Multiple Secure Key Import - AES
- Symmetric Algorithm Encipher - Secure AES
- Symmetric Algorithm Decipher - Secure AES
- Symmetric Key Generate - AES, PKCSOEAP, PKCS- 1.2
- Symmetric Key Generate - AES, ZERO-PAD
Symmetric Key Import - AES, PKCS-OAEP, PKCS-1.2
Symmetric Key Import - AES, ZERO-PAD
Symmetric Key Export - AES, PKCS-OAEP, PKCS-1.2
Symmetric Key Export - AES, ZERO-PAD

These access control points require the Nov. 2008 or later licensed internal code (LIC).

Access Control Points for HCR7731 are:
• Remote Key Export - Generate or export a key for use by a CCA node
• Trusted Block Create - Activate an Inactive Trusted Key Block
• Trusted Block Create - Create a Trusted Key Block in Inactive Form
• PKA Key Generate - Permit Regeneration Data
• PKA Key Generate - Permit Regeneration Data for Retained Keys
• PTR Enhanced PIN Security
  Callable services affected by PTR enhanced PIN security:
  – Clear PIN Encrypt - PTR Enhanced PIN Security
  – Clear Pin Generate Alternate - PTR Enhanced PIN Security
  – Encrypted PIN Generate - PTR Enhanced PIN Security
  – Encrypted PIN Translate - PTR Enhanced PIN Security
  – Encrypted PIN Verify - PTR Enhanced PIN Security
  – PIN Change/Unblock - PTR Enhanced PIN Security

Access Control Points for HCR770B are:
• Diversified Key Generate - TDES-XOR
• Diversified Key Generate - TDESEMV2/TDESEMV4
• PIN Change/Unblock - change EMV PIN with OPINENC
• PIN Change/Unblock - change EMV PIN with IPINENC
• Transaction Validation - Generate
• Transaction Validation - Verify CSC-3
• Transaction Validation - Verify CSC-4
• Transaction Validation - Verify CSC-5
• Key Part Import - RETRKPR

Access Control Points for HCR770A are:
• CKDS Conversion Program
• Clear Key Import
• Decipher
• Digital Signature Verify
• DSG ZERO-PAD Unrestricted Hash Length
• Encipher
• Key Part Import - ADD-PART keyword
• Key Part Import - COMPLETE keyword
• NOCV Exporter
• NOCV Importer
• Prohibit Export Extended
• Public Key Encrypt
These access control points are only supported on the PCIXCC/CEX2C/CEX3C.

For the relationship between access control points and callable services, see Table 277 on page 706.

---

**TKE Version 3.1**

Access to services that are executed on the PCI Cryptographic Coprocessor is through Access Control Points in the DEFAULT Role. To execute callable services on the PCI Cryptographic Coprocessor, access control points must be enabled for each service in the DEFAULT Role. The ability to enable/disable access control points in the DEFAULT Role was introduced on OS/390 V2R10 through APAR OW46381 for the Trusted Key Entry Workstation. New TKE customers and Non-TKE customers have all access control points enabled. This is also true for brand new TKE V3.1 users (not converting from TKE V3.0).

**Note:** Access control point DKYGENKY-DALL is always disabled in the DEFAULT Role for all customers (TKE and Non-TKE). A TKE Workstation is required to enable this access control point for the Diversified Key Generate service.

For existing TKE V3.0 users, upgrading to TKE V3.1 (APAR OW46381 and its corresponding ECA), current (for the level of ICSF you are running) access control points in the DEFAULT Role are enabled. Any new access control points are disabled in the DEFAULT Role and must be enabled through TKE if the service is required.

**Notes:**
1. APAR OW46381 will update the TKE Host Code
2. ECA 186 will update the TKE Workstation Code
3. The latest or most current driver is required for the PCI Cryptographic Coprocessor licensed internal code for the S/390 G5 Enterprise Server or the S/390 G6 Enterprise Server
4. The latest or most current driver is required for the PCI Cryptographic Coprocessor licensed internal code for the IBM @server zSeries 900

All of the previously discussed components are required for complete access control point support.

Access to services which execute on the Cryptographic Coprocessor Feature is through SAF. Disablement through SAF is sufficient to prevent execution of a service by either the Cryptographic Coprocessor Feature or the PCI Cryptographic Coprocessor. For functions which can be executed on the PCI Cryptographic Coprocessor, enablement of the function requires that the function be enabled through SAF and through the access control point in the DEFAULT Role.

If you are on OS/390 V2 R10, using a TKE V3.0 workstation, access control points for new services (requiring APARs OW46380 and OW46382) will be disabled. Existing access control points will be enabled in the DEFAULT Role. APAR OW46381 must be installed to enable the OS/390 V2 R10 interface. This will allow the TKE Administrator to enable any new access control points for ICSF services that execute in the PCI Cryptographic Coprocessor under the DEFAULT Role.

Access Control Points (requiring APARs OW46380 and OW46382) for OS/390 V2R10 are:
- DATAM Key Management Control
**Note:** For existing TKE installations (upgrading to TKE V3.1), it is required that this access control point be enabled. Failure to do so will result in processing errors for Double MAC keys in Key Import, Key Export, and Key Generate.

- Diversified Key Generate - Single length or same halves
- Diversified Key Generate - CLR8-ENC
- Diversified Key Generate - TDES-ENC
- Diversified Key Generate - TDES-DEC
- Diversified Key Generate - SESS-XOR
- Diversified Key Generate - DKYGENKY-DALL

**Note:** This access control point is always disabled in the DEFAULT Role for all customers (TKE and Non-TKE). A TKE Workstation is required to enable the function.

- MAC Generate - For existing TKE installations, it is recommended that this access control point be enabled.
- MAC Verify - For existing TKE installations, it is recommended that this access control point be enabled.

Access Control Points for z/OS V1 R2 are:

- PKA Key Token Change
- Secure Messaging for Keys
- Secure Messaging for PINs

Access Control Points for z/OS V1 R3 are:

- UKPT - PIN Verify, PIN Translate

Access Control Points for APAR OW53666 are:

- Data Key Export - Unrestricted
- Data Key Import - Unrestricted
- Key Export - Unrestricted
- Key Import - Unrestricted
- Key Part Import - Unrestricted

---

**Callable Service Access Control Points**

If an access control point is disabled, the corresponding ICSF callable service will fail during execution with an access denied error.

*Table 277. Callable service access control points*

<table>
<thead>
<tr>
<th>Access Control Point</th>
<th>Callable Service</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Clear Key Import / Multiple Clear Key Import - DES</em></td>
<td>CSNCKI or CSNCKM</td>
</tr>
<tr>
<td><em>Clear Key Import / Multiple Clear Key Import - AES</em></td>
<td>CSNCKI, CSNCKM or CSNBSKM</td>
</tr>
<tr>
<td>Clear PIN Encrypt</td>
<td>CSNCPEN</td>
</tr>
<tr>
<td>Clear PIN Generate - 3624</td>
<td>CSNBPGN</td>
</tr>
<tr>
<td>Clear PIN Generate - GBP</td>
<td>CSNBPGN</td>
</tr>
<tr>
<td>Clear PIN Generate - VISA PVV</td>
<td>CSNBPGN</td>
</tr>
<tr>
<td>Clear PIN Generate - Interbank</td>
<td>CSNBPGN</td>
</tr>
<tr>
<td>Clear Pin Generate Alternate - 3624 Offset</td>
<td>CSNBCPA</td>
</tr>
<tr>
<td>Callable service access control points (continued)</td>
<td>Callable services</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Clear PIN Generate Alternate - VISA PVV</td>
<td>CSNCPA</td>
</tr>
<tr>
<td>Control Vector Translate</td>
<td>CSNBCVT</td>
</tr>
<tr>
<td>Cryptographic Variable Encipher</td>
<td>CSNBCVE</td>
</tr>
<tr>
<td>CVV Generate</td>
<td>CSNBCSG</td>
</tr>
<tr>
<td>CVV Verify</td>
<td>CSNBCSV</td>
</tr>
<tr>
<td>DATAM Key Management Control</td>
<td>CSNBKGN, CSNBKIM, CSNBKEX and CSNBDKG</td>
</tr>
<tr>
<td>Data Key Export</td>
<td>CSNBDKX</td>
</tr>
<tr>
<td>Data Key Export - Unrestricted</td>
<td>CSNBDKX</td>
</tr>
<tr>
<td>Data Key Import</td>
<td>CSNBDKM</td>
</tr>
<tr>
<td>Data Key Import - Unrestricted</td>
<td>CSNBDKM</td>
</tr>
<tr>
<td>*Decipher - DES</td>
<td>CSNBDEC</td>
</tr>
<tr>
<td>Digital Signature Generate</td>
<td>CSNDDSG</td>
</tr>
<tr>
<td>*DSG ZERO-PAD restriction lifted</td>
<td>CSNDDSG</td>
</tr>
<tr>
<td>*Digital Signature Verify</td>
<td>CSNDDSV</td>
</tr>
<tr>
<td>Diversified Key Generate - CLR8–ENC</td>
<td>CSNBDKG</td>
</tr>
<tr>
<td>Diversified Key Generate - SESS-XOR</td>
<td>CSNBDKG</td>
</tr>
<tr>
<td>Diversified Key Generate - TDES-ENC</td>
<td>CSNBDKG</td>
</tr>
<tr>
<td>Diversified Key Generate - TDES-DEC</td>
<td>CSNBDKG</td>
</tr>
<tr>
<td>***Diversified Key Generate - TDES-XOR</td>
<td>CSNBDKG</td>
</tr>
<tr>
<td>**Diversified Key Generate - TDESEMV2/ TDESEMV4</td>
<td>CSNBDKG</td>
</tr>
<tr>
<td>Diversified Key Generate - single length or same halves</td>
<td>CSNBDKG</td>
</tr>
<tr>
<td>DKYGENKY - DALL</td>
<td>CSNBDKG</td>
</tr>
<tr>
<td>*Encipher - DES</td>
<td>CSNBENC</td>
</tr>
<tr>
<td>Encrypted PIN Generate - 3624</td>
<td>CSNBEPG</td>
</tr>
<tr>
<td>Encrypted PIN Generate - GBP</td>
<td>CSNBEPG</td>
</tr>
<tr>
<td>Encrypted PIN Generate - Interbank</td>
<td>CSNBEPG</td>
</tr>
<tr>
<td>Encrypted PIN Translate - Translate</td>
<td>CSNBPTR</td>
</tr>
<tr>
<td>Encrypted PIN Translate - Reformat</td>
<td>CSNBPTR</td>
</tr>
<tr>
<td>Encrypted PIN Verify - 3624</td>
<td>CSNPBVPR</td>
</tr>
<tr>
<td>Encrypted PIN Verify - GBP</td>
<td>CSNPBVPR</td>
</tr>
<tr>
<td>Encrypted PIN Verify - VISA PVV</td>
<td>CSNPBVPR</td>
</tr>
<tr>
<td>Encrypted PIN Verify - Interbank</td>
<td>CSNPBVPR</td>
</tr>
<tr>
<td>Key Export</td>
<td>CSNBKEX</td>
</tr>
<tr>
<td>Key Export - Unrestricted</td>
<td>CSNBKEX</td>
</tr>
<tr>
<td>Key Generate - OPIM, OPEX, IMEX, etc.</td>
<td>CSNBKGN</td>
</tr>
<tr>
<td>Key Generate - EX, IM, OP</td>
<td>CSNBKGN</td>
</tr>
<tr>
<td>Key Generate - CVARs</td>
<td>CSNBKGN</td>
</tr>
<tr>
<td>Key Generate - SINGLE-R</td>
<td>CSNBKGN</td>
</tr>
<tr>
<td>Key Import</td>
<td>CSNBKIM</td>
</tr>
</tbody>
</table>
### Table 277. Callable service access control points (continued)

<table>
<thead>
<tr>
<th>Callable Service</th>
<th>Access Control Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Import - Unrestricted</td>
<td>CSNBKIM</td>
</tr>
<tr>
<td>*Key Part Import - ADD-PART</td>
<td>CSNBKPI</td>
</tr>
<tr>
<td>*Key Part Import - COMPLETE</td>
<td>CSNBKPI</td>
</tr>
<tr>
<td>Key Part Import - first key part</td>
<td>CSNBKPI</td>
</tr>
<tr>
<td>Key Part Import - middle and final</td>
<td>CSNBKPI</td>
</tr>
<tr>
<td>Key Part Import - unrestricted</td>
<td>CSNBKPI</td>
</tr>
<tr>
<td>Key Part Import - RETRKPR</td>
<td>CSNBKPI</td>
</tr>
<tr>
<td>Key Translate</td>
<td>CSNBKTR</td>
</tr>
<tr>
<td>MAC Generate</td>
<td>CSNBMGN</td>
</tr>
<tr>
<td>MAC Verify</td>
<td>CSNBMVR</td>
</tr>
<tr>
<td>*NOCV KEK usage for export-related functions</td>
<td>CSNBKEX, CSNBSKM, and CSNBKGN</td>
</tr>
<tr>
<td>*NOCV KEK usage for import-related functions</td>
<td>CSNBKIM, CSNBSKI, CSNBSKM, and CSNBKGN</td>
</tr>
<tr>
<td>*PCF CKDS Conversion Program</td>
<td>CSFCONV</td>
</tr>
<tr>
<td>**PIN Change/Unblock - change EMV PIN with OPINENC</td>
<td>CSNBPCU</td>
</tr>
<tr>
<td>**PIN Change/Unblock - change EMV PIN with IPINENC</td>
<td>CSNBPCU</td>
</tr>
<tr>
<td>**PIN Change/Unblock - PTR Enhanced PIN Security</td>
<td>CSNBPCU</td>
</tr>
<tr>
<td>PKA Decrypt</td>
<td>CSNDPKD</td>
</tr>
<tr>
<td>PKA Encrypt</td>
<td>CSNDPKE</td>
</tr>
<tr>
<td>PKA Key Generate</td>
<td>CSNDPKG</td>
</tr>
<tr>
<td>PKA Key Generate - Clear</td>
<td>CSNDPKG</td>
</tr>
<tr>
<td>PKA Key Generate - Clone</td>
<td>CSNDPKG</td>
</tr>
<tr>
<td>PKA Key Generate - Permit Regeneration Data Retain</td>
<td>CSNDPKG</td>
</tr>
<tr>
<td>PKA Key Generate - Permit Regeneration Data Retain</td>
<td>CSNDPKG</td>
</tr>
<tr>
<td>PKA Key Import</td>
<td>CSNDPKI</td>
</tr>
<tr>
<td>PKA Key Import - Import an External Trusted Key Block to internal form</td>
<td>CSNDPKI</td>
</tr>
<tr>
<td>PKA Key Token Change RTCMK</td>
<td>CSNDKTC</td>
</tr>
<tr>
<td>PKA Key Token Change RTNMK</td>
<td>CSNDKTC</td>
</tr>
<tr>
<td>PKA Key Translate - from CCA RSA to SC Visa Format</td>
<td>CSNDPKT</td>
</tr>
<tr>
<td>PKA Key Translate - from CCA RSA to SC ME Format</td>
<td>CSNDPKT</td>
</tr>
<tr>
<td>PKA Key Translate - from CCA RSA to SC CRT Format</td>
<td>CSNDPKT</td>
</tr>
<tr>
<td>PKA Key Translate - from source EXP KEK to target EXP KEK</td>
<td>CSNDPKT</td>
</tr>
<tr>
<td>PKA Key Translate - from source IMP KEK to target EXP KEK</td>
<td>CSNDPKT</td>
</tr>
</tbody>
</table>
Table 277. Callable service access control points (continued)

<table>
<thead>
<tr>
<th>Callable Service Description</th>
<th>Access Control Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKA Key Translate - from source IMP KEK to target IMP KEK</td>
<td>CSNDPKT</td>
</tr>
<tr>
<td>Prohibit Export</td>
<td>CSNBPEX</td>
</tr>
<tr>
<td>*Prohibit Export Extended</td>
<td>CSNBPEXX</td>
</tr>
<tr>
<td>PTR Enhanced PIN Security</td>
<td>CSNBCPE, CSNBCPA, CSNBEPG, CSNBPTR, CSNBPVR, and CSNBPCU</td>
</tr>
<tr>
<td>Remote Key Export - Generate or export a key for use by a non-CCA node</td>
<td>CSNDRKX</td>
</tr>
<tr>
<td>Retained Key Delete</td>
<td>CSNDRKD</td>
</tr>
<tr>
<td>Retained Key List</td>
<td>CSNDRKL</td>
</tr>
<tr>
<td>Secure Key Import - IM</td>
<td>CSNBSKI or CSNBSKM</td>
</tr>
<tr>
<td>Secure Key Import - OP</td>
<td>CSNBSKI or CSNBSKM</td>
</tr>
<tr>
<td>Secure Messaging for Keys</td>
<td>CSNBSKY</td>
</tr>
<tr>
<td>Secure Messaging for PINs</td>
<td>CSNBSPN</td>
</tr>
<tr>
<td>SET Block Compose</td>
<td>CSNDSBC</td>
</tr>
<tr>
<td>SET Block Decompose</td>
<td>CSNDSBD</td>
</tr>
<tr>
<td>SET Block Decompose - PIN ext IPINENC</td>
<td>CSNDSBD</td>
</tr>
<tr>
<td>SET Block Decompose - PIN ext OPINENC</td>
<td>CSNDSBD</td>
</tr>
<tr>
<td>Symmetric Algorithm Decipher - Secure AES</td>
<td>CSNBSAD or CSNBSAD1</td>
</tr>
<tr>
<td>Symmetric Algorithm Encipher - Secure AES</td>
<td>CSNBSAE or CSNBSAE1</td>
</tr>
<tr>
<td>Symmetric Key Export - AES, PKCS-1.2</td>
<td>CSNDSYX</td>
</tr>
<tr>
<td>Symmetric Key Export - DES, PKCS-1.2</td>
<td>CSNDSYX</td>
</tr>
<tr>
<td>Symmetric Key Export - AES, ZERO-PAD</td>
<td>CSNDSYX</td>
</tr>
<tr>
<td>Symmetric Key Export - DES, ZERO-PAD</td>
<td>CSNDSYX</td>
</tr>
<tr>
<td>Symmetric Key Encipher/Decipher - Encrypted DES keys</td>
<td>CSNBSYD or CSNBSYE</td>
</tr>
<tr>
<td>Symmetric Key Encipher/Decipher - Encrypted AES keys</td>
<td>CSNBSYD or CSNBSYE</td>
</tr>
<tr>
<td>Symmetric Key Generate - DES, PKA92</td>
<td>CSNDSYG</td>
</tr>
<tr>
<td>Symmetric Key Generate - AES, PKCS-1.2</td>
<td>CSNDSYG</td>
</tr>
<tr>
<td>Symmetric Key Generate - DES, PKCS-1.2</td>
<td>CSNDSYG</td>
</tr>
<tr>
<td>Symmetric Key Generate - AES, ZERO-PAD</td>
<td>CSNDSYG</td>
</tr>
<tr>
<td>Symmetric Key Generate - DES, ZERO-PAD</td>
<td>CSNDSYG</td>
</tr>
<tr>
<td>Symmetric Key Import - DES, PKA92 KEK</td>
<td>CSNDSYI</td>
</tr>
<tr>
<td>Symmetric Key Import - AES, PKCS-1.2</td>
<td>CSNDSYI</td>
</tr>
<tr>
<td>Symmetric Key Import - AES, PKCS-1.2</td>
<td>CSNDSYI</td>
</tr>
<tr>
<td>Symmetric Key Import - AES, ZERO-PAD</td>
<td>CSNDSYI</td>
</tr>
<tr>
<td>Symmetric Key Import - DES, ZERO-PAD</td>
<td>CSNDSYI</td>
</tr>
<tr>
<td>**Transaction Validation - Generate</td>
<td>CSNBTRV</td>
</tr>
<tr>
<td>**Transaction Validation - Verify CSC-3</td>
<td>CSNBTRV</td>
</tr>
<tr>
<td>**Transaction Validation - Verify CSC-4</td>
<td>CSNBTRV</td>
</tr>
<tr>
<td>**Transaction Validation - Verify CSC-5</td>
<td>CSNBTRV</td>
</tr>
</tbody>
</table>
Table 277. Callable service access control points (continued)

<table>
<thead>
<tr>
<th>Callable Service Access Control Points</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trusted Block Create - Activate an Inactive Trusted Key Block</td>
<td>CSNDTBC</td>
</tr>
<tr>
<td>Trusted Block Create - Create Trusted Key Block in Inactive Form</td>
<td>CSNDTBC</td>
</tr>
<tr>
<td>UKPT - PIN Verify, PIN Translate</td>
<td>CSNBPRVR and CSNBPTR</td>
</tr>
</tbody>
</table>

Notes:

1. * indicates that the access control point is only available with a PCIXCC/CEX2C/CEX3C.
2. ** indicates that the access control point is only available with a PCIXCC/CEX2C/CEX3C and requires Requires May 2004 or later version of Licensed Internal Code (LIC).
3. To use PKA Key Generate - Clear or PKA Key Generate - Clone, the PKA Key Generate access control point must be enabled or the callable service will fail.
4. To use SET Block Decompose - PIN ext IPINENC or PIN ext OPINENC, the SET Block Decompose access control point must be enabled or the callable service will fail.
5. Diversified Key Generate - single length or same halves requires either Diversified Key Generate - TDES-ENC or Diversified Key Generate - TDES-DEC be enabled.
6. In order to use ATM Remote Key Loading, TKE users will have to enable the access control points for these functions:
   - Trusted Block Create - Activate an Inactive Trusted Key Block
   - Trusted Block Create - Create Trusted Key Block in Inactive Form
   - PKA Key Import - Import an External Trusted Key Block to internal form
   - Remote Key Export - Generate or export a key for use by a non-CCA node
Appendix I. Accessibility

Accessibility features help a user who has a physical disability, such as restricted
mobility or limited vision, to use software products successfully. The major
accessibility features in z/OS® enable users to:

- Use assistive technologies such as screen readers and screen magnifier
  software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen readers, function with the user
interfaces found in z/OS. Consult the assistive technology documentation for
specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to
for information about accessing TSO/E and ISPF interfaces. These guides describe
how to use TSO/E and ISPF, including the use of keyboard shortcuts or function
keys (PF keys). Each guide includes the default settings for the PF keys and
explains how to modify their functions.

z/OS information

z/OS information is accessible using screen readers with the BookServer/Library
Server versions of z/OS books in the Internet library at:

http://www.ibm.com/systems/z/os/zos/bkserv/
This information was developed for products and services offered in the USA.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
USA

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation  
Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.
Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

---

Programming Interface Information

This book documents intended Programming Interfaces that allow the customer to write programs to obtain the services of z/OS Integrated Cryptographic Service Facility.

---

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

- AIX
- AS/400
- CICS
- ES/3090
- ES/9000
- eServer
- IBM
- IBMLink
- Multiprise
- MVS
- MVS/ESA
- MVS/SP
- OS/390
- Parallel Sysplex
- Personal Security
- Processor Resource/Systems Manager
- PR/SM
- RACF
- Resource Link
- RMF
- S/370
- S/390
- S/390 Parallel Enterprise Server
- System/390
- VTAM
The following terms are trademarks of other companies:

<table>
<thead>
<tr>
<th>Term</th>
<th>Company Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>American Express</td>
<td>American Express Company</td>
</tr>
<tr>
<td>BSAFE</td>
<td>RSA Data Security, Incorporated</td>
</tr>
<tr>
<td>EMV</td>
<td>EMVCo LLC</td>
</tr>
<tr>
<td>MasterCard</td>
<td>MasterCard International, Incorporated</td>
</tr>
<tr>
<td>Netscape</td>
<td>Netscape Communications Corporation</td>
</tr>
<tr>
<td>SET</td>
<td>SET Secure Electronic Transaction, LLC</td>
</tr>
<tr>
<td>UNIX</td>
<td>The Open Group</td>
</tr>
<tr>
<td>VISA</td>
<td>VISA International Service Association</td>
</tr>
</tbody>
</table>

Other company, product, and service names may be trademarks or service marks of others.
Glossary

This glossary defines terms and abbreviations used in Integrated Cryptographic Service Facility (ICSF). If you do not find the term you are looking for, refer to the index of the appropriate Integrated Cryptographic Service Facility document or view IBM Glossary of Computing Terms located at:

http://www.ibm.com/ibm/terminology

This glossary includes terms and definitions from:
- IBM Glossary of Computing Terms. Definitions are identified by the symbol (D) after the definition.
- The American National Standard Dictionary for Information Systems, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies can be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.
- The Information Technology Vocabulary, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

Definitions specific to the Integrated Cryptographic Services Facility are labeled “In ICSF.”

A

access method services (AMS). The facility used to define and reproduce VSAM key-sequenced data sets (KSDS). (D)

Advanced Encryption Standard (AES). In computer security, the National Institute of Standards and Technology (NIST) Advanced Encryption Standard (AES) algorithm. The AES algorithm is documented in a draft Federal Information Processing Standard.

AES. Advanced Encryption Standard.

American National Standard Code for Information Interchange (ASCII). The standard code using a coded character set consisting of 7-bit characters (8 bits including parity check) that is used for information exchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

ANSI key-encrypting key (AKEK). A 64- or 128-bit key used exclusively in ANSI X9.17 key management applications to protect data keys exchanged between systems.

ANSI X9.17. An ANSI standard that specifies algorithms and messages for DES key distribution.

ANSI X9.19. An ANSI standard that specifies an optional double-MAC procedure which requires a double-length MAC key.

application program. (1) A program written for or by a user that applies to the user’s work, such as a program that does inventory control or payroll. (2) A program used to connect and communicate with stations in a network, enabling users to perform application-oriented activities. (D)

application program interface (API). (1) A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program. (D) (2) In ICSF, a callable service.

asymmetric cryptography. Synonym for public key cryptography. (D)

authentication pattern. An 8-byte pattern that ICSF calculates from the master key when initializing the cryptographic key data set. ICSF places the value of the authentication pattern in the header record of the cryptographic key data set.

authorized program facility (APF). A facility that permits identification of programs authorized to use restricted functions. (D)

C

callable service. A predefined sequence of instructions invoked from an application program, using a CALL instruction. In ICSF, callable services perform cryptographic functions and utilities.

CBC. Cipher block chaining.

CCA. Common Cryptographic Architecture.
CCF. Cryptographic Coprocessor Feature.

CDMF. Commercial Data Masking Facility.

CEDA. A CICS transaction that defines resources online. Using CEDA, you can update both the CICS system definition data set (CSD) and the running CICS system.

CEX2A. Crypto Express2 Accelerator
CEX2C. Crypto Express2 Coprocessor
CEX3A. Crypto Express3 Accelerator
CEX3C. Crypto Express3 Coprocessor

checksum. (1) The sum of a group of data associated with the group and used for checking purposes. (T) (2) In ICSF, the data used is a key part. The resulting checksum is a two-digit value you enter when you use the key-entry unit to enter a master key part or a clear key part into the key-storage unit.

Chinese Remainder Theorem (CRT). A mathematical theorem that defines a format for the RSA private key that improves performance.

CICS. Customer Information Control System.

cipher block chaining (CBC). A mode of encryption that uses the data encryption algorithm and requires an initial chaining vector. For encipher, it exclusively ORs the initial block of data with the initial control vector and then enciphers it. This process results in the encryption of both the input block and of the initial control vector that it uses on the next input block as the process repeats. A comparable chaining process works for decipher.

ciphertext. (1) In computer security, text produced by encryption. (2) Synonym for enciphered data. (D)

CKDS. Cryptographic Key Data Set.

clear key. Any type of encryption key not protected by encryption under another key.

CMOS. Complementary metal oxide semiconductor.

coexistence mode. An ICSF method of operation during which CUSP or PCF application program can run independently and simultaneously on the same ICSF system. A CUSP or PCF application program can run on ICSF in this mode if the application program has been reassembled.

Commercial Data Masking Facility (CDMF). A data-masking algorithm using a DES-based kernel and a key that is shortened to an effective key length of 40 DES key-bits. Because CDMF is not as strong as DES, it is called a masking algorithm rather than an encryption algorithm. Implementations of CDMF, when used for data confidentiality, are generally exportable from the USA and Canada.

Common Cryptographic Architecture: Cryptographic Application Programming Interface. Defines a set of cryptographic functions, external interfaces, and a set of key management rules that provide a consistent, end-to-end cryptographic architecture across different IBM platforms.

compatibility mode. An ICSF method of operation during which a CUSP or PCF application program can run on ICSF without recompiling it. In this mode, ICSF cannot run simultaneously with CUSP or PCF.

complementary keys. A pair of keys that have the same clear key value, are different but complementary types, and usually exist on different systems.

console. A part of a computer used for communication between the operator or maintenance engineer and the computer. (A)

count-area split. In systems with VSAM, the movement of the contents of some of the control intervals in a control area to a newly created control area in order to facilitate insertion or lengthening of a data record when there are no remaining free control intervals in the original control area. (D)

control block. (1) A storage area used by a computer program to hold control information. (I) Synonymous with control area. (2) The circuitry that performs the control functions such as decoding microinstructions and generating the internal control signals that perform the operations requested. (A)

control interval. A fixed-length area of direct-access storage in which VSAM stores records and creates distributed free space. Also, in a key-sequenced data set or file, the set of records pointed to by an entry in the sequence-set index record. The control interval is the unit of information that VSAM transmits to or from direct access storage. A control interval always comprises an integral number of physical records. (D)

control interval split. In systems with VSAM, the movement of some of the stored records in a control interval to a free control interval to facilitate insertion or lengthening of a record that does not fit in the original control interval. (D)

control statement input data set. A key generator utility program data set containing control statements that a particular key generator utility program job will process.

control statement output data set. A key generator utility program data set containing control statements to create the complements of keys created by the key generator utility program.

control vector. In ICSF, a mask that is exclusive ORed with a master key or a transport key before ICSF uses that key to encrypt another key. Control vectors ensure that keys used on the system and keys
distributed to other systems are used for only the cryptographic functions for which they were intended.

**CPACF.** CP Assist for Cryptographic Functions

**CP Assist for Cryptographic Functions.** Implemented on all z890, z990, z9 EC, z9 BC, z10 EC and z10 BC processors to provide SHA-1 secure hashing.

**cross memory mode.** Synchronous communication between programs in different address spaces that permits a program residing in one address space to access the same or other address spaces. This synchronous transfer of control is accomplished by a calling linkage and a return linkage.

**CRT.** Chinese Remainder Theorem.

**Crypto Express2 Coprocessor.** An asynchronous cryptographic coprocessor available on the z890, z990, z9 EC, z9 BC, z10 EC and z10 BC.

**Crypto Express3 Coprocessor.** An asynchronous cryptographic coprocessor available on z10 EC and z10 BC.

**cryptographic adapter (4755 or 4758).** An expansion board that provides a comprehensive set of cryptographic functions for the network security processor and the workstation in the TSS family of products.

**cryptographic coprocessor.** A microprocessor that adds cryptographic processing functions to specific z890, z990, z9 EC, z9 BC, z10 EC and z10 BC processors. The Cryptographic Coprocessor Feature is a tamper-resistant chip built into the processor board.

**cryptographic key data set (CKDS).** (1) A data set that contains the encrypting keys used by an installation. (D) (2) In ICSF, a VSAM data set that contains all the cryptographic keys. Besides the encrypted key value, an entry in the cryptographic key data set contains information about the key.

**cryptography.** (1) The transformation of data to conceal its meaning. (2) In computer security, the principles, means, and methods for encrypting plaintext and decrypting ciphertext. (D) (3) In ICSF, the use of cryptography is extended to include the generation and verification of MACs, the generation of MDCs and other one-way hashes, the generation and verification of PINS, and the generation and verification of digital signatures.

**CUSB (Cryptographic Unit Support Program).** The IBM cryptographic offering, program product 5740-XY6, using the channel-attached 3848, CUSB is no longer in service.

**CUSB/PCF conversion program.** A program, for use during migration from CUSB or PCF to ICSF, that converts a CUSB or PCF cryptographic key data set into an ICSF cryptographic key data set.

**Customer Information Control System (CICS).** An IBM licensed program that enables transactions entered at remote terminals to be processed concurrently by user written application programs. It includes facilities for building, using, and maintaining databases.

**CVC.** Card verification code used by MasterCard.

**CVV.** Card verification value used by VISA.

**D**

**data encryption algorithm (DEA).** In computer security, a 64-bit block cipher that uses a 64-bit key, of which 56 bits are used to control the cryptographic process and 8 bits are used for parity checking to ensure that the key is transmitted properly. (D)

**data encryption standard (DES).** In computer security, the National Institute of Standards and Technology (NIST) Data Encryption Standard, adopted by the U.S. government as Federal Information Processing Standard (FIPS) Publication 46, which allows only hardware implementations of the data encryption algorithm. (D)

**data key or data-encrypting key.** (1) A key used to encipher, decipher, or authenticate data. (D) (2) In ICSF, a 64-bit encryption key used to protect data privacy using the DES algorithm or the CDMF algorithm. AES data keys are now supported by ICSF.

**data set.** The major unit of data storage and retrieval, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access. (D)

**data-translation key.** A 64-bit key that protects data transmitted through intermediate systems when the originator and receiver do not share the same key.

**DEA.** Data encryption algorithm.

**decipher.** (1) To convert enciphered data in order to restore the original data. (T) (2) In computer security, to convert ciphertext into plaintext by means of a cipher system. (3) To convert enciphered data into clear data. Contrast with encipher. Synonymous with decrypt. (D)

**decode.** (1) To convert data by reversing the effect of some previous encoding. (I) (A) (2) In ICSF, to decipher data by use of a clear key.

**decrypt.** See decipher.

**DES.** Data Encryption Standard.
**diagnostics data set.** A key generator utility program data set containing a copy of each input control statement followed by a diagnostic message generated for each control statement.

**digital signature.** In public key cryptography, information created by using a private key and verified by using a public key. A digital signature provides data integrity and source nonrepudiation.

**Digital Signature Standard (DSS).** A standard describing the use of algorithms for digital signature purposes. One of the algorithms specified is DSA (Digital Signature Algorithm).

**domain.** (1) That part of a network in which the data processing resources are under common control. (T) (2) In ICSF, an index into a set of master key registers.

**double-length key.** A key that is 128 bits long. A key can be either double- or single-length. A single-length key is 64 bits long.

**DSA.** Digital Signature Algorithm.

**DSS.** Digital Signature Standard.

**E**

**ECB.** Electronic codebook.

**ECI.** Eurochèque International S.C., a financial institution consortium that has defined three PIN block formats.

**EID.** Environment Identification.

**electronic codebook (ECB) operation.** (1) A mode of operation used with block cipher cryptographic algorithms in which plaintext or ciphertext is placed in the input to the algorithm and the result is contained in the output of the algorithm. (D) (2) A mode of encryption using the data encryption algorithm, in which each block of data is enciphered or deciphered without an initial chaining vector. It is used for key management functions and the encode and decode callable services.

**electronic funds transfer system (EFTS).** A computerized payment and withdrawal system used to transfer funds from one account to another and to obtain related financial data. (D)

**encipher.** (1) To scramble data or to convert data to a secret code that masks the meaning of the data to any unauthorized recipient. Synonymous with encrypt. (2) Contrast with decipher. (D)

**enciphered data.** Data whose meaning is concealed from unauthorized users or observers. (D)

**encode.** (1) To convert data by the use of a code in such a manner that reconversion to the original form is possible. (T) (2) In computer security, to convert plaintext into an unintelligible form by means of a code system. (D) (3) In ICSF, to encipher data by use of a clear key.

**encrypt.** See encipher.

**exit.** (1) To execute an instruction within a portion of a computer program in order to terminate the execution of that portion. Such portions of computer programs include loops, subroutines, modules, and so on. (T) (2) In ICSF, a user-written routine that receives control from the system during a certain point in processing—for example, after an operator issues the START command.

**exportable form.** A condition a key is in when enciphered under an exporter key-encrypting key. In this form, a key can be sent outside the system to another system. A key in exportable form cannot be used in a cryptographic function.

**exporter key-encrypting key.** A 128-bit key used to protect keys sent to another system. A type of transport key.

**F**

**file.** A named set of records stored or processed as a unit. (T)

**G**

**GBP.** German Bank Pool.

**German Bank Pool (GBP).** A German financial institution consortium that defines specific methods of PIN calculation.

**H**

**hashing.** An operation that uses a one-way (irreversible) function on data, usually to reduce the length of the data and to provide a verifiable authentication value (checksum) for the hashed data.

**header record.** A record containing common, constant, or identifying information for a group of records that follows. (D)

**I**

**ICSF.** Integrated Cryptographic Service Facility.

**importable form.** A condition a key is in when it is enciphered under an importer key-encrypting key. A key is received from another system in this form. A key in importable form cannot be used in a cryptographic function.
importer key-encrypting key. A 128-bit key used to protect keys received from another system. A type of transport key.

initial chaining vector (ICV). A 64-bit random or pseudo-random value used in the cipher block chaining mode of encryption with the data encryption algorithm.

initial program load (IPL). (1) The initialization procedure that causes an operating system to commence operation. (2) The process by which a configuration image is loaded into storage at the beginning of a work day or after a system malfunction. (3) The process of loading system programs and preparing a system to run jobs. (D)

input PIN-encrypting key. A 128-bit key used to protect a PIN block sent to another system or to translate a PIN block from one format to another.

installation exit. See exit.

Integrated Cryptographic Service Facility (ICSF). A licensed program that runs under MVS/System Product 3.1.3, or higher, or OS/390 Release 1, or higher, or z/OS, and provides access to the hardware cryptographic feature for programming applications. The combination of the hardware cryptographic feature and ICSF provides secure high-speed cryptographic services.

International Organization for Standardization. An organization of national standards bodies from many countries, established to promote the development of standards to facilitate the international exchange of goods and services and to develop cooperation in intellectual, scientific, technological, and economic activity. ISO has defined certain standards relating to cryptography and has defined two PIN block formats.

ISO. International Organization for Standardization.

J

job control language (JCL). A control language used to identify a job to an operating system and to describe the job's requirements. (D)

K

key-encrypting key (KEK). (1) In computer security, a key used for encryption and decryption of other keys. (D) (2) In ICSF, a master key or transport key.

key generator utility program (KGUP). A program that processes control statements for generating and maintaining keys in the cryptographic key data set.

key output data set. A key generator utility program data set containing information about each key that the key generator utility program generates except an importer key for file encryption.

key part. A 32-digit hexadecimal value that you enter for ICSF to combine with other values to create a master key or clear key.

key part register. A register in the key storage unit that stores a key part while you enter the key part.

key store policy. Ensures that only authorized users and jobs can access secure key tokens that are stored in one of the ICSF key stores - the CKDS or the PKDS.

key store policy controls. Resources that are defined in the XFACILIT class. A control can verify the caller has authority to use a secure token and identify the action to take when the secure token is not stored in the CKDS or PKDS.

L

linkage. The coding that passes control and parameters between two routines.

load module. All or part of a computer program in a form suitable for loading into main storage for execution. A load module is usually the output of a linkage editor. (T)

LPAR mode. The central processor mode that enables the operator to allocate the hardware resources among several logical partitions.

M

MAC generation key. A 64-bit or 128-bit key used by a message originator to generate a message authentication code sent with the message to the message receiver.

MAC verification key. A 64-bit or 128-bit key used by a message receiver to verify a message authentication code received with a message.

magnetic tape. A tape with a magnetizable layer on which data can be stored. (T)

master key. (1) In computer security, the top-level key in a hierarchy of key-encrypting keys. (2) ICSF uses master keys to encrypt operational keys. Master keys are known only to the cryptographic coprocessors and are maintained in tamper proof cryptographic coprocessors. Examples of cryptographic coprocessors are CCF, PCC, PCCXCC, CEX2C, and CEX3C. Some of the master keys that ICSF supports are a 128-bit DES master key, a 192-bit signature master key, and the 192-bit key management master key, a 192-bit symmetric master key (that is, DES), a 192-bit asymmetric master key, and a 256-bit AES master key.

master key concept. The idea of using a single cryptographic key, the master key, to encrypt all other keys on the system.
**master key register.** A register in the cryptographic coprocessors that stores the master key that is active on the system.

**master key variant.** A key derived from the master key by use of a control vector. It is used to force separation by type of keys on the system.

**MD4.** Message Digest 4. A hash algorithm.

**MD5.** Message Digest 5. A hash algorithm.

**message authentication code (MAC).** (1) The cryptographic result of block cipher operations on text or data using the cipher block chain (CBC) mode of operation. (D) (2) In ICSF, a MAC is used to authenticate the source of the message, and verify that the message was not altered during transmission or storage.

**modification detection code (MDC).** (1) A 128-bit value that interrelates all bits of a data stream so that the modification of any bit in the data stream results in a new MDC. (2) In ICSF, an MDC is used to verify that a message or stored data has not been altered.

**multiple encipherment.** The method of encrypting a key under a double-length key-encrypting key.

**new master key register.** A register in the key storage unit that stores a master key before you make it active on the system.

**NIST.** U.S. National Institute of Science and Technology.

**NOCV processing.** Process by which the key generator utility program or an application program encrypts a key under a transport key itself rather than a transport key variant.

**noncompatibility mode.** An ICSF method of operation during which CUSP or PCF can run independently and simultaneously on the same z/OS, OS/390 or MVS system. You cannot run a CUSP or PCF application program on ICSF in this mode.

**nonrepudiation.** A method of ensuring that a message was sent by the appropriate individual.

**notarization.** The ANSI X9.17 process involving the coupling of an ANSI key-encrypting key (AKEK) with ASCII character strings containing origin and destination identifiers and then exclusive ORing (or offsetting) the result with a binary counter.

**offset.** The process of exclusively ORing a counter to a key.

**old master key register.** A register in the key storage unit that stores a master key that you replaced with a new master key.

**operational form.** The condition of a key when it is encrypted under the master key so that it is active on the system.

**output PIN-encrypting key.** A 128-bit key used to protect a PIN block received from another system or to translate a PIN block from one format to another.

**PAN.** Personal Account Number.

**parameter.** Data passed between programs or procedures. (D)

**parmlib.** A system parameter library, either SYS1.PARMLIB or an installation-supplied library.

**partial notarization.** The ANSI X9.17 standard does not use the term partial notarization. IBM has divided the notarization process into two steps and defined the term partial notarization as a process during which only the first step of the two-step ANSI X9.17 notarization process is performed. This step involves the coupling of an ANSI key-encrypting key (AKEK) with ASCII character strings containing origin and destination identifiers.

**partitioned data set (PDS).** A data set in direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data. (D)

**PCI Cryptographic Coprocessor.** The 4758 model 2 standard PCI-bus card supported on the field upgraded IBM S/390 Parallel Enterprise Server - Generation 5, the IBM S/390 Parallel Enterprise Server - Generation 6 and the IBM @server zSeries.

**PCICA.** PCI Cryptographic Accelerator.

**PCICC.** PCI Cryptographic Coprocessor.

**PCI X Cryptographic Coprocessor.** An asynchronous cryptographic coprocessor available on the IBM @server zSeries 990 and IBM @server zSeries 800.

**PCIXCC.** PCI X Cryptographic Coprocessor.

**Personal Account Number (PAN).** A Personal Account Number identifies an individual and relates that individual to an account at a financial institution. It consists of an issuer identification number, customer account number, and one check digit.
personal identification number (PIN). The 4- to 12-digit number entered at an automatic teller machine to identify and validate the requester of an automatic teller machine service. Personal identification numbers are always enciphered at the device where they are entered, and are manipulated in a secure fashion.

Personal Security card. An ISO-standard “smart card” with a microprocessor that enables it to perform a variety of functions such as identifying and verifying users, and determining which functions each user can perform.

PIN block. A 64-bit block of data in a certain PIN block format. A PIN block contains both a PIN and other data.

PIN generation key. A 128-bit key used to generate PINs or PIN offsets algorithmically.

PIN key. A 128-bit key used in cryptographic functions to generate, transform, and verify the personal identification numbers.

PIN offset. For 3624, the difference between a customer-selected PIN and an institution-assigned PIN. For German Bank Pool, the difference between an institution PIN (generated with an institution PIN key) and a pool PIN (generated with a pool PIN key).

PIN verification key. A 128-bit key used to verify PINs algorithmically.

PKA. Public Key Algorithm.

PKCS. Public Key Cryptographic Standards (RSA Data Security, Inc.)

PKDS. Public key data set (PKA cryptographic key data set).

plaintext. Data in normal, readable form.

primary space allocation. An area of direct access storage space initially allocated to a particular data set or file when the data set or file is defined. See also secondary space allocation. (D)

private key. In computer security, a key that is known only to the owner and used with a public key algorithm to decrypt data or generate digital signatures. The data is encrypted and the digital signature is verified using the related public key.

processor complex. A configuration that consists of all the machines required for operation.

Processor Resource/Systems Manager. Enables logical partitioning of the processor complex, may provide additional byte-multiplexor channel capability, and supports the VM/XA System Product enhancement for Multiple Preferred Guests.

Programmed Cryptographic Facility (PCF). (1) An IBM licensed program that provides facilities for enciphering and deciphering data and for creating, maintaining, and managing cryptographic keys. (D) (2) The IBM cryptographic offering, program product 5740-XY5, using software only for encryption and decryption. This product is no longer in service; ICSF is the replacement product.

PR/SM. Processor Resource/Systems Manager.

public key. In computer security, a key made available to anyone who wants to encrypt information using the public key algorithm or verify a digital signature generated with the related private key. The encrypted data can be decrypted only by use of the related private key.

public key algorithm (PKA). In computer security, an asymmetric cryptographic process in which a public key is used for encryption and digital signature verification and a private key is used for decryption and digital signature generation.

public key cryptography. In computer security, cryptography in which a public key is used for encryption and a private key is used for decryption. Synonymous with asymmetric cryptography.

R

RACE Integrity Primitives Evaluation Message Digest. A hash algorithm.

RDO. Resource definition online.

record chaining. When there are multiple cipher requests and the output chaining vector (OCV) from the previous encipher request is used as the input chaining vector (ICV) for the next encipher request.

Resource Access Control Facility (RACF). An IBM licensed program that provides for access control by identifying and verifying the users to the system, authorizing access to protected resources, logging the detected unauthorized attempts to enter the system, and logging the detected accesses to protected resources. (D)

retained key. A private key that is generated and retained within the secure boundary of the PCI Cryptographic Coprocessor.

return code. (1) A code used to influence the execution of succeeding instructions. (A) (2) A value returned to a program to indicate the results of an operation requested by that program. (D)

Rivest-Shamir-Adleman (RSA) algorithm. A process for public key cryptography that was developed by R. Rivest, A. Shamir, and L. Adleman.

RMF. Resource Manager Interface.

RMI. Resource Measurement Facility.
RSA. Rivest-Shamir-Adleman.

SAF. Security Authorization Facility.

Save area. Area of main storage in which contents of registers are saved. (A)

Secondary space allocation. In systems with VSAM, area of direct access storage space allocated after primary space originally allocated is exhausted. See also primary space allocation. (D)


Secure Sockets Layer. A security protocol that provides communications privacy over the Internet by allowing client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

Sequential data set. A data set whose records are organized on the basis of their successive physical positions, such as on magnetic tape. (D)

SET. Secure Electronic Transaction.

SHA (Secure Hash Algorithm, FIPS 180). (Secure Hash Algorithm, FIPS 180) The SHA (Secure Hash Algorithm) family is a set of related cryptographic hash functions designed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST). The first member of the family, published in 1993, is officially called SHA. However, today, it is often unofficially called SHA-0 to avoid confusion with its successors. Two years later, SHA-1, the first successor to SHA, was published. Four more variants, have since been published with increased output ranges and a slightly different design: SHA-224, SHA-256, SHA-384, and SHA-512 (all are sometimes referred to as SHA-2).

SHA-1 (Secure Hash Algorithm 1, FIPS 180). A hash algorithm required for use with the Digital Signature Standard.

SHA-2 (Secure Hash Algorithm 2, FIPS 180). Four additional variants to the SHA family, with increased output ranges and a slightly different design: SHA-224, SHA-256, SHA-384, and SHA-512 (all are sometimes referred to as SHA-2).

SHA-224. One of the SHA-2 algorithms.

SHA-256. One of the SHA-2 algorithms.

SHA-384. One of the SHA-2 algorithms.

SHA-512. One of the SHA-2 algorithms.

Single-length key. A key that is 64 bits long. A key can be single- or double-length. A double-length key is 128 bits long.

Smart card. A plastic card that has a microchip capable of storing data or process information.


System Authorization Facility (SAF). An interface to a system security system like the Resource Access Control Facility (RACF).

System Management Facility (SMF). A base component of z/OS that provides the means for gathering and recording information that can be used to evaluate system usage. (D)

T

TDEA. Triple Data Encryption Algorithm.

TKE. Trusted key entry.

Transaction Security System. An IBM product offering including both hardware and supporting software that provides access control and basic cryptographic key-management functions in a network environment. In the workstation environment, this includes the 4755 Cryptographic Adapter, the Personal Security Card, the 4754 Security Interface Unit, the Signature Verification feature, the Workstation Security Services Program, and the AIX Security Services Program/6000. In the host environment, this includes the 4753 Network Security Processor and the 4753 Network Security Processor MVS Support Program.

Transport key. A 128-bit key used to protect keys distributed from one system to another. A transport key can either be an exporter key-encrypting key, an importer key-encrypting key, or an ANSI key-encrypting key.

Transport key variant. A key derived from a transport key by use of a control vector. It is used to force separation by type for keys sent between systems.
TRUE. Task-related User Exit (CICS). The CICS-ICSF Attachment Facility provides a CSFATRUE and CSFATREN routine.

U

UAT. UDX Authority Table.

UDF. User-defined function.

UDK. User-derived key.

UDP. User Developed Program.

UDX. User Defined Extension.

V

verification pattern. An 8-byte pattern that ICSF calculates from the key parts you enter when you enter a master key or clear key. You can use the verification pattern to verify that you have entered the key parts correctly and specified a certain type of key.

Virtual Storage Access Method (VSAM). An access method for indexed or sequential processing of fixed and variable-length records on direct-access devices. The records in a VSAM data set or file can be organized in logical sequence by means of a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry-sequence), or by means of relative-record number.

Virtual Telecommunications Access Method (VTAM). An IBM licensed program that controls communication and the flow of data in an SNA network. It provides single-domain, multiple-domain, and interconnected network capability. (D)

VISA. A financial institution consortium that has defined four PIN block formats and a method for PIN verification.

VISA PIN Verification Value (VISA PVV). An input to the VISA PIN verification process that, in practice, works similarly to a PIN offset.

Numerics

3621. A model of an IBM Automatic Teller Machine that has a defined PIN block format.

3624. A model of an IBM Automatic Teller Machine that has a defined PIN block format and methods of PIN calculation.

4753. The Network Security processor. The IBM 4753 is a processor that uses the Data Encryption Algorithm and the RSA algorithm to provide cryptographic support for systems requiring secure transaction processing (and other cryptographic services) at the host computer.

4758. The IBM PCI Cryptographic processor provides a secure programming and hardware environment where DES and RSA processes are performed.

The NSP includes a 4755 cryptographic adapter in a workstation which is channel attached to a S/390 host computer.
Index

Numerics
3621 PIN block format 305, 670
3624 PIN block format 305, 670
4700-PAD processing rule 227, 236
4704-EPP PIN block format 305

A
accessibility 711
accessing
callable service 10
invocation requirements 8
affinity (IEAAFFN callable service) 10
AKEK key type 23
ALET (alternate entry point)
format 4
algorithm 48
3624 PIN generation 672
3624 PIN verification 675
DES 15, 48
GBP PIN generation 673
GBP PIN verification 677
GBP-PIN 337
GBP-PINO 337
IBM-PIN 337
IBM-PINO 337
PIN offset generation 674
PIN, detailed 672
PIN, general 51
PVV generation 678
PVV verification 679
VISA PIN 678
VISA-PVV 320, 337
VISA-PVV4 337
ANSI 9.9-1 algorithm 269
ANSI key-encrypting key (AKEK) 23
ANSI X3.106 processing rule 680
ANSI X9.17 EDC generate callable service (CSNAEGN)
format 471
overview 47
parameters 471
syntax 471
ANSI X9.17 key export callable service (CSNAKEX)
format 473
overview 47
parameters 473
syntax 473
ANSI X9.17 key import callable service (CSNAKIM)
format 478
overview 47
parameters 478
syntax 478
ANSI X9.17 key management 471
overview 46
ANSI X9.17 key translate callable service (CSNAKTR)
format 483
overview 47
ANSI X9.17 key translate callable service (CSNAKTR) (continued)
parameters 483
syntax 483
ANSI X9.17 key-encrypting key 21
ANSI X9.17 transport key partial notarize callable service (CSNATKN)
overview 47
ANSI X9.17 transport key partial notorize (CSNATKN)
format 488
parameters 488
syntax 488
ANSI X9.19 optional double MAC procedure 269
ANSI X9.23 processing rule 218, 227, 236, 680
ANSI X9.8 333
ANSI X9.8 PIN block format 669
ASCII to EBCDIC conversion
table 701
asymmetric_key parameter
remote key export callable service 187
asymmetric_key_length parameter
remote key export callable service 187
authenticating messages 269

C
c-variable encrypting key identifier parameter
cryptographic variable encipher callable service 96
call
successful 12
unsuccessful 12
callable service
ANSI X9.17 EDC generate (CSNAEGN) 47, 471
ANSI X9.17 key export (CSNAKEX) 47, 473
ANSI X9.17 key import (CSNAKIM) 47, 478
ANSI X9.17 key translate (CSNAKTR) 47, 483
ANSI X9.17 transport key partial notarize (CSNATKN) 47
ANSI X9.17 transport key partial notorize (CSNATKN) 488
character/nibble conversion (CSNBXBC and CSNBXCB) 433
ciphertext 60
ciphertext translate (CSNBCTT or CSNBCTT1) 219
clear key import (CSNBCKI) 24, 87
clear PIN encrypt (CSNBCPE) 52, 309
clear PIN generate (CSNBPGN) 53, 313
clear PIN generate alternate (CSNBCPA) 52, 317
code conversion (CSNBXEAE) 55
code conversion (CSNBXBC) 55
code conversion (CSNBXCB) 55
code conversion (CSNBXEA and CSNBXEAE) 435
code conversion (CSNBXEAE) 55
coding examples 649
Assembler H 653
C 649
COBOL 651
PL/1 656

© Copyright IBM Corp. 1997, 2009

727
Callable service (continued)
callable service (continued)
callable service (continued)
callable service (continued)
callable service (continued)
callable service (continued)
callable service (continued)
callable service (continued)
callable service (continued)
  transaction validation  54
  Transaction Validation (CSNBTRV)  365
  transform CDMF key (CSNBTCK)  27, 206
  translating ciphertext  48
  trusted block create (CSNDTBC)  27, 208
  User derived key (CSFUDK)  211
  using key types and key forms  11
  VISA CVV service generate (CSNBCSG)  369
  VISA CVV service verify (CSNBCSV)  373
  with ALETs (alternate entry point)  4
  X9.9 data editing (CSNB9ED)  55, 459
  CBC processing rule  218, 227, 236
  CDMF
    overview  45
    CDMF key, transforming
      algorithm  694
      callable service  206
    certificate length parameter
      remote key export callable service  183
    certificate parameter
      remote key export callable service  184
    certificate_parms parameter
      remote key export callable service  184
    certificate_parms_length parameter
      remote key export callable service  184
    chaining vector length parameter
      one-way hash generate callable service  288
      Symmetric MAC generate callable service  293
      Symmetric MAC verify callable service  297
    chaining vector parameter
      decipher callable service  228
      encipher callable service  237
      MAC generate callable service  274
      MAC verify callable service  279
      MDC generate callable service  284
      one-way hash generate callable service  288
      symmetric MAC generate callable service  293
      Symmetric MAC verify callable service  298
    changing control vectors  643
    character/nibble conversion callable service (CSNBXBC and CSNBXCB)
      format  433
      parameters  433
      syntax  433
    character/nibble conversion callable services (CSNBXBC and CSNBXCB)
      overview  55
    choosing between
      CSNBCTT and CSNBCTT1  219
      CSNBDEC and CSNBDEC1  224
      CSNBENC and CSNBENC1  233
      CSNBMDG and CSNBMDG1  282
      CSNBMGN and CSNBMGN1  271
      CSNBMRV and CSNBMRV1  276
      CSNBSYD and CSNBSYD1  241, 254, 290
      CSNBSYE and CSNBSYE1  262
      CSNESAE and CSNESAE1  247
  CIPHER
    keys  20
    cipher block chaining (CBC)  217
    cipher block chaining (CBC) (continued)
      mode  218
    cipher feedback (CFB)
      mode  217
    cipher text id parameter
      decipher callable service  246
      encipher callable service  252
    ciphertext
      cryptographic variable encipher callable service  97
      deciphering  48, 217
      encoding  239
      field  229, 238, 260, 268
      translating  48, 219
    ciphertext id parameter
      decipher callable service  228, 259
      encipher callable service  237, 267
    ciphertext parameter
      decipher callable service  226
      decode callable service  231
      encipher callable service  237
      encode callable service  240
    ciphertext translate callable service (CSNBCTT or CSNBCTT1)
      format  220
      parameters  220
      syntax  220
      using  60
    clear key
      deciphering data with  229
      definition  23
      enciphering  189
      encoding with  239
      encoding and decoding data with  48
      protecting  217
    clear key import callable service (CSNBCKI)
      format  87
      overview  24
      parameters  87
      syntax  87
    clear key length parameter
      multiple clear key import callable service  160, 163
    clear key parameter
      clear key import callable service  89
      decode callable service  230
      encode callable service  240
      multiple clear key import callable service  160, 163
      secure key import callable service  190
    clear PIN encrypt callable service (CSNBCPE)
      format  310
      syntax  310
    clear PIN encrypt service (CSNBCPE)
      parameters  310
    clear PIN generate alternate callable service (CSNBCPA)
      format  317
      overview  52
      parameters  317
      syntax  317
    clear PIN generate callable service (CSNBPGN)
      format  313
      parameters  314
clear PIN generate callable service (CSNBPGN) (continued)
syntax 313
clear PIN generate key identifier parameter 319
clear PIN generate callable service 314
clear text id parameter
decipher callable service 228, 246, 259
encipher callable service 237, 252, 267
clear text parameter
decipher callable service 228
decode callable service 231
encipher callable service 235
encode callable service 240
code conversion callable service (CSNBXEA and
CSNBXAE)
format 435
parameters 435
syntax 435
code conversion callable services (CSNBXEA and
CSNBXAE)
overview 55
code table parameter
character/nibble conversion callable service 434
code conversion callable service 436
coding examples 649
Assembler H 653
C 649
COBOL 651
PL/I 656
Commercial Data Masking Facility (CDMF) 217
control information
for digital signature generate 381
for digital signature verify 387
for diversified key generate 104
for key test 141
for key test extended 145
for MAC generate 273
for MAC verify 278, 297
for MDC generate 284
for multiple clear key import 160
for multiple secure key import 163
for one-way hash generate 287
for PKA key token build 401
for symmetric algorithm encipher 243, 249
for symmetric key encipher 256, 264
for symmetric key generate 197
for symmetric key import 202
for symmetric MAC generate 292
for user derived key 213
random number generate callable service 180
control vector
description 633
value 633
control vector generate (CSNBCVG)
parameters 90
control vector generate callable service (CSNBCVG)
format 90
overview 24
syntax 90
control vector parameter
control vector generate callable service 91
control vector translate callable service (CSNCVT)
format 92
overview 24
parameters 92
syntax 92
control vector, description of 16, 18
control vectors, changing 643
cryptographic feature
description xxvii
cryptographic key data set (CKDS)
held keys 21
storing keys 24, 43, 87
cryptographic variable encipher (CSNBCVE)
parameters 96
cryptographic variable encipher callable service
(CSNBCVE)
format 96
overview 24
syntax 96
CSFIQA callable service 437
CSFIQF callable service 442
CSFIQF6 437, 442
CSFPCI callable service 463
CSFPDMK callable service 493
CSFPDVK callable service 496
CSFPGAV callable service 500
CSFPGKP callable service 503
CSFPGSK callable service 505
CSFPHMG callable service 507
CSFPHMV callable service 511
CSFPKSC callable service 467
CSFPOWH callable service 514
CSFPKKS callable service 518
CSFPKPV callable service 520
CSFPKRF callable service 523
CSFPKSAV callable service 526
CSFPKSKD callable service 528
CSFPKSKF callable service 532
CSFPKTRC callable service 537
CSFPKTRD callable service 541
CSFPKTRL callable service 543
CSFPUWK callable service 547
CSFPWPK callable service 550
CSFUDK callable service 211
CSFxxxx format 3
CSNAEGN callable service 471
CSNAKEX callable service 473, 478
CSNAKTR callable service 483
CSNATKN callable service 488
CSNB9ED callable service 459
CSNBCKI callable service 87
CSNBCKM callable service 159
CSNBCPA callable service 317
CSNBCPE callable service 309
CSNBCSG callable service 369
CSNBCSV callable service 373
CSNBCTT or CSNBCTT1 callable service 219
CSNBCVE callable service 95
CSNBCVG callable service 89
CSNBCVT callable service 92
CSNBDCO callable service 229
CSNBDEC or CSNBDEC1 callable service 223
CSNBDKG callable service 102
CSNBDMK callable service 100
CSNBDKX callable service 98
CSNBECO callable service 229
CSNBENC or CSNBENC1 callable service 231
CSNBEPG callable service 323
CSNBKEX callable service 107
CSNBKGN callable service 111
CSNBKIM callable service 123
CSNBKPI callable service 127
CSNBKRC callable service 131
CSNBKRD callable service 133
CSNBKRR callable service 135
CSNBKRW callable service 137
CSNBKTB callable service 147
CSNBKTR callable service 156
CSNBKYT callable service 139
CSNBKTYX callable service 144
CSNBMDG or CSNBMDG1 callable service 281
CSNBMGN or CSNBMGN1 callable service 271
CSNBMV or CSNBMV1 callable service 276
CSNBOWH, CSNEOWH and CSNBOWH1 callable services 286
CSNBPCU callable service 341
CSNBPEX callable service 175
CSNBPEXX callable service 177
CSNBPGN callable service 313
CSNBPTR callable service 328
CSNBPVR callable service 335
CSNBRRNG callable service 178
CSNBRRNGL callable service 178
CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1 callable service 247
CSNBISKI callable service 189
CSNBISKM callable service 161
CSNBISKY callable service 347
CSNBSMG, CSNBSMG1, CSNESMG, and CSNESMG1 callable service 290
CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1 callable service 294
CSNBSPN callable service 350
CSNBSYD and CSNBSYD callable service 241
CSNBSYD and CSNBSYD1 callable service 253
CSNBSYE and CSNBSYE1 callable service 260
CSNBTCX callable service 206
CSNBTRV callable service 365
CSNBXAE callable service 435
CSNBXBC callable service 433
CSNBXCB callable service 433
CSNBXE callable service 435
CSNbxxxx format 3
CSNDDS available service 379
CSNDDSV available service 385
CSNDKRC callable service 418
CSNDRK callable service 420
CSNDRKK callable service 422
CSNDFKRW callable service 424
CSNDFKTC callable service 409
CSNDPKB callable service 399
CSNDPKD callable service 166

CSNDPKE callable service 171
CSNDPKG callable service 391
CSNDPKI callable service 395
CSNDPKT callable service 412
CSNDPXX callable service 415
CSNDRKD callable service 426
CSNDRKL callable service 429
CSNDSBC callable service 355
CSNDSBD callable service 360
CSNDSYG callable service 196
CSNDSY1 callable service 201
CSNDSYX callable service 192
CSNDSYBC callable service 208
CSNECKI 88
CSNECKM 159
CSNEDEC 224
CSNEENC 233
CSNEKGN 112
CSNEOWH 286
CSNERNG 179
CSNESYD 254
CSNESYE 261
CSNFRK 418
CSNFRKD 420
CSNFPKB 400
CSNFPKD 166
CSNFPKE 171
CSNFPKG 392
CSNFPKI 396
CSNFPKX 415
CSNFRKD 426
CSNFRKL 429
CUSP processing rule 227, 236, 681
CUSP/IPS processing rule 218

D

data
deciphering 223
ciphering 231
ciphering and deciphering 48
encoding and decoding 48
protecting 217
data array parameter
clear PIN generate alternate callable service 321
clear PIN generate callable service 315
ciphered PIN generate callable service 325
ciphered PIN verify callable service 338
data integrity
ensuring 49
verifying 269
data key
exporting 98
importing 87
reencrypting 98
data key export callable service (CSNBDKX)
format 98
overview 25
parameters 98
syntax 98
data key import callable service (CSNBDKM)
  format 100
  overview 25
  parameters 100
  syntax 100
DATA key type 23
data length parameter
diversified key generate callable service 105
data space
callable services that use data in data spaces 4
data-encrypting key 19
data-translation key 19, 219
DATAM key type 23
DATAMV key type 23
DATAXLAT key type 23
decipher callable service (CSNBDEC or CSNBDEC1)
  format 224
  syntax 224
deciphering
data 217, 223
data with clear key 229
  multiple 684
decode callable service (CSNBDCO)
  format 229
  parameters 229
  syntax 229
DES algorithm 15, 48, 217
DES external key token format 599
DES internal key token format 597
destination identifier 47
digital signature generate callable service (CSNDDSG)
  format 379
  overview 71
  parameters 379
  syntax 379
digital signature verify callable service (CSNDDSV)
  format 385
  overview 71
  parameters 385
  syntax 385
disability 711
diversified key generate callable service (CSNBDKG)
  format 102
  overview 25
  parameters 102
  syntax 102
double-length key
  multiple deciphertext 688
  multiple encryption 687
  using 21
DSS private external key token 613
DSS private internal key token 615
DSS public token 613
dynamic CKDS update callable services
description 43

E
EBCDIC to ASCII conversion
table 701
ECI-1 333
exporter key-encrypting key (continued)
enciphering data key 98
exporting keys
  trusted blocks 37
external key token 7, 18, 77
  DES 599
  PKA 79
  DSS private 613
  RSA private 603
extra_data parameter
  remote key export callable service 188
extra_data_length parameter
  remote key export callable service 187
extraction rules, PIN 671

F
FEATURE=CRYPTO keyword
  SCHEDULE macro 10
form parameter
  random number generate callable service 180
format control 308
formats, PIN 51
functions of
  cryptographic keys 15
  ICSF 15

G
GBP-PIN algorithm 337
GBP-PINO algorithm 337
generated key identifier 1 parameter
  key generate callable service 118
generated key identifier 2 parameter
  key generate callable service 119
generated key identifier parameter
  diversified key generate callable service 106
generating an error detection code (EDC) 471
generating encrypted keys 111
generating key identifier parameter
  diversified key generate callable service 105
generating keys
  remote key export 39
German Banking Pool PIN algorithm 673
get attribute value callable service (CSFPGAV)
  format 500
  parameters 500
  syntax 500

H
hash length parameter
  digital signature generate callable service 381
digital signature verify callable service 388
one-way hash generate callable service 289
hash parameter
  digital signature generate callable service 382
digital signature verify callable service 388
one-way hash generate callable service 289
HEXDIGIT PIN extraction method keyword 306
high-level languages 3

I
IBM 3624 313, 335
IBM 4700 processing rule 218, 680
IBM GBP 313, 335
IBM-4700 PIN block format 670
IBM-PIN algorithm 337
IBM-PINO algorithm 337
ICSF
  functions 15
  overview 15
ICSF Query Algorithm (CSFIQA)
  parameters 437
  syntax 437
ICSF Query Algorithm (CSFIQA)
  format 437
ICSF Query Algorithm Service (CSFIQA)
  overview 55
ICSF Query Facility (CSFIQF)
  parameters 442
  syntax 442
ICSF Query Facility (CSFIQF)
  format 442
ICSF Query Facility Service (CSFIQF)
  overview 55
IEAFFFN callable service (affinity) 10
IM key form 57
IMEX key form 59
IMIM key form 58
importable key form 17
definition 17
generating 57
value 113
imported key identifier length parameter
  multiple secure key import callable service 164
imported key identifier parameter
  multiple secure key import callable service 164
importer key identifier parameter
  key import callable service 124
  secure key import callable service 191
IMPORTER key type 23
importer key-encrypting key 20
enciphering clear key 189, 191
importer_key_identifier parameter
  remote key export callable service 186
importer_key_length parameter
  remote key export callable service 186, 187
importing a non-exportable key 177
improving performance using partial notarization 693
INBK PIN 304, 313
INBK-PIN 335
Information Protection System (IPS) 682
initial chaining vector (ICV)
  description 218, 680
initialization vector in parameter
ciphertext translate callable service 221
initialization vector out parameter
ciphertext translate callable service 221
initialization vector parameter
cryptographic variable encipher callable service 97
decipher callable service 226
encipher callable service 235

Index 733
initialization vector parameter (continued)
key token build callable service 153
input data transport key 219
input KEK key identifier parameter
  key translate callable service 157
input PIN profile parameter
  clear PIN generate alternate callable service 319
  encrypted PIN translate callable service 329
  encrypted PIN verify callable service 336
input PIN-encrypting key identifier parameter
  encrypted PIN translate callable service 329
  encrypted PIN verify callable service 336
input_block parameter
  trusted block create callable service 210
input_block_identifier parameter
  trusted block create callable service 210
installation exit
  post-processing 7
  preprocessing 7
installation-defined callable service 15
Integrated Cryptographic Service Facility (ICSF)
description xxvii
Integrity 618
Interbank PIN 66, 304, 313, 335
internal key token 7, 17, 77, 78
aes; 597
DES 597, 598
PKA
  DSS private 615
  RSA private 608, 609, 610
invocation requirements 8
IPINENC key type 23, 329
IPS processing rule 227, 236, 682
ISO-0 PIN block format 305
ISO-1 PIN block format 305, 670
ISO-2 PIN block format 305, 670
ISO-3 PIN block format 305

J
JCL statements, sample 13

K
KEK key identifier parameter
  control vector translate callable service 93
KEK key identifier 1 parameter
  key generate callable service 117
KEK key identifier 2 parameter
  key generate callable service 118
KEK key identifier parameter
  key test extended callable service 146
  prohibit export extended callable service 178
  transform CDMF key callable service 207
key array parameter
  control vector translate callable service 93
key array right parameter
  control vector translate callable service 93
key encrypting key identifier parameter 198
key export callable service (CSNBKEX) (continued)
  overview 25
  parameters 107
  syntax 107
key export callable service (CSNBKEX)
  overview 25
  parameters 107
  syntax 107
key flow 17
key form
  combinations for a key pair 120
  combinations with key type 120
  definition 16
  exportable 17
  importable 17
  operational 16, 17
  value 113
key form parameter
  key generate callable service 113
  secure key import callable service 191
key generate callable service (CSNBKGN)
  format 111
  overview 24
  parameters 111
  syntax 111
  using 56
key generator utility program (KGUP)
  description 24
key identifier 7
  PKA keys 77
key identifier in parameter
  ciphertext translate callable service 221
key identifier length parameter
  key generate callable service 114
key identifier out parameter
  ciphertext translate callable service 221
key identifier parameter
  clear key import callable service 89
  decipher callable service 225
  diversified key generate callable service 106
  encipher callable service 234
  key test callable service 142
  key test extended callable service 145
  MAC generate callable service 273
  MAC generation callable service 278
  multiple clear key import callable service 160
  secure key import callable service 191
  Symmetric MAC generate callable service 292
  symmetric MAC verify callable service 296
key identifier parameter
  key generate callable service (CSNBKIM)
  format 123
  overview 25
  parameters 123
  syntax 123
key label 8, 77
  security considerations 10
key length parameter
  key generate callable service 114
key management
  ANSI X9.17 standard 471
key pair 120
key part import callable service (CSNBKPI)
  format 127
keys (continued)
data-translation 19
double-length 58, 59
enciphered 191
export
values for keys 26
exporter key-encrypting 20
forms 17
generating
encrypted 111
values for keys 26
held in applications 21
held in CKDS 21
importer key-encrypting 20
key-encrypting 20
list of types 23
MAC 20
managing 87
master key variant 16
master, DES 19
master, AES 19
multiple decipherment/encipherment 684
pair 58, 59
parity 87
PIN 20
PIN-encrypting key 328
PKA master 69
Key Management Master Key (KMMK) 69
Signature Master Key (SMK) 69
possible forms 25
protecting 217
reenciphered 123
reenciphering 107
separation 15
single-length 57, 58
transport 20
transport key variant 16
types of 19
using 11
VISA PVV
generating 317

L
languages, high-level 3
large data object 681
linking callable services 13
local enciphered key token parameter 199

M
MAC
generation callable service 49
keys 20
length keywords 273, 278, 292, 297
managing 49
verification callable service 50
MAC generate callable service (CSNBMSGM or CSNBMSMN1) (continued)
format 272
parameters 272
MAC generate callable service (CSNBMSGM or CSNBMSMN1)
syntax 272
MAC key type 23
mac length parameter
Symmetric MAC generate callable service 294
Symmetric MAC verify callable service 298
mac parameter
MAC generate callable service 274
MAC verify callable service 280
Symmetric MAC generate callable service 294
Symmetric MAC verify callable service 298
MAC verify callable service (CSNBMSMV or CSNBMSMV1)
format 277
parameters 277
syntax 277
MACVER key type 23
managing keys 87
mask array left parameter
close vector translate callable service 93
mask array preparation 643
mask array right parameter
close vector translate callable service 94
master key
AES 19
changing
possible effect on internal key tokens 18
enciphered key 123
master key variant 16
master key, DES 19
MAXLEN keyword 221, 226, 235
MDC
generate callable service 51
length keywords 284
managing 51
mdc parameter
MDC generate callable service 285
message authentication
definition 49
message authentication code (MAC)
description 269
generating 269, 271, 290
verifying 269, 276, 294
messages
authenticating 269
migration consideration
return codes from PCF macros 6
mode, special secure 10
modes of operation 217
modification detection
definition 51
modification detection code (MDC)
generating 270, 281
verifying 270
multiple
decipherment 684
encipherment 684
multiple clear key import callable service (CSNBCKM) 159
format 159
overview 26, 28
multiple clear key import callable service (CSNCKM)
(continued)
parameters 159
syntax 159
multiple node network 219
multiple secure key import callable service
(CSNBSSKM) 161
format 162
overview 26, 28
parameters 162
syntax 162

N
notarization 47
Notices 713
null key token 18
format 601, 602
number, generated 178

O
object ion key (OPK) 628
offsetting 46, 693
one-way hash generate callable service (CSNBOWH
and CSNBOWH1)
overview 50
one-way hash generate callable service (CSNBOWH,
CSNEOWH and CSNBOWH1)
format 286
parameters 286
syntax 286
OP key form 57
operational key form 16
definition 17
generating 56
value 113
OPEX key form 58
OPIM key form 58
OPINENC key type 23, 329
OPK, object protection key 628
OPOP key form 58
origin identifier 47
output chaining vector (OCV)
description 219, 680
output data transport key 219
output KEK key identifier parameter
key translate callable service 158
output PIN profile parameter
encrypted PIN translate callable service 331
output PIN-encrypt translation key identifier parameter
encrypted PIN-encrypt translate callable service 329
overview of callable services 3

P
pad character parameter
encipher callable service 237
key token build callable service 153
pad digit 309
format 308
PADDIGIT PIN extraction method keyword 306
padding schemes 224, 232
PADEXIST PIN extraction method keyword 306
pair of keys 58, 59
PAN data in parameter
encrypted PIN translate callable service 330
PAN data out parameter
encrypted PIN translate callable service 332
PAN data parameter
clear PIN encrypt callable service 312
clear PIN generate alternate callable service 319
encrypted PIN generate callable service 326
encrypted PIN verify callable service 336
parameter
attribute definitions 5
definitions 6
direction 5
exit data 7
exit data length 7
reason code 6
return code 6
type 5
parity of key 87, 189
adjusting 141, 145
EVEN 180
ODD 180
partial notarization 47, 693
calculation for a double-length AKEK 694
calculation for a single-length AKEK 694
PCF
key separation 16
keys 21
macros 6
migration consideration 6
PCI interface callable service (CSFPCI)
parameters 463
syntax 463
performance considerations 10
personal account number (PAN)
for encrypted PIN translate 330
for encrypted PIN verify 336
personal authentication
definition 51
personal identification number (PIN)
3624 PIN generation algorithm 672
3624 PIN verification algorithm 675
algorithm value 320, 337
algorithms 51, 303, 313
block format 304, 328
clear PIN encrypt callable service 52
clear PIN generate alternate callable service 52, 317
definition 51
description 301
detailed algorithms 672
encrypted generation callable service 53
encrypting key 304, 328
extraction rules 671
formats 51
GBP PIN verification algorithm 677
generating 303, 313
remote key distribution 29
  benefits 42
  scenario 41
remote key export
  exporting keys 37
  generating keys 39
remote key export callable service (CSNDRKKX)
  format 182
  overview 26
  parameters 182
  syntax 182
remote key export callable service (CSNDRKKX)
  format 182
  overview 26
  parameters 182
  syntax 182
remote key loading 29
  example 30
  new method 30
remote key-loading
  CCA API changes 35
reserved
random number generate callable service 181
reserved data length parameter
  symmetric MAC generate callable service 293
  symmetric MAC verify callable service 298
reserved data parameter
  Symmetric MAC generate callable service 293
  symmetric MAC verify callable service 298
reserved parameter
  control vector generate callable service 91, 158
  reserved_length
  random number generate callable service 180
retained key delete callable service (CSNDRKD)
  format 426
  overview 74
  parameters 426
  syntax 426
retained key list callable service (CSNDRKL)
  format 429
  overview 74
  parameters 429
  syntax 429
retained private keys
  overview 73
return codes 6, 12
  from PCF macros
  migration consideration 6
returned PVV parameter 321
returned result parameter
  clear PIN encrypt callable service 311
  Rivest-Shamir-Adleman (RSA) algorithm 69
  RKX key token 35
  RKX key-token 600
RSA 1024-bit private internal key token 609, 610
RSA algorithm 69
RSA enciphered key length parameter
  symmetric key generate callable service 199
  symmetric key import callable service 203
RSA enciphered key parameter
  symmetric key generate callable service 199
  symmetric key import callable service 203
RSA private external Chinese remainder theorem key
token 606
RSA private external key token 603
RSA private external modulus-exponent key
token 604, 605
RSA private internal Chinese remainder theorem key
token 611
RSA private internal key token 608
RSA private key identifier 203
RSA private key identifier length 203
RSA public key identifier length parameter
  for symmetric key generate 199
RSA public key identifier parameter 199
RSA public token 602
rule array count parameter
  clear PIN encrypt callable service 311
  Clear PIN encrypt callable service 93, 325
  clear PIN generate alternate callable service 319
  clear PIN generate callable service 314
  control vector translate callable service 94
  decipher callable service 226
  digital signature generate callable service 380
  digital signature verify callable service 387
  diversified key generate callable service 104
  encipher callable service 235
  encrypted PIN translate callable service 330
  encrypted PIN verify callable service 337
  key test callable service 141
  key test extended callable service 145
  key token build callable service 150
MAC generate callable service 273
MAC generation callable service 278
MDC generate callable service 284
one-way hash generate callable service 287
PKA key generate callable service 393, 413
PKA key import callable service 396
PKA key token build callable service 401
PKA public key extract callable service 416
symmetric key export callable service 193
symmetric key generate callable service 197
symmetric key import callable service 202
Symmetric MAC generate callable service 292
Symmetric MAC verify callable service 297
transform CDMF key callable service 183, 207
trusted block create callable service 209
user derived key callable service 213
rule array parameter
  clear PIN encrypt callable service 311
  clear PIN generate alternate callable service 319
  clear PIN generate callable service 314
  control vector translate callable service 94
decipher callable service 226
digital signature generate callable service 380
digital signature verify callable service 387
diversified key generate callable service 104
encipher callable service 235
encrypted PIN generate callable service 325
encrypted PIN translate callable service 330
encrypted PIN verify callable service 337
key test callable service 141
key test extended callable service 145
key token build callable service 150
MAC generate callable service 273
rule array parameter (continued)
  MAC generation callable service 278
  MDC generate callable service 284
  one-way hash generate callable service 287
  PKA key generate callable service 393, 413
  PKA key import callable service 397
  PKA key token build callable service 401
  PKA public key extract callable service 416
  random number generate callable service 180
  symmetric key export callable service 193
  symmetric key generate callable service 197
  symmetric key import callable service 202
  Symmetric MAC generate callable service 292
  symmetric MAC verify callable service 297
  transform CDMF key callable service 183, 207
  trusted block create callable service 209
  user derived key callable service 213
rule_array_count
  ICSF query service callable service 438, 443
rule_id parameter
  remote key export callable service 186
rule_id_length parameter
  remote key export callable service 186
set attribute value callable service (CSFPSAV)
  format 526
  parameters 526
  syntax 526
SET block compose callable service (CSNDSBC) 355
  format 355
  overview 75
  parameters 355
  syntax 355
SET block decompose callable service
  (CSNDSBD) 360
  format 360
  overview 75
  parameters 360
  syntax 360
SET protocol 74
SET Secure Electronic Transaction 74
short blocks 232
shortcut keys 711
signature bit length parameter 382
signature field length parameter
  digital signature generate callable service 382
  digital signature verify callable service 388
signature field parameter
  digital signature generate callable service 382
  digital signature verify callable service 388
single-length key
  multiple decipherment 686
  multiple encipherment 685
  purpose 57, 58
  using 21
source key identifier length parameter
  PKA key import callable service 397
  PKA public key extract callable service 417
source key identifier parameter
  data key export callable service 98
  key export callable service 109
  key import callable service 124
  PKA key import callable service 397
  PKA public key extract callable service 417
  transform CDMF key callable service 207
source key token length parameter
  prohibit export extended callable service 178
source text parameter
  character/nibble conversion callable service 434
  code conversion callable service 436
  X9.9 data editing callable service 460
source_key_length parameter
  remote key export callable service 186
special secure mode 10
SRB, scheduling 10
SSL support 44
sym_encrypted_key_length parameter
  remote key export callable service 187
symmetric algorithm decipher callable service
  (CSNBSAD, CSNBSAD1, CSNESAD and CSNESAD1)
  format 241
  parameters 241
  syntax 241
symmetric algorithm encipher callable service (CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1)
  format 247
  syntax 247

symmetric algorithm encipher callable service
  CSNBSAE, CSNBSAE1, CSNESAE, and CSNESAE1
  parameters 247

symmetric key decipher callable service (CSNBSYD and CSNBSYD1)
  format 253
  parameters 253
  syntax 253

symmetric key encipher callable service (CSNBSYE, CSNBSYE1, CSNESYE, and CSNESYE1)
  format 260
  parameters 260
  syntax 260

symmetric key export callable service (CSNDSYX)
  format 192
  overview 27, 28
  parameters 192
  syntax 192

symmetric key generate callable service (CSNDSYG)
  format 196
  overview 27, 28
  parameters 196
  syntax 196

symmetric key import callable service (CSNDSYI)
  format 201
  overview 27, 29
  parameters 201
  syntax 201

Symmetric MAC generate callable service (CSNBSMG, CSNBSMG1, CSNESMG, and CSNESMG1)
  format 291
  parameters 291
  syntax 291

Symmetric MAC verify callable service (CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1)
  format 295
  parameters 296
  syntax 295

syntax for callable service 3

target text parameter (continued)
  code conversion callable service 436
  ICSF query facility callable service 444
  X9.9 data editing callable service 460

text id in parameter
  ciphertext translate callable service 222
  MAC generate callable service 275
  MAC verify callable service 280
  MDC generate callable service 285
  one-way hash generate callable service 289
  symmetric MAC verify callable service 298

text id out parameter
  ciphertext translate callable service 222
  text in parameter
  ciphertext translate callable service 221

text length parameter
  character/nibble conversion callable service 434
  ciphertext translate callable service 221
  code conversion callable service 436
  cryptographic variable encipher callable service 96
  decipher callable service 226
  encipher callable service 235
  MAC generate callable service 273
  MAC generation callable service 278
  MDC generate callable service 283
  one-way hash generate callable service 288
  Symmetric MAC generate callable service 292
  Symmetric MAC verify callable service 296
  X9.9 data editing callable service 460

text out parameter
  ciphertext translate callable service 222
  text parameter
  ciphertext translate callable service 221
  MAC generate callable service 273
  MAC generation callable service 278
  MDC generate callable service 283
  one-way hash generate callable service 288
  Symmetric MAC generate callable service 292
  Symmetric MAC verify callable service 297

text, translating 219

TKE
  overview 54

token record create callable service (CSFPTRC)
  format 537
  parameters 537
  syntax 537

token record delete callable service (CSFPTRD)
  format 541
  parameters 541
  syntax 541

token record list callable service (CSFPTRL)
  format 543
  parameters 543
  syntax 543

token validation value (TVV) 598

trailing short blocks 232

transaction validation callable service (CSNBSKY)
  format 366
  syntax 366

transaction validation callable service (CSNBTRV) 365

transform CDMF key algorithm 694
transform CDMF key callable service (CSNBTCR)
  format 206
  overview 27
  parameters 206
  syntax 206
transformed shortened DES key 206
transport key 20
transport key variant 16
transport_key_identifier parameter
  remote key export callable service 185
  trusted block create callable service 210
transport_key_length parameter
  remote key export callable service 185
triple-length keys
  multiple encipherment 689
  multiple decipherment 690
trusted block 31
trusted block create
callable service 208
trusted block create callable service (CSNDTBC)
  format 208
  overview 27
  parameters 208
  syntax 208
trusted block key token
  trusted block key token
  trusted block key token 616
trusted blocks
  CCA API changes 35
  creating 36
  exporting keys 37
  using 36
Trusted Key Entry
  overview 54
trusted_block_identifier parameter
  trusted block create callable service 183, 211
trusted_block_length parameter
  remote key export callable service 183
  trusted block create callable service 210
types of keys 19

U
UKPT
  format 309
user derived key
  generating 211
  processing rules 213
utilities
  character/nibble conversion 433
  code conversion 435
  ICSF Query Algorithm 437
  ICSF Query Facility 442
  key token build 147
  PKA key token build 399
  X9.9 data editing 459

V
verification pattern parameter 142, 146
verification pattern, generating and verifying 139, 144
verifying data integrity and authenticity 269
VISA CVV service generate callable service
  (CSNBCCG) 369
  format 369
  parameters 369
  syntax 369
VISA CVV service verify callable service
  (CSNBCCSV) 373
  format 373
  parameters 374
  syntax 373
VISA PVV 313
  generating 317
VISA-1 333
VISA-2 PIN block format 305, 670
VISA-3 PIN block format 305, 670
VISA-4 PIN block format 305
VISA-PVV algorithm 320, 337
VISAPVV4 algorithm 337

X
X9.9 data editing callable service (CSNB9ED)
  format 459
  overview 55
  parameters 459
  syntax 459
X9.9-1 keyword 273, 278
Readers’ Comments — We’d Like to Hear from You

z/OS
Cryptographic Services
Integrated Cryptographic Service Facility
Application Programmer’s Guide

Publication No. SA22-7522-13

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.
Submit your comments using one of these channels:
• Send your comments to the address on the reverse side of this form.
• Send your comments via e-mail to: mhvrdfs@us.ibm.com

If you would like a response from IBM, please fill in the following information:

Name

Address

Company or Organization

Phone No.

E-mail address