

z/OS



Cryptographic Services Integrated Cryptographic Service Facility Writing PKCS #11 Applications

Version 2 Release 1

Note

Before using this information and the product it supports, read the information in "Notices" on page 85.

This edition applies to ICSF FMID HCR77B0 and Version 2 Release 1 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright IBM Corporation 2007, 2015.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	v
--------------------------	----------

Tables	vii
-------------------------	------------

About this document ix

Who should read this document	ix
How this document is organized	ix
How to use this document	ix
Where to find more information	x
IBM Crypto Education	x

How to send your comments to IBM . . . xi

If you have a technical problem.	xi
--	----

Summary of Changes xiii

Changes made in Cryptographic Support for z/OS V1R13 - z/OS V2R1 (FMID HCR77B0)	xiii
Changes made in Cryptographic Support for z/OS V1R13-V2R1 (FMID HCR77A1)	xiii

Chapter 1. Overview of z/OS support for PKCS #11 1

Tokens	1
Secure key PKCS #11	2
The token data set (TKDS).	2
Controlling token access and key policy	3
Managing tokens	7
Sample scenario for setting up z/OS PKCS #11 tokens	8
Sample scenario for controlling clear key processing	9
Auditing PKCS #11 functions	10
Component trace for PKCS #11 functions	10
Object types	11
Session objects	11
Token objects	11
Operating in compliance with FIPS 140-2	11
Requiring signature verification for ICSF module CSFINPV2.	14
Requiring FIPS 140-2 compliance from all z/OS PKCS #11 applications.	16
Requiring FIPS 140-2 compliance from select z/OS PKCS #11 applications.	16
Preparing to use PKCS #11 applications	18
Tasks for the system programmer	18
Tasks for the security administrator	19
Tasks for the auditor	19
Tasks for application programmers	19

Chapter 2. The C API. 21

Using the C API.	21
--------------------------	----

Deleting z/OS PKCS #11 tokens	21
Environment	21
Cross memory considerations	22
Key types and mechanisms supported	23
Objects and attributes supported	30
Library, slot, and token information	47
Functions supported	48
Standard functions supported	48
Non-standard functions supported	55
Non-standard mechanisms supported	55
Enterprise PKCS #11 coprocessors	56
Key algorithms/usages that are unsupported or disallowed by the Enterprise PKCS #11 coprocessors	56
PKCS #11 Coprocessor Access Control Points	58
Standard compliance modes	61
Function return codes	62
Troubleshooting PKCS #11 applications	63

Chapter 3. Sample PKCS #11 C programs 65

Running the pre-compiled version of testpkcs11	66
Steps for running the pre-compiled version of testpkcs11	66
Building sample PKCS #11 applications from source code.	67

Chapter 4. ICSF PKCS #11 callable services 69

Appendix A. SMP/E installation data sets, directories, and files 71

Appendix B. Source code for the testpkcs11 sample program. 73

Appendix C. Accessibility 81

Accessibility features	81
Using assistive technologies	81
Keyboard navigation of the user interface	81
Dotted decimal syntax diagrams	81

Notices 85

Policy for unsupported hardware	86
Minimum supported hardware	87
Trademarks	87

Glossary 89

Index 101

Figures

Tables

1. Token access levels	4	17. RSA private key object attributes that ICSF supports	45
2. Resources in the CSFSERV class for token services	5	18. DSA private key object attributes that ICSF supports	46
3. CLEARKEY.token-label resource access and key security policy	6	19. Diffie-Hellman private key object attributes that ICSF supports	46
4. Mechanism information as returned by C_GetMechanismInfo (CK_MECHANISM_INFO)	23	20. Elliptic Curve private key object attributes that ICSF supports.	46
5. Mechanisms supported by specific cryptographic hardware	28	21. Domain parameter object attributes that ICSF supports	46
6. Restricted algorithms and uses when running in compliance with FIPS 140-2	29	22. DSA domain parameter object attributes that ICSF supports.	47
7. Common footnotes for object attribute tables	30	23. Diffie-Hellman domain parameter object attributes that ICSF supports.	47
8. Data object attributes that ICSF supports	31	24. Standard PKCS #11 functions that ICSF supports	48
9. X.509 certificate object attributes that ICSF supports	32	25. Syntax of the CK_RV CSN_FindALLObjects() function.	55
10. Secret key object attributes that ICSF supports	34	26. Syntax of the CKM_IBM_ATTRIBUTEBOUND_WRAP function.	56
11. Public key object attributes that ICSF supports	38	27. List of algorithms/uses not supported/disallowed by Enterprise PKCS #11 coprocessors	57
12. RSA public key object attributes that ICSF supports	40	28. PKCS #11 Access Control Points.	58
13. DSA public key object attributes that ICSF supports	41	29. Environment variables for capturing trace data	63
14. Diffie-Hellman public key object attributes that ICSF supports.	41		
15. Elliptic Curve public key object attributes that ICSF supports.	41		
16. Private key object attributes that ICSF supports	41		

About this document

This document describes the support for PKCS #11 provided by the z/OS[®] Integrated Cryptographic Service Facility (ICSF). ICSF is a component of z/OS Cryptographic Services, which includes the following components:

- z/OS Integrated Cryptographic Service Facility (ICSF)
- 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.

PKCS #11 is an industry-accepted standard provided by RSA Laboratories of RSA Security Inc. It specifies an application programming interface (API) to devices, referred to as *tokens*, that hold cryptographic information and perform cryptographic functions. PKCS #11 provides an alternative to IBM[®]'s Common Cryptographic Architecture (CCA).

Who should read this document

This document is primarily intended for application programmers who want to write PKCS #11 applications for z/OS. It also contains information for security administrators, system programmers, and auditors in installations that use PKCS #11 applications.

How this document is organized

- Chapter 1, "Overview of z/OS support for PKCS #11," on page 1 provides an overview of ICSF support for PKCS #11. It discusses tokens, the token data set (TKDS), auditing and tracing PKCS #11 functions, session objects, and tasks that must be performed before using PKCS #11 applications.
- Chapter 2, "The C API," on page 21 discusses the PKCS #11 C API provided by ICSF, highlighting differences between the z/OS implementation and the PKCS #11 standard.
- Chapter 3, "Sample PKCS #11 C programs," on page 65 discusses how to build and run the testpkcs11 sample.
- Chapter 4, "ICSF PKCS #11 callable services," on page 69 provides a brief introduction to the PKCS #11 callable services, which are documented in *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

How to use this document

Application programmers should read the entire book.

Security administrators should read the section "Tasks for the security administrator" on page 19 and the information that it references.

System programmers should read the section "Tasks for the system programmer" on page 18 and the information that it references.

Auditors should read the section “Tasks for the auditor” on page 19 and the information that is references.

Where to find more information

Before using this document, application programmers must be familiar with the PKCS #11 specification. The PKCS #11 standard can be found at <http://www.cryptsoft.com/pkcs11doc/>. Application programmers should also be familiar with the ICSF library and C programming.

Security administrators should be familiar with *z/OS Security Server RACF Security Administrator's Guide*.

Auditors should be familiar with *z/OS Security Server RACF Auditor's Guide*.

The callable services for PKCS #11 functions are documented in *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

The format of the token data set is documented in *z/OS Cryptographic Services ICSF System Programmer's Guide*.

IBM Crypto Education

The IBM Crypto Education community provides detailed explanations and samples pertaining to IBM cryptographic technology at <https://www-304.ibm.com/connections/communities/community/crypto>.

How to send your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or provide any other feedback that you have.

Use one of the following methods to send your comments:

1. Send an email to mhvrcfs@us.ibm.com.
2. Send an email from the "Contact us" web page for z/OS (<http://www.ibm.com/systems/z/os/zos/webqs.html>).
3. Mail the comments to the following address:
IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
US
4. Fax the comments to us, as follows:
From the United States and Canada: 1+845+432-9405
From all other countries: Your international access code +1+845+432-9405

Include the following information:

- Your name and address.
- Your email address.
- Your telephone or fax number.
- The publication title and order number:
z/OS ICSF Writing PKCS #11 Applications
SC14-7510-01
- The topic and page number that is related to your comment.
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

Do not use the feedback methods that are listed for sending comments. Instead, take one of the following actions:

- Contact your IBM service representative.
- Call IBM technical support.
- Visit the IBM Support Portal at z/OS support page (<http://www.ibm.com/systems/z/support/>).

Summary of Changes

ICSF is an element of z/OS, but provides independent ICSF releases as web deliverables. These web deliverables are identified by their FMID. Each release of z/OS includes a particular ICSF FMID level as part of its base.

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

Changes made in Cryptographic Support for z/OS V1R13 - z/OS V2R1 (FMID HCR77B0)

This document contains information previously presented in *z/OS ICSF Writing PKCS #11 Applications*, SC14-7510-00.

This document is for ICSF FMID HCR77B0. This release of ICSF runs on z/OS V1R13 and z/OS V2R1 and only on zSeries hardware.

New

- The table, 'Mechanisms supported by specific cryptographic hardware' in "Key types and mechanisms supported" on page 23 has been updated to include new information for IBM z13.

Changed

- "Standard compliance modes" on page 61 is updated.
- "The token data set (TKDS)" on page 2 is updated.

Deleted

No content was removed from this information.

Changes made in Cryptographic Support for z/OS V1R13-V2R1 (FMID HCR77A1)

This document contains information previously presented in *z/OS ICSF Writing PKCS #11 Applications*, SA23-2231-05.

This document is for ICSF FMID HCR77A1. This release of ICSF runs on z/OS V1R13, and z/OS V2R1 and only on zSeries hardware.

New information

- Chapter 2. The C API
 - RSA PSS mechanisms have been added to the table Mechanism information as returned by C_GetMechanismInfo (CK_MECHANISM_INFO).
 - Table Mechanisms supported by specific cryptographic hardware has been updated to include new information for zEC12 and zBC12.
 - CKK_DH private keys in topic Non-standard functions supported have been added to discussion of attribute bound wrapping and key usage flags.

- New sample program, for modifying the compliance mode of all keys in token, has been added to Chapter 3. Sample PKCS #11 C programs.
- Access control points have been added for DH private key use and derivation in table PKCS #11 Access Control Points.
- Standard compliance modes, which have not changed, are now described in section Standard compliance modes.

Changed information

- Chapter 2. The C API
 - DH Key pair generation, DH and DSA parameter generation, and ECDH Derive mechanisms have been updated to indicate that the mechanism can be performed in hardware in table Mechanism information as returned by C_GetMechanismInfo (CK_MECHANISM_INFO).
 - DH keys are now allowed with a 1024 bit restriction in table List of algorithms/uses not supported/disallowed by Enterprise PKCS #11 coprocessors.
 - CKA_IBM_CARD_COMPLIANCE description in topic Objects and attributes supported has been updated to allow modification for Secret, Public, and Private Key Objects.
- Changes to Chapter 3. Sample PKCS #11 C programs:
 - Naming convention of makefiles changed, and location of samples changed.
- Key wrapping and unwrapping now supports the wrapping of symmetric keys with symmetric keys as indicated in table Standard PKCS #11 functions that ICSF Supports.

Deleted information

- Chapter 2. The C API
 - ECC keys have been removed from Table Restricted algorithms and uses when running in compliance with FIPS 140-2.
 - ECC keys have been removed from the table List of algorithms/uses not supported/disallowed by Enterprise PKCS #11 coprocessors.

Chapter 1. Overview of z/OS support for PKCS #11

RSA Laboratories of RSA Security Inc. offers its Public Key Cryptography Standards (PKCS) to developers of computers that use public key and related technology. PKCS #11, also known as Cryptoki, is the cryptographic token interface standard. It specifies an application programming interface (API) to devices, referred to as *tokens*, that hold cryptographic information and perform cryptographic functions. The PKCS #11 API is an industry-accepted standard commonly used by cryptographic applications. ICSF supports PKCS #11, providing an alternative to IBM's Common Cryptographic Architecture (CCA) and broadening the scope of cryptographic applications that can make use of zSeries cryptography. PKCS #11 applications developed for other platforms can be recompiled and run on z/OS.

The PKCS #11 standard can be found at <http://www.cryptsoft.com/pkcs11doc/>. This document describes how ICSF supports that standard. The support includes the following:

- A token data set (TKDS) that serves as a repository for persistent cryptographic keys and certificates used by PKCS #11 applications.
- Instore memory that serves as a repository for temporary (session-only) cryptographic keys and certificates used by PKCS #11 applications.
- A C application programming interface (API) that supports a subset of the V2.20 level of the PKCS #11 specification
- PKCS #11 specific ICSF callable services. The C API uses these callable services.

Tokens

On most single-user systems, a token is a smart card or other plug-installed cryptographic device, accessed through a card reader or *slot*. The PKCS #11 specification assigns numbers to slots, known as *slot IDs*. An application identifies the token that it wants to access by specifying the appropriate slot ID. On systems that have multiple slots, it is the application's responsibility to determine which slot to access.

z/OS must support multiple users, each potentially needing a unique key store. In this multiuser environment, the system does not give users direct access to the cryptographic cards installed as if they were personal smart cards. Instead, z/OS PKCS #11 tokens are virtual, conceptually similar to RACF (SAF) key rings. An application can have one or more z/OS PKCS #11 tokens, depending on its needs.

Typically, PKCS #11 tokens are created in a factory and initialized either before they are installed or upon their first use. In contrast, z/OS PKCS #11 tokens can be created using system software such as RACF, the *gskkyman* utility, or by applications using the C API. Each token has a unique token name, or label, that is specified by the end user or application at the time that the token is created.

Rules: A token name must follow these rules:

- Up to 32 characters in length
- Permitted characters are:
 - Alphanumeric
 - National: @ (X'5B'), # (X'7B'), or \$ (X'7C')

- Period: . (X'4B')
- The first character must be alphabetic or national
- Lowercase letters can be used, but are folded to uppercase
- The IBM1047 code page is assumed

In addition to any tokens your installation may create, ICSF will also create a token that will be available to all applications. This "omnipresent" token is created by ICSF in order to enable PKCS #11 services when no other token has been created. This token supports session objects only. Session objects are objects that do not persist beyond the life of a PKCS #11 session. The omnipresent token is always mapped to slot ID #0, and its token label is SYSTOK-SESSION-ONLY.

Tip: To reference the omnipresent token by label, use the constant `SESS_ONLY_TOK`, which is defined in `csnpdefs.h`.

Because PKCS #11 tokens are typically physical hardware devices, the PKCS #11 specification provides no mechanism to delete tokens. However, because z/OS PKCS #11 tokens are virtual, z/OS must provide a way to delete them. For information on how to delete tokens using the C API, see "Deleting z/OS PKCS #11 tokens" on page 21.

Secure key PKCS #11

z/OS PKCS #11 supports two different keying models, Secure vs Clear. A secure key is one where the sensitive key material is always in wrapped form whenever it is outside the cryptographic device. The key is wrapped using a master key that has been established in the cryptographic device and is not available in its entirety outside that device. A clear key does not have this extra protection. A clear key's sensitive key material will appear in the virtual storage of ICSF in-the-clear and may even appear outside ICSF in-the-clear. Obviously, secure keys provide an extra layer of security. However, clear keys are more versatile than secure keys as they are not bound to any particular cryptographic device. They may even be used via software, when no cryptographic device is available.

The decision on whether to create a clear or secure key happens at the time the key is created. Absent any direction from the applications themselves (through new vendor-defined attributes), ICSF will use the context of the request along with a new RACF profile setting to decide. See "Controlling token access and key policy" on page 3.

z/OS PKCS #11 tokens may contain clear keys or secure keys, or a mixture of both. Secure keys require a secure coprocessor. The Crypto Express4 is an optional hardware feature available on IBM zEnterprise EC12 and IBM zEnterprise BC12. The Crypto Express4 may be configured as a cryptographic accelerator, a secure CCA cryptographic coprocessor, or (new with the IBM zEnterprise EC12 and IBM zEnterprise BC12), a secure PKCS #11 cryptographic coprocessor, also known as an Enterprise PKCS #11 coprocessor or EP11. An Enterprise PKCS #11 coprocessor, with an active master key must be available in order to generate and use secure PKCS #11 keys. Clear keys do not require a coprocessor.

The token data set (TKDS)

The token data set (TKDS) is a VSAM data set that serves as the repository for persistent cryptographic keys and certificates used by PKCS #11 applications. The system programmer creates the TKDS and updates the ICSF installation options data set to identify the data set name of the TKDS.

As of ICSF FMID HCR7770, a TKDS is no longer required in order to run PKCS #11 applications. If ICSF is started without a TKDS, however, only the omnipresent token will be available.

A TKDS is required to utilize Secure Key PKCS #11.

Rules: The token data set must follow these rules:

- It must be a key-sequenced VSAM data set with spanned variable length records.
- It must be allocated on a permanently resident volume.

Clear keys in the token data set are not encrypted. Therefore, it is important that the security administrator create a RACF profile to protect the token data set from unauthorized access.

For the format of the TKDS, see 'Creating the TKDS' in *z/OS Cryptographic Services ICSF System Programmer's Guide*.

To optimize performance, ICSF utilizes in-storage copy of the TKDS.

Controlling token access and key policy

The PKCS #11 standard was designed for systems that grant access to token information based on a PIN. The standard defines two types of users, the standard user (*User*) and the security officer (*SO*), each having its own personal identification number (PIN). The SO can initialize a token (zero the contents) and set the User's PIN. The SO can also access the public objects on the token, but not the private ones. The User has access to the private objects on a token and has the power to change his or her own PIN. The User cannot reinitialize a token. The PIN that a user enters determines which role that user takes. A user can fill both roles by having knowledge of both PINs.

z/OS does not use PINs. Instead, profiles in the SAF CRYPTOZ class control access to tokens. For each token, there are two resources in the CRYPTOZ class for controlling access to tokens:

- The resource *USER.token-name* controls the access of the User role to the token.
- The resource *SO.token-name* controls the access of the SO role to the token.

A user's access level to each of these resources (read, update, or control) determines the user's access level to the token.

There are six possible token access levels. Three are defined by the PKCS #11 standard, and three are unique to z/OS. The PKCS #11 token access levels are:

- User R/O: Allows the user to read the token including its private objects, but the user cannot create new token or session objects or alter existing ones.
- User R/W: Allows the user read/write access to the token object including its private objects.
- SO R/W: Allows the user to act as the security officer for the token and to read, create, and alter public objects on the token.

The token access levels unique to z/OS are:

- Weak SO: A security officer that can modify the CA certificates contained in a token but not initialize the token. (For example, a system administrator who determines the trust policy for all applications on the system.)

- Strong SO: A security officer that can add, generate or remove private objects in a token. (For example, a server administrator.)
- Weak User: A User that cannot change the trusted CAs contained in a token. (For example, to prevent an end-user from changing the trust policy of his or her token.)

Table 1 shows how a user's access level to a token is derived from the user's access level to a resource in the SAF CRYPTOZ class.

Table 1. Token access levels

CRYPTOZ resource	READ (SAF access level)	UPDATE (SAF access level)	CONTROL (SAF access level)
SO.token-label	Weak SO Can read, create, delete, modify, and use public objects	SO R/W Same ability as Weak SO plus can create and delete tokens	Strong SO Same ability as SO R/W plus can read but not use (see Note2) private objects; create, delete, and modify private objects
USER.token-label	User R/O Can read and use (see Note 2) public and private objects	Weak User Same ability as User R/O plus can create, delete, and modify private and public objects. Cannot add, delete, or modify certificate authority objects	User R/W Same ability as Weak User plus can add, delete, and modify certificate authority objects

Note:

1. The USER.token-name and SO.token-name profiles will **not** be checked to determine access to the omnipresent token SYSTOK-SESSION-ONLY. ICSF creates this token to provide PKCS #11 support even if no other token is available to an application. All users will always be considered to have R/W access to this token.
2. "Use" is defined as any of the following:
 - Performing any cryptographic operation involving the key object; for example C_Encrypt
 - Searching for key objects using sensitive search attributes
 - Retrieving sensitive key object attributes.

The sensitive attribute for a secret key is CKA_VALUE. The sensitive attribute for Diffie Hellman, DSA, and Elliptic Curve private key objects is CKA_VALUE. The sensitive attributes for RSA private key objects are CKA_PRIVATE_EXPONENT, CKA_PRIME_1, CKA_PRIME_2, CKA_EXPONENT_1, CKA_EXPONENT_2, and CKA_COEFFICIENT.
3. The CRYPTOZ resources can be defined as "RACF-DELEGATED" if required. For information about delegated resources, see the topic on delegated resources in *z/OS Security Server RACF Security Administrator's Guide*.
4. Although the use of generic profiles in the CRYPTOZ class is permitted, you should not use a single generic profile to cover both the SO.token-label and USER.token-label resources. You should not do this, because there are additional

resources in the class controlling key policy. (See Guidelines in this topic for FIPSEXEMPT.token-label and CLEARKEY.token-label.) Creating a generic profile that uses generic characters to match both the SO and USER portion of the resource name (for example *.token-label) will also inadvertently match these other resources and can have unintended consequences.

5. If the CSFSERV class is active, ICSF performs access control checks on the underlying callable services. The user must have READ access to the appropriate CSFSERV class resource. Table 2 lists the resources in the CSFSERV class for token services.

Table 2. Resources in the CSFSERV class for token services

Name of resource	Service	Called by
CSF1TRC	Token or object creation	C_InitToken, C_CreateObject, C_CopyObject
CSF1TRD	Token or object deletion	C_InitToken, C_DestroyObject
CSF1TRL	Token or object find	C_Initialize, C_FindObjects, CSN_FindALLObjects
CSF1SAV	Set object attributes	C_SetAttributeValue
CSF1GAV	Get object attributes	C_GetAttributeValue
CSF1GSK	Generate secret key	C_GenerateKey
CSF1GKP	Generate key pair	C_GenerateKeyPair
CSF1PKS	Private key sign	C_Decrypt, C_DecryptUpdate, C_DecryptFinal, C_Sign, C_SignFinal
CSF1PKV	Public key verify	C_Encrypt, C_EncryptUpdate, C_EncryptFinal, C_Verify, C_VerifyFinal
CSF1SKD	Secret key decrypt	C_Decrypt, C_DecryptUpdate, C_DecryptFinal
CSF1SKE	Secret key encrypt	C_Encrypt, C_EncryptUpdate, C_EncryptFinal
CSFOWH	One-way hash	C_Digest, C_DigestUpdate, C_DigestFinal, C_Sign, C_SignUpdate, C_SignFinal, C_Verify, C_VerifyUpdate, C_VerifyFinal
CSF1WPK	Wrap key	C_Wrapkey
CSF1UWK	Unwrap key	C_Unwrapkey
CSF1HMG	Generate HMAC	C_Sign
CSF1HMV	Verify HMAC	C_Verify
CSF1DVK	Derive key	C_DeriveKey
CSF1DMK	Derive multiple keys	C_DeriveKey
CSFIQA	PKCS #11 initialization	C_Initialize
CSFRNG	Random number generate	C_GenerateRandom

Guidelines:

1. If your organization controls access to ICSF callable services using the CSFSERV class, define the resources listed in Table 2 on page 5 and grant access accordingly.
Tip: Define generic profiles. For example, a profile named CSF* covers all the ICSF services. A profile named CSF1* covers the PKCS #11 subset of the ICSF services, with the exception of those covered by the CSFOWH, CSFIQF, and CSFRNG resources.
2. The CRYPTOZ class supports generic profiles. Take advantage of this by creating a token naming convention for your organization and enforce it with generic profiles. For example, require users and applications to prefix their token names with their user IDs, as with data set names. (See “Sample scenario for setting up z/OS PKCS #11 tokens” on page 8.)
3. For server applications, grant security officers (server administrators) Strong SO access and their end-users (server daemon user IDs) Weak User or User R/W access.
4. For applications for which you do not wish to separate the security officer and end-user roles, grant the appropriate user IDs access to both the SO and USER profiles.

In addition to these two resources for controlling access to tokens, each token also has two additional resources in the CRYPTOZ class: FIPSEXEMPT.token-name and CLEARKEY.token-name.

The FIPSEXEMPT.token-name resource is used for identifying applications that are subject to FIPS 140 restrictions when ICSF is running in FIPS compatibility mode. Refer to “Operating in compliance with FIPS 140-2” on page 11 for more information.

The CLEARKEY.token-name resource will be queried to determine the policy for creating a clear in contrast to a secure key when CKA_IBM_SECURE=TRUE has not been specified for key generation. The following table indicates the significance of the different access levels. When there is no matching profile defined, the row indicating RACF access of UPDATE or No Decision is considered, the policy is to base the decision on the key’s sensitivity and whether an Enterprise PKCS #11 coprocessor is available or not.¹

1.

Table 3. CLEARKEY.token-label resource access and key security policy

Key Security Objective	RACF ACCESS	Action taken when PKCS #11 coprocessor not available or algorithm not supported	Action taken when PKCS #11 coprocessor available and algorithm supported
Generate no secure keys. Stay compatible with earlier releases	CONTROL	Sensitive – Clear Key Non-sensitive – Clear Key	Sensitive – Clear Key Non-sensitive – Clear Key
Use key sensitivity and environment to determine security	UPDATE or No Decision	Sensitive – Clear Key Non-sensitive – Clear Key	Sensitive – Secure Key Non-sensitive – Clear Key
Ensure keys explicitly marked sensitive are always secure keys	READ	Sensitive – Denied Non-sensitive – Clear Key	Sensitive – Secure Key Non-sensitive – Clear Key
Prevent generation or creation of any clear keys	NONE	Sensitive – Denied Non-sensitive – Denied	Sensitive – Secure Key Non-sensitive – Secure Key

Managing tokens

z/OS provides several facilities to manage tokens:

- A C language application programming interface (API) that implements a subset of the PKCS #11 specification. For a description of this API, see Chapter 2, “The C API,” on page 21.
- PKCS #11 specific ICSF callable services. The C API uses these callable services. For information about these callable services, see Chapter 4, “ICSF PKCS #11 callable services,” on page 69.
- ISPF panels. The ICSF ISPF panels provide the capability to see a formatted view of TKDS objects, and make limited updates to them.
- The RACF RACDCERT command supports the certificate, public key, and private key objects, and provides the following subfunctions to manage these objects:
 - ADDTOKEN - creates a new empty token
 - DELTOKEN - deletes an existing token and everything in it
 - LISTTOKEN - displays information on the certificate objects in a token and whether associated public and private key objects exist
 - BIND - connects a RACF certificate, its public key, and potentially its private key to an existing token
 - UNBIND - removes a certificate and its keys from a token
 - IMPORT - defines a token certificate to RACF

For information about the RACDCERT command, see *z/OS Security Server RACF Command Language Reference* and *z/OS Security Server RACF Security Administrator's Guide*.

- The SAF CRYPTOZ class controls access to tokens. For information about this class, see “Controlling token access and key policy” on page 3.

Service specific notes:

1. For generate key and generate key-pair, CLEARKEY.token-label checking is always performed as described above.
2. For create object, no CLEARKEY.token-label checking is performed. By default, all keys created via create object are clear keys. To get an encrypted key, the caller must specify CKA_IBM_SECURE=TRUE. Such keys are not true secure keys since the sensitive key material has appeared in-the-clear outside the bounds of the secure coprocessor.
3. For copy object, no CLEARKEY.token-label checking is performed. By default, the source key's security is carried forward to the target key. Clear keys may be upgraded to encrypted keys by specifying CKA_IBM_SECURE=TRUE.
4. For unwrap key, the security of the base key always determines the security of the unwrapped key. However, in the case of clear key unwrap, CLEARKEY.token-label checking is performed to see if clear keys are permitted.
5. For derive key, the base key can be clear or secure. The resulting derived key will be clear. CLEARKEY.token-label checking is performed to see if clear keys are permitted.

General notes:

1. You should avoid setting a discrete or generic profile that would restrict clear key creation in the omnipresent token. This token is used by other z/OS components to create session keys only. It is typical for session keys to be clear keys. The clear key resource checked for the omnipresent token is CLEARKEY.SYSTOK-SESSION-ONLY.
2. If no CLEARKEY profile is created to protect a given token, the No Decision row governs the action taken for that token for key creation and generation requests. The action taken depends on whether a PKCS #11 coprocessor is active or not. If no PKCS #11 coprocessor is active, all key creation and generate requests result in clear keys. If a PKCS #11 coprocessor is made active, sensitive keys (CKA_SENSITIVE=TRUE) may be generated as secure keys, depending on the algorithm. This could have an unexpected effect on existing programs.

- The RACF R_Datalib callable service (IRRSDL00) allows applications to read tokens by providing a user ID of *TOKEN* to indicate that the key ring name is really a token name. For information about R_Datalib, see *z/OS Security Server RACF Callable Services*.

Note: IRRSDL00 was originally created to allow applications to read RACF (SAF) key rings, but has been enhanced to read PKCS #11 tokens as well. Thus applications written to read key rings can also read tokens without being modified.

Sample scenario for setting up z/OS PKCS #11 tokens

The following examples show how to control access to z/OS PKCS #11 tokens. In this scenario, a company wants to use z/OS PKCS #11 tokens as the key stores for its FTP and Web servers. The company has established a naming convention for their tokens requiring that all tokens have the owning user ID as the high-level qualifier. The owning user IDs for the FTP and Web server tokens are the daemons FTPSRV and WEBSRV, respectively. User ABIGAIL is the administrator for the servers.

The security administrator, who has the RACF SPECIAL attribute, creates the protection profiles for the tokens. The security administrator's goal is to give user ABIGAIL the Security Officer role for these profiles, and to give the daemon user IDs the User role. To do this, the security administrator issues RACF TSO commands. First, the security administrator activates the CRYPTOZ class with generics and RACLISTs it:

```
SETROPTS CLASSACT(CRYPTOZ) GENERIC(CRYPTOZ) RACLIST(CRYPTOZ)
```

Next, the security administrator creates profiles for the security officer's access to the FTP and Web Server tokens:

```
RDEFINE CRYPTOZ SO.FTPSRV.* UACC(NONE)
RDEFINE CRYPTOZ SO.WEBSRV.* UACC(NONE)
```

Then, the security administrator creates profiles for the standard user's access to the FTP and Web Server tokens:

```
RDEFINE CRYPTOZ USER.FTPSRV.* UACC(NONE)
RDEFINE CRYPTOZ USER.WEBSRV.* UACC(NONE)
```

The security administrator now gives user ABIGAIL Strong SO power for the tokens by giving her CONTROL access to the profiles that protect the tokens. The Strong SO power does not allow ABIGAIL to use the private objects in the tokens:

```
PERMIT SO.FTPSRV.* CLASS(CRYPTOZ) ID(ABIGAIL) ACC(CONTROL)
PERMIT SO.WEBSRV.* CLASS(CRYPTOZ) ID(ABIGAIL) ACC(CONTROL)
```

Next, the security administrator gives the users FTPSRV and WEBSRV Weak User power for their respective tokens. This power allows them to use the private objects within the tokens, but not change the set of trusted CA certificates.

```
PERMIT USER.FTPSRV.* CLASS(CRYPTOZ) ID(FTPSRV) ACC(UPDATE)
PERMIT USER.WEBSRV.* CLASS(CRYPTOZ) ID(WEBSRV) ACC(UPDATE)
```

Finally, the security administrator refreshes the in-storage profiles for the CRYPTOZ class, so that the changes he just made take effect:

```
SETROPTS RACLIST(CRYPTOZ) REFRESH
```

Now the set up is complete: ABIGAIL has Strong SO power over the tokens for the FTP server and the Web server, and can create the required tokens. FTPSRV and WEBSRV have User power over their respective tokens, and can use them as key stores after ABIGAIL has created them.

The task now is to create and populate the tokens for the servers with RACF certificates. The following certificates exist:

1. A root CA certificate installed under CERTAUTH with label 'Local Root CA for Servers'.
2. An end-entity certificate and private key installed under user FTPSRV with label 'FTP Key'. This certificate was signed by the first certificate.
3. An end-entity certificate and private key installed under user WEBSRV with label 'Web Key'. This certificate was also signed by the first certificate.

User ABIGAIL issues the following TSO commands to create the tokens, using the company's naming conventions:

```
RACDCERT ADDTOKEN(ftpsrv.ftp.server.pkcs11.token)
RACDCERT ADDTOKEN(websrv.web.server.pkcs11.token)
```

Next, issue the commands that bind the root CA certificate to the two tokens:

```
RACDCERT BIND(CERTAUTH LABEL('Local Root CA for Servers')
TOKEN(ftpsrv.ftp.server.pkcs11.token)
RACDCERT BIND(CERTAUTH LABEL('Local Root CA for Servers')
TOKEN(websrv.web.server.pkcs11.token)
```

Now, bind the end-entity certificates to their respective tokens. Each should be the default in the token.

```
RACDCERT BIND(ID(FTPSRV) LABEL("FTP key")
TOKEN(ftpsrv.ftp.server.pkcs11.token) DEFAULT)
RACDCERT BIND(ID(WEBSRV) LABEL("Web key")
TOKEN(websrv.web.server.pkcs11.token) DEFAULT)
```

The final step is for the user (ABIGAIL) to configure both servers to use their respective tokens: add directives to the servers' configuration files.

For the web server (IBM HTTP Server), the keyfile directive in the httpd.conf file is set as follows:

```
keyfile *TOKEN*/WEBSRV.WEB.SERVER.PKCS11.TOKEN SAF
```

The SAF keyword indicates to SSL that this is a key ring and is controlled by SAF; it is not a KDB file. The TOKEN keyword indicates that the key ring is a token. The FTP server configuration file also requires a token-qualified key ring name:

```
keyfile *TOKEN*/FTPSRV.FTP.SERVER.PKCS11.TOKEN
```

Sample scenario for controlling clear key processing

The following examples show how the RACF administrator will use the new CRYPTOZ resource, **CLEARKEY**,*token-label*, to set policy on the use of clear keys.

In this scenario, company XYZ wishes to use the ABC program. The ABC program will be creating session keys and be using the system level token named 'SYSTOK-SESSION-ONLY' for cryptographic operations. Company XYZ wants to ensure that all keys created by the ABC program are secure keys. The user ID assigned to the ABC program is ABCUSER.

Company XYZ also has other applications that use the system level token for cryptographic operations. These applications should not be restricted to using only secure keys.

User RACFADM, who has the RACF SPECIAL creates the profiles necessary by issuing the following RACF TSO commands:

1. Activate the CRYPTOZ class with generics and RACLIST it:
SETROPTS CLASSACT(CRYPTOZ) GENERIC(CRYPTOZ) RACLIST(CRYPTOZ)
2. Create the CLEARKEY profile for the system level token:
RDEF CRYPTOZ CLEARKEY.SYSTOK-SESSION-ONLY UACC(NONE)
3. Restrict user ID ABCUSER to secure keys only:
PERMIT CLEARKEY.SYSTOK-SESSION-ONLY CLASS(CRYPTOZ) ID(ABCUSER) ACC(NONE)
4. Allow all other user IDs to create clear keys – normal mode:
PERMIT CLEARKEY.SYSTOK-SESSION-ONLY CLASS(CRYPTOZ) ID(*) ACC(UPDATE)
5. Refresh the RACLIST to pick up the changes:
SETROPTS RACLIST(CRYPTOZ) REFRESH

Auditing PKCS #11 functions

PKCS #11 functions are audited in the following ways:

- The SMF type 82 subtype 1 record that is written during ICSF initialization contains the data set name of the token data set (TKDS).
- The SMF type 82 subtype 21 record that is written when a member joins or leaves a sysplex group contains the cryptographic keys data set (CKDS) data set name if the member joined or left the ICSF CKDS sysplex group, or the TKDS data set name if the member joined or left the ICSF TKDS sysplex group.
- ICSF writes SMF type 82 subtype 23 records whenever a TKDS record for a token or token object is created, modified, or deleted. ICSF does not write SMF records for changes to session objects.

For descriptions of the SMF records that ICSF writes, see *z/OS MVS System Management Facilities (SMF)*.

Component trace for PKCS #11 functions

The following ICSF component trace entries trace events related to the token data set (TKDS):

- Type 16 (XCFTMSGGS) traces the broadcast of an XCF message related to TKDS I/O.
- Type 17 (XCFTMSGGR) traces the receipt of an XCF message related to TKDS I/O.
- Type 18 (XCFTENQ) traces the return of control to the TKDS I/O subtask following the request for an exclusive ENQ on the SYSZTKDS.TKDS dsn resource.

These trace entry types are always traced.

When viewed via IPCS, these entries show the ASCB address, the TCB address, the ASID, the general purpose registers, the GPR length, and the CSS address. For more information about IPCS, see *z/OS MVS IPCS User's Guide*.

Object types

ICSF supports PKCS #11 session objects and token objects. The following classes of objects can be associated with these object types:

- Certificate
- Public key
- Private key
- Secret key
- Data objects
- Domain parameters

Session objects

A session object exists for the life of a PKCS #11 session. ICSF allocates session object memory areas to hold session objects; they are not maintained on DASD. ICSF associates a session object memory area with the application that requested the creation of a session object. There is only one session object memory area for an application, even if the application spawns multiple PKCS #11 sessions. The same session objects are available to all PKCS #11 sessions within an application.

ICSF creates a session object memory area the first time a session object is created, if there is currently no session object memory area associated with the application. The session object memory area exists as long as the PKCS #11 application's address space and job step TCB exist. ICSF deletes the memory area if either the address space or job step TCB terminates. If ICSF terminates, all session object memory areas are destroyed.

ICSF creates one session-object token, the omnipresent token, to provide PKCS #11 support even if no other token is available to an application. For example, no other token is available when a TKDS is not identified using the TKDSN option in the ICSF installation options data set, or when the SAF CRYPTOZ class has not been activated. This session object token (labeled SYSTOK-SESSION-ONLY) is write protected, cannot be used to store persistent attributes, and cannot be deleted.

On z/OS, an application can be running in either single address space mode, or in cross memory mode. The PKCS #11 standard has no concept of cross memory mode, so there is no predefined expected behavior for a PKCS #11 application running in cross memory mode. If running in cross memory mode, you should be aware of the guidelines pertaining to session objects described in "Cross memory considerations" on page 22.

Token objects

Token objects are stored in the token data set, with one record per object. They are visible to all applications that have sufficient permission to the token. They are persistent: they remain associated with the token even after a session is closed.

Operating in compliance with FIPS 140-2

The National Institute of Standards and Technology (NIST) is the US federal technology agency that works with industry to develop and apply technology, measurements, and standards. One of the standards published by NIST is the Federal Information Processing Standard Security Requirements for Cryptographic Modules, referred to as FIPS 140-2. FIPS 140-2 provides a standard that can be required by organizations who specify that cryptographic-based security systems are to be used to provide protection for sensitive or valuable data.

z/OS PKCS #11 cryptography is designed to meet FIPS 140-2 Level 1 criteria, and can be configured to operate in compliance with FIPS 140-2 specifications. Applications that need to comply with the FIPS 140-2 standard can therefore use the z/OS PKCS #11 services in a way that allows only the cryptographic algorithms (including key sizes) approved by the standard and restricts access to the algorithms that are not approved. There are three modes of FIPS operation:

- The services can be configured so that all z/OS PKCS #11 applications are forced to comply with the FIPS 140-2 standard. This is called *FIPS standard mode*.
- For installations where only certain z/OS PKCS #11 applications need to comply with the FIPS 140-2 standard, the services can be configured so that only the necessary applications are restricted from using the non-approved algorithms and key sizes, while other applications are not. This is called *FIPS compatibility mode*. You can also use FIPS compatibility mode to test individual applications to ensure FIPS compliance before switching to FIPS standard mode.
- In *FIPS no enforcement mode*, ICSF will not impose FIPS algorithm or key size restrictions unless the calling application explicitly requests it.

ICSF installation options are described in the *z/OS Cryptographic Services ICSF System Programmer's Guide*. The installation option FIPSMODE indicates one of the following:

FIPSMODE(YES, FAIL(*fail-option*))

Indicates that ICSF is to operate in FIPS standard mode - all applications that call ICSF PKCS #11 services will have a need to run in FIPS 140-2 compliant fashion. Therefore, ICSF is to honor FIPS 140 restrictions pertaining to PKCS #11 algorithms and keys for all applications that call ICSF PKCS #11 services.

ICSF initialization will test that it is running on an IBM z Systems model type and version/release of z/OS that supports FIPS. If so, ICSF initialization will also perform a series of cryptographic known answer self tests. Should a test fail, the action ICSF initialization takes is dependent on the fail option:

FIPSMODE(YES, FAIL(YES))

Indicates ICSF is to terminate abnormally if there is a failure in any of the tests performed.

FIPSMODE(YES, FAIL(NO))

Indicates ICSF initialization processing is to continue even if there is a failure in any of the tests performed. However, PKCS #11 support will be limited or nonexistent depending on the test that failed:

- If ICSF is running on an IBM z Systems model type or with a version/release of z/OS that does not support FIPS, most FIPS processing is bypassed. PKCS #11 callable services will be available, but ICSF will not adhere to FIPS 140 restrictions. Requests to generate or use a key with CKA_IBM_FIPS140=TRUE or those requests that explicitly ask for FIPS processing will result in a failure return code.
- If a known answer test failed, all ICSF PKCS #11 callable services will be unavailable.

FIPSMODE(COMPAT, FAIL(*fail-option*))

Indicates that ICSF is to operate in FIPS compatibility mode - some selected applications that call ICSF PKCS #11 services will have a need to run in FIPS 140-2 compliant fashion while others may not be so restricted.

ICSF initialization will test that it is running on an IBM z Systems model type and version/release of z/OS that supports FIPS. If so, ICSF initialization will also perform a series of cryptographic known answer self tests. Should a test fail, the action ICSF initialization takes is dependent on the fail option:

FIPSMODE(COMPAT, FAIL(YES))

Indicates ICSF is to terminate abnormally if there is a failure in any of the tests performed.

FIPSMODE(COMPAT, FAIL(NO))

Indicates ICSF initialization processing is to continue even if there is a failure in any of the tests performed. However, PKCS #11 support will be limited or nonexistent depending on the test that failed:

- If ICSF is running on an IBM z Systems model type or with a version/release of z/OS that does not support FIPS, most FIPS processing is bypassed. PKCS #11 callable services will be available, but ICSF will not adhere to FIPS 140 restrictions. Requests to generate or use a key with CKA_IBM_FIPS140=TRUE or those requests that explicitly ask for FIPS processing will result in a failure return code.
- If a known answer test failed, all ICSF PKCS #11 callable services will be unavailable

Assuming all tests are successful, ICSF will honor FIPS 140 restrictions pertaining to PKCS #11 algorithms and keys for selected applications that call ICSF PKCS #11 services. This selection process is enabled by the following function:

- New CRYPTOZ Class resource profiles will allow the customer to control the scope of FIPS 140-2 compliance at the token level. The naming convention for these resources is:

FIPSEXEMPT.token-name

The levels of access are as follows:

- Profile not defined or Profile defined, caller has access NONE – User not exempt. Algorithms and key sizes restricted as per FIPS 140-2. For example: usage is treated as if ICSF was started with FIPSMODE(YES, FAIL(*fail-option*)).
- Profile defined, caller has access READ – User's use of the token is exempt from the FIPS 140-2 algorithm restrictions.

A new vendor defined Boolean key attribute is now supported, CKA_IBM_FIPS140. Applications may explicitly set this at the time the key is created. The default value is FALSE. If set to TRUE, ICSF will ensure that the key is only used in a FIPS 140-2 compliant fashion, and treated as if FIPSEXEMPT.token-name access NONE was specified.

Note: If CKA_IBM_FIPS140 is specified as a key generation attribute, this would include the generation of the key as well.

FIPSMODE(NO,FAIL(*fail-option*))

Indicates that ICSF should operate in FIPS no enforcement mode, also known as FIPS on-demand mode. Applications may request strict adherence to FIPS 140 restrictions when requesting ICSF services. However, applications not requesting FIPS processing are not required to adhere to

FIPS 140 restrictions. FIPSEXEMPT.token-name profiles, if they exist, will not be examined. If ICSF is running on an IBM z Systems model type that does not support FIPS, requests to generate or use a key with CKA_IBM_FIPS140=TRUE or those requests that explicitly ask for FIPS processing will result in a failure return code.

ICSF initialization will test that it is running on an IBM z Systems model type and version/release of z/OS that supports FIPS. If so, ICSF initialization will also perform a series of cryptographic known answer self tests. Should a test fail, the action ICSF initialization takes is dependent on the fail option:

FIPSMODE(NO, FAIL(YES))

Indicates ICSF is to terminate abnormally if there is a failure in any of the tests performed.

FIPSMODE(NO, FAIL(NO))

Indicates ICSF initialization processing is to continue even if there is a failure in any of the tests performed. However, PKCS #11 support will be limited or nonexistent depending on the test that failed:

- If ICSF is running on an IBM z Systems model type or with a version/release of z/OS that does not support FIPS, most FIPS processing is bypassed. PKCS #11 callable services will be available, but ICSF will not adhere to FIPS 140 restrictions. Requests to generate or use a key with CKA_IBM_FIPS140=TRUE or explicitly ask for FIPS processing will result in a failure return code.
- If a known answer test failed, all ICSF PKCS #11 callable services will be unavailable.

If the FIPSMODE option is not specified, FIPSMODE(NO, FAIL(NO)) is the default.

If any z/OS PKCS #11 application intends to use the services in compliance with the FIPS 140-2 standard, then, in accordance with that standard, the integrity of the load module containing the z/OS PKCS #11 services must be checked when ICSF is started. This load module is digitally signed, and, in order for applications using its services to be FIPS 140-2 compliant, the signature must be verified when ICSF is started. For more information, refer to “Requiring signature verification for ICSF module CSFINPV2.”

If any application will use PKCS #11 objects for AES Galois/Counter Mode (GCM) encryption or GMAC generation, and will have ICSF generate the initialization vectors, then you need to set ECVTSPLX or CVTSNAME to a unique value. Refer to *z/OS Cryptographic Services ICSF System Programmer's Guide* for more information.

Requiring signature verification for ICSF module CSFINPV2

If your installation needs to operate z/OS PKCS #11 in compliance with the FIPS 140-2 standard, then the integrity of the cryptographic functions shipped by IBM must be verified at your installation during ICSF startup. The load module that contains the software cryptographic functions is SYS1.SIEALNKE(CSFINPV2), and this load module is digitally signed when it is shipped from IBM. Using RACF, you can verify that the module has remained unchanged from the time it was built and installed on your system. To do this, you create a profile in the PROGRAM

class for the CSFINPV2 module, and use this profile to indicate that signature verification is required before the module can be loaded.

To require signature verification for ICSF module CSFINPV2:

1. Make sure that RACF has been prepared to verify signed programs. As described in *z/OS Security Server RACF Security Administrator's Guide*, a security administrator prepares RACF to verify signed programs by creating a key ring for signature verification, and adding the code-signing CA certificate that is supplied with RACF to the key ring. If RACF has been prepared to verify signed programs, there will be a key ring dedicated to signature verification, the code-signing CA certificate will be attached to the key ring, and the PROGRAM class will be active.

- a. If RACF has been prepared to verify signed programs, the discrete profile IRR.PROGRAM.SIGNATURE.VERIFICATION in the FACILITY class will specify the name of the signature-verification key ring. To determine if a signature key ring is already active, enter the command:

```
RLIST FACILITY IRR.PROGRAM.SIGNATURE.VERIFICATION
```

If there is no discrete profile with this name, have your security administrator prepare RACF to verify signed programs using the information in *z/OS Security Server RACF Security Administrator's Guide*.

- b. If the signature verification key ring exists, the RLIST command will display information for the discrete profile

```
IRR.PROGRAM.SIGNATURE.VERIFICATION in the FACILITY class. The name of the signature verification key ring and the name of the key ring owner will be included in the APPLICATION DATA field of the RLIST command output. Using this information, enter the RACDCERT LISTRING command to make sure the code-signing CA certificate is attached to the key ring:
```

```
RACDCERT ID(key-ring-owner) LISTRING(key-ring-name)
```

The label of the code-signing CA certificate is 'STG Code Signing CA'. If this label is not shown in the RACDCERT LISTRING command output, have your security administrator prepare RACF to verify signed programs using the information in *z/OS Security Server RACF Security Administrator's Guide*.

- c. Program control must be active in order for RACF to perform signature verification processing. To make sure the PROGRAM class is active, enter the SETROPTS LIST command.

```
SETROPTS LIST
```

The ACTIVE CLASSES field of the command output should include the PROGRAM class. If it does not, have your security administrator prepare RACF to verify signed programs using the information in *z/OS Security Server RACF Security Administrator's Guide*.

2. Create a profile for the CSFINPV2 program module in the PROGRAM class, indicating that the program must be signed. The following command specifies that the program should fail to load if the signature cannot be verified for any reason. This command also specifies that all signature verification failures should be logged.

Note: Due to space constraints, this command example appears on two lines. However, the RDEFINE command should be entered completely on one line.

```
RDEFINE PROGRAM CSFINPV2 ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))
```

You will need to activate your profile changes in the PROGRAM class.
SETRPTS WHEN(PROGRAM) REFRESH

Requiring FIPS 140-2 compliance from all z/OS PKCS #11 applications

If all z/OS PKCS #11 applications running on your system must comply with the FIPS 140-2 standard, your installation's system programmer should configure ICSF so that z/OS PKCS #11 operates in FIPS standard mode. To do this:

1. Make sure the integrity of the cryptographic functions shipped by IBM in the ICSF module CSFINPV2 will be verified by RACF before the module is loaded. This is done by following the instructions in "Requiring signature verification for ICSF module CSFINPV2" on page 14. If these steps are not followed to verify the digital signature of the module, no application calling the z/OS PKCS #11 services can be considered FIPS 140-2 compliant.
2. To specify FIPS standard mode, have your installation's system programmer include the installation option FIPSMODE(YES, FAIL(*fail-option*)) in the ICSF installation options data set.

When this option is used, ICSF will operate in FIPS standard mode. In this mode, ICSF initialization will test that it is running on an IBM z Systems model type and a version and release of z/OS that supports FIPS. If so, then ICSF will perform a series of cryptographic known answer tests as required by the FIPS 140-2 standard. If the tests succeed, then all applications calling z/OS PKCS services will be restricted from using the PKCS #11 algorithms and key sizes that are prohibited by the FIPS 140-2 standard (as outlined in Table 6 on page 29).

If any of the installation tests should fail, the action ICSF initialization takes depends on the *fail-option* specified. The *fail-option* within the FIPSMODE(YES, FAIL(*fail-option*)) installation option can be either:

- YES (which indicates that ICSF should terminate abnormally if there is a failure in any of the tests that are performed).
- NO (which indicates that ICSF initialization processing should continue even if there is a failure in one or more of the tests that are performed). If an initialization test does fail, however, PKCS #11 support will be limited or nonexistent depending on the test that failed.
 - If ICSF is running on an IBM z Systems model type or with a version of z/OS that does not support FIPS, most FIPS processing is bypassed. PKCS #11 callable services will be available, but ICSF will not adhere to FIPS 140 restrictions. Requests to generate or use a key with CKA_IBM_FIPS140=TRUE will result in a failure return code.
 - If a known answer test failed, all ICSF PKCS #11 callable services will be unavailable.

For more information on this and other ICSF installation options, refer to *z/OS Cryptographic Services ICSF System Programmer's Guide*.

Requiring FIPS 140-2 compliance from select z/OS PKCS #11 applications

If only certain z/OS PKCS #11 applications running on your system must comply with the FIPS 140-2 standard, while other z/OS PKCS #11 applications do not, your system programmer should configure ICSF so that z/OS PKCS #11 operates in FIPS compatibility mode. In FIPS compatibility mode, you can use resource profiles in the CRYPTOZ class to specify, at a token level, the applications that are

exempt from FIPS 140-2 compliance and, for that reason, should not be subject to FIPS restrictions. To configure the z/OS PKCS #11 services to operate in FIPS compatibility mode:

1. Make sure the integrity of the cryptographic functions shipped by IBM in the module ICSF module CSFINPV2 will be verified by RACF before the module is loaded. This is done by following the instructions in “Requiring signature verification for ICSF module CSFINPV2” on page 14. If these steps are not followed to verify the digital signature of the module, no application calling the z/OS PKCS #11 services can be considered FIPS 140-2 compliant.
2. To specify FIPS compatibility mode, have your installation's system programmer include the installation option FIPSMODE(COMPAT, FAIL(*fail-option*)) in the ICSF installation options data set.

When this option is used, ICSF will operate in FIPS compatibility mode. In this mode, ICSF initialization will test that it is running on an IBM z Systems model type, and a version and release of z/OS, that supports FIPS. If so, then ICSF will perform a series of cryptographic known answer tests as required by the FIPS 140-2 standard. If the tests are successful, then, by default, all applications calling z/OS PKCS services will be restricted from using the PKCS #11 algorithms and key sizes that are prohibited by the FIPS 140-2 standard (as outlined in Table 6 on page 29). Using profiles in the CRYPTOZ class, however, you can identify applications that are exempt from FIPS 140-2 compliance (as described in the next step).

If any of the installation tests should fail, the action ICSF initialization takes depends on the *fail-option* specified. The *fail-option* within the FIPSMODE(COMPAT, FAIL(*fail-option*)) installation option can be either:

- YES (which indicates that ICSF should terminate abnormally if there is a failure in any of the tests that are performed).
- NO (which indicates that ICSF initialization processing should continue even if there is a failure in one or more of the tests that are performed). If an initialization test does fail, however, PKCS #11 support will be limited or nonexistent depending on the test that failed.
 - If ICSF is running on an IBM z Systems model type or with a version of z/OS that does not support FIPS, most FIPS processing is bypassed. PKCS #11 callable services will be available, but ICSF will not adhere to FIPS 140 restrictions. Requests to generate or use a key with CKA_IBM_FIPS140=TRUE will result in a failure return code.
 - If a known answer test failed, all ICSF PKCS #11 callable services will be unavailable.

For more information on this and other ICSF installation options, refer to z/OS *Cryptographic Services ICSF System Programmer's Guide*.

3. To specify which applications must comply with FIPS 140-2 restrictions and which applications do not need to comply, create FIPSEXEMPT.*token-label* resource profiles in the CRYPTOZ class. If no FIPSEXEMPT.*token-label* resource profiles are created, then all z/OS PKCS #11 applications will be subject to FIPS restrictions. By creating a FIPSEXEMPT.*token-label* resource profile for a particular token, however, you can specify whether or not a particular user ID should be considered exempt from FIPS restrictions when using that token.
 - If a user ID has access authority NONE to the FIPSEXEMPT.*token-label* resource, ICSF will enforce FIPS 140-2 compliance for that user ID.
 - If a user ID has access authority READ to the FIPSEXEMPT.*token-label* resource, that user ID is exempt from FIPS 140-2 restrictions.

To specify which applications must comply with the FIPS 140-2 restrictions, and which do not, the security administrator must:

- a. If it is not already activated, activate the CRYPTOZ class with generics and RACLIST it:

```
SETROPTS CLASSACT(CRYPTOZ) GENERIC(CRYPTOZ) RACLIST(CRYPTOZ)
```

- b. Create the FIPSEXEMPT.*token-label* resource profile for each z/OS PKCS #11 token. The following command creates the profile for the omnipresent session-object token SYSTOK-SESSION-ONLY.

```
RDEF CRYPTOZ FIPSEXEMPT.SYSTOK-SESSION-ONLY UACC(NONE)
```

Although the use of generic profiles in the CRYPTOZ class is permitted, you should begin the profile name with "FIPSEXEMPT". Failure to do this could result in generic characters unintentionally matching the SO.*token-label* or USER.*token-label* resources for token access, and so could have unintended consequences.

- c. Using the PERMIT command, specify READ access authority for user IDs that are exempt from FIPS 140-2 restrictions, and NONE access authority for user IDs that must comply with FIPS 140-2. The following command indicates that all user IDs are exempt, except for the daemon user ID BOGD.

```
PERMIT FIPSEXEMPT.SYSTOK-SESSION-ONLY CLASS(CRYPTOZ) ID(*) ACC(READ)  
PERMIT FIPSEXEMPT.SYSTOK-SESSION-ONLY CLASS(CRYPTOZ) ID(BOGD) ACC(NONE)
```

- d. Refresh the CRYPTOZ class in common storage:

```
SETROPTS RACLIST(CRYPTOZ) REFRESH
```

Specifying FIPS 140-2 compliance from within a z/OS PKCS #11 application

When running in FIPS compatibility mode or FIPS no enforcement mode, a PKCS #11 application can, when creating a key, specify that generation and subsequent use of the key must adhere to FIPS 140-2 restrictions. An application specifies this by setting the Boolean attribute CKA_IBM_FIPS140 to TRUE when creating the key. If an application does this, the FIPS 140-2 restrictions (as outlined in Table 6 on page 29) will be enforced for the key regardless of any specifications made at the token level using FIPSEXEMPT.*token-label* resource profiles.

If the FIPSMODE(YES, FAIL(*fail-option*)) installation option is specified, indicating FIPS 140-2 compliance is required by all applications, setting the Boolean attribute CKA_IBM_FIPS140 to TRUE is merely redundant and does not result in an error.

Preparing to use PKCS #11 applications

Before an installation can use PKCS #11 applications, some preparation is required on the part of the system programmer, security administrator, auditor, and application programmers. This topic describes the preparation required.

Tasks for the system programmer

If persistent PKCS #11 tokens and objects are needed, the system programmer allocates a token data set (TKDS) for use by PKCS #11 functions, and specifies the data set name of the TKDS in the TKDSN option of the ICSF installation options data set.

The system programmer must decide whether or not sysplex-wide consistency of the TKDS is required, and must specify the SYSPLEXTKDS option in the ICSF installation options data set to define the processing of TKDS updates in a sysplex environment.

If any application must comply with the FIPS 140-2 standard, the system programmer must configure ICSF to run PKCS #11 services in compliance with FIPS 140-2. To do this, the system administrator uses the FIPSMODE option to specify either FIPS standard mode or FIPS compatibility mode as required. For more information on the FIPSMODE option, refer to “Operating in compliance with FIPS 140-2” on page 11, and the *z/OS Cryptographic Services ICSF System Programmer’s Guide*.

The system programmer should run the testpkcs11 utility program to test the PKCS #11 configuration. For information about running the testpkcs11 program, see “Running the pre-compiled version of testpkcs11” on page 66.

Tasks for the security administrator

The security administrator should create a RACF profile to protect the data set that contains the token data set. It is important to protect this data set, because keys in the token data set are not encrypted.

The security administrator needs to grant the appropriate access authority to users for accessing tokens and objects, by defining profiles in the CRYPTOZ class. For more information, see “Controlling token access and key policy” on page 3.

The security administrator controls access to the PKCS #11 callable services by defining profiles in the CSFSERV class. For information about defining profiles in the CSFSERV class, see *z/OS Cryptographic Services ICSF Administrator’s Guide*. For a list of the resource names for token services, see Table 2 on page 5.

If PKCS #11 services must run in compliance with the FIPS 140-2 standard, then the security administrator must ensure that the digital signature of the load module containing the z/OS PKCS #11 services is verified when ICSF starts. This must be done to satisfy FIPS 140-2 requirements. Refer to “Requiring signature verification for ICSF module CSFINPV2” on page 14, and *z/OS Security Server RACF Security Administrator’s Guide* for more information.

To use Secure Key PKCS #11, an active Enterprise PKCS #11 coprocessor is required. For the steps necessary to activate the Enterprise PKCS #11 coprocessors, see ‘Managing Crypto Express4 Features on an IBM zEnterprise BC12 and EC12’ in *z/OS Cryptographic Services ICSF Administrator’s Guide*.

Tasks for the auditor

Auditors should become familiar with the data in SMF records that is related to PKCS #11 functions. For more information, see “Auditing PKCS #11 functions” on page 10.

Tasks for application programmers

Application programmers can write applications using the PKCS #11 API provided by ICSF. They should become familiar with the PKCS #11 specification, and with the information in this book. The PKCS #11 standard can be found at <http://www.cryptsoft.com/pkcs11doc/>.

Application programmers can use the sample program, `testpkcs11`, to learn about building and running PKCS #11 applications, and to troubleshoot problems. For information about the sample program, see Chapter 3, “Sample PKCS #11 C programs,” on page 65.

Chapter 2. The C API

ICSF provides a PKCS #11 C language application program interface (API). This topic highlights the differences between the z/OS API and the PKCS #11 V2.20 specification. To use this API, you must be familiar with both the PKCS #11 specification and the information in this topic.

All manifest constants specified in this chapter can be found in the `csnpdefs.h` include file and (with the exceptions noted) in the PKCS #11 specification.

Using the C API

To create or use a z/OS PKCS #11 token, an application needs to do the following:

1. Implicitly or explicitly load the PKCS #11 API DLL (CSNPCAPI for applications running in 31-bit addressing mode not using XPLINK, CSNPCA3X for applications running in 31-bit addressing mode using XPLINK, CSNPCA64 for 64-bit addressing mode).
2. Locate the functions within that DLL, using the `C_GetFunctionList` function.
3. Call `C_Initialize`, which enables the application to call other functions in the API.
4. Determine the slots present, using the `C_GetSlotList` function. This function returns a slot number for each existing token to which the application has access.
5. To use an existing token, the application iterates through the slots using `C_GetTokenInfo` to find the token wanted.

To create a new token, the application uses the `C_WaitForSlotEvent` function to add a new slot containing an uninitialized token. The application then uses the `C_InitToken` function to initialize the new token and save it in the TKDS.

Deleting z/OS PKCS #11 tokens

Because PKCS #11 tokens are typically physical hardware devices, the PKCS #11 specification provides no mechanism to delete tokens. However, for z/OS PKCS #11 tokens, which are virtual, there must be a capability to delete tokens. An application does this by calling the `C_InitToken` function with a special label value `$$DELETE-TOKEN$$` (assuming code page IBM1047), left justified and padded on the right to 32 characters.

Tip: Use the constant `DEL_TOK` defined in `csnpdefs.h`.

You cannot delete the omnipresent token `SYSTOK-SESSION-ONLY` (created by ICSF to provide PKCS #11 support even if no other token is available to an application). If an application attempts to delete the omnipresent token, the `C_InitToken` function will fail with a return value of `CKR_TOKEN_WRITE_PROTECTED`.

Environment

Restriction: The calling program must be running as a Language-Environment-enabled (LE-enabled) application in TCB mode only. SRB mode is not supported.

Guideline: To use PKCS #11 in SRB mode, you must call the PKCS #11 ICSF callable services directly.

Cross memory considerations

On z/OS, an application can be running in either single address space mode, or in cross memory mode. The PKCS #11 standard has no concept of cross memory mode, so there is no predefined expected behavior for a PKCS #11 application running in cross memory mode.

When running in cross memory mode, the unit of work is running with PRIMARY set to an address space that differs from HOME. This PRIMARY space may be another address space that is logically part of the overall application (for example, if the application was designed to be cross memory aware) or it may be a daemon or subsystem address space dedicated to some system service that the calling application has invoked using a Program Call (PC). Either way, you should be aware of the following z/OS PKCS #11 application behaviors and associated guidelines.

- The C API invokes Language Environment (LE) services that are not supported in cross memory mode.

Guideline: To use PKCS #11 in cross memory mode, you must call the PKCS #11 ICSF callable services directly.

- Tokens, token objects, and session objects belonging to installation-defined PKCS #11 tokens (but not to the omnipresent token) are protected by RACF access control. Additionally, the ICSF callable services themselves may also be protected by RACF access control. In both cases, the user ID that is used for the access check is always associated with the unit of work. This is either the user ID assigned to the HOME ASCB or the user ID assigned to the Task Control Block (TCB) or System Request Block (SRB).

Guideline: If the PRIMARY address space function invoked by a PC uses PKCS #11 services, the user ID associated with the caller's unit of work must be appropriately permitted to the CRYPTOZ or CSFSERV resource being checked. If this is a system service, then all such callers must have access.

- FIPS140 compatibility mode behavior is also controlled by resources in the CRYPTOZ Class.

Guideline: If the system is configured for FIPS140 compatibility mode and the PRIMARY address space function invoked by a PC is expected to adhere to FIPS 140-2 restrictions, the user ID associated with the caller's unit of work should not be permitted to the CRYPTOZ FIPSEXEMPT resource being checked. If this is a system service, then all such callers should not be given access.

- By definition, session objects (including those belonging to the omnipresent token) are scoped to a single address space. For session objects belonging to installation defined PKCS #11 tokens, the scoping is to the HOME address space at the time of object creation, even if PRIMARY does not equal HOME. These objects are accessible to all units of work belonging to the HOME address space only, even if the PRIMARY address space function invoked by a PC is intended to be the logical owner of the PKCS #11 object.

In contrast, session objects belonging to the omnipresent token are scoped to the PRIMARY address space at the time of object creation and are addressable by all units of work running with that address space set as PRIMARY. For session objects created by system services invoked by a PC, such session objects would not be addressable by the caller once returning from the service call.

Guideline: System services invoked by a PC should use the omnipresent token instead of an installation defined PKCS #11 token when creating session objects that are to be owned by the system service.

- For certain multipart PKCS #11 cryptographic operations, ICSF will save session-state information across calls. This state information is scoped to the PRIMARY address space, similar to the scoping for omnipresent token objects. Such state objects are only addressable to units of work running with that address space set as PRIMARY.

Guideline: If you begin a multipart PKCS #11 cryptographic operation, you must remain running in the same PRIMARY address space in order to continue the operation.

Key types and mechanisms supported

ICSF supports the following PKCS #11 key types (CK_KEY_TYPE). All of these key types are supported in software. Whether they are also supported in hardware will depend on the limitations of your cryptographic hardware configuration.

- CKK_AES - key lengths 128, 192, and 256 bits
- CKK_BLOWFISH - key lengths 8 up to 448 bits (in increments of 8 bits)
- CKK_DES
- CKK_DES2
- CKK_DES3
- CKK_DH - key lengths 512 up to 2048 bits (in increments of 64 bits)
- CKK_DSA - key lengths 512 up to 2048 bit prime lengths (in increments of 64 bits)
- CKK_EC (CKK_ECDSA) - key lengths 160 up to 521 bits
- CKK_GENERIC_SECRET - key lengths 8 up to 2048 bits, unless further restricted by the generation mechanism:
 - CKM_DH_PKCS_DERIVE - key lengths 512 up to 2048 bits
 - CKM_SSL3_MASTER_KEY_DERIVE - 384-bit key lengths
 - CKM_SSL3_MASTER_KEY_DERIVE_DH - 384-bit key lengths
 - CKM_SSL3_PRE_MASTER_KEY_GEN - 384-bit key lengths
 - CKM_TLS_MASTER_KEY_DERIVE - 384-bit key lengths
 - CKM_TLS_MASTER_KEY_DERIVE_DH - 384-bit key lengths
 - CKM_TLS_PRE_MASTER_KEY_GEN - 384-bit key lengths
- CKK_RC4 - key lengths 8 up to 2048 bits
- CKK_RSA - key lengths 512 up to 4096 bits

The following table shows the mechanisms supported by different hardware configurations. All the mechanisms are supported in software, and some may be available in hardware. If the mechanism is available in hardware, ICSF will use the hardware mechanism. If the mechanism is not available in hardware, ICSF will use the software mechanism. The following table also shows the flags returned by the C_GetMechanismInfo function in the CK_MECHANISM_INFO structure. Whether or not the CKF_HW flag is returned in the CK_MECHANISM_INFO structure indicates whether or not the mechanism is supported in the hardware.

Table 4. Mechanism information as returned by C_GetMechanismInfo (CK_MECHANISM_INFO)

Type (CK_MECHANISM_TYPE)	Size factor	Flags
CKM_RSA_PKCS_KEY_PAIR_GEN	Bits	[CKF_HW] CKF_GENERATE_KEY_PAIR

Table 4. Mechanism information as returned by C_GetMechanismInfo (CK_MECHANISM_INFO) (continued)

Type (CK_MECHANISM_TYPE)	Size factor	Flags
CKM_DES_KEY_GEN ⁷	not applicable	[CKF_HW] CKF_GENERATE
CKM_DES2_KEY_GEN ⁷	not applicable	[CKF_HW] CKF_GENERATE
CKM_DES3_KEY_GEN	not applicable	[CKF_HW] CKF_GENERATE
CKM_RSA_PKCS ^{5,6}	Bits	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT CKF_WRAP CKF_UNWRAP CKF_SIGN CKF_VERIFY CKF_SIGN_RECOVER CKF_VERIFY_RECOVER
CKM_RSA_X_509 ^{5,6,7}	Bits	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT CKF_SIGN CKF_VERIFY CKF_SIGN_RECOVER CKF_VERIFY_RECOVER
CKM_MD2_RSA_PKCS ^{5,6}	Bits	CKF_SIGN CKF_VERIFY
CKM_MD5_RSA_PKCS ^{5,6}	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA1_RSA_PKCS ^{5,6}	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA224_RSA_PKCS ^{5,6}	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA256_RSA_PKCS ^{5,6}	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA384_RSA_PKCS ^{5,6}	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA512_RSA_PKCS ^{5,6}	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_DES_ECB ⁷	not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_DES_CBC ⁷	not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_DES_CBC_PAD ⁷	not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT CKF_WRAP CKF_UNWRAP
CKM_DES3_ECB ³	not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_DES3_CBC ³	not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_DES3_CBC_PAD ³	not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT CKF_WRAP CKF_UNWRAP
CKM_SHA_1	not applicable	[CKF_HW] CKF_DIGEST
CKM_SHA224	not applicable	[CKF_HW] CKF_DIGEST
CKM_SHA256	not applicable	[CKF_HW] CKF_DIGEST
CKM_SHA384	not applicable	[CKF_HW] CKF_DIGEST
CKM_SHA512	not applicable	[CKF_HW] CKF_DIGEST
CKM_RIPEMD160	not applicable	CKF_DIGEST
CKM_MD2	not applicable	CKF_DIGEST
CKM_MD5	not applicable	CKF_DIGEST
CKM_AES_KEY_GEN	Bytes	[CKF_HW] CKF_GENERATE
CKM_AES_ECB ³	Bytes	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_AES_CBC ³	Bytes	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_AES_CBC_PAD ³	Bytes	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT CKF_WRAP CKF_UNWRAP
CKM_AES_GCM ^{3,7}	Bytes	CKF_ENCRYPT CKF_DECRYPT
CKM_DSA_KEY_PAIR_GEN	Bits	[CKF_HW] CKF_GENERATE_KEY_PAIR
CKM_DH_PKCS_KEY_PAIR_GEN	Bits	[CKF_HW] CKF_GENERATE_KEY_PAIR

Table 4. Mechanism information as returned by `C_GetMechanismInfo` (`CK_MECHANISM_INFO`) (continued)

Type (<code>CK_MECHANISM_TYPE</code>)	Size factor	Flags
<code>CKM_EC_KEY_PAIR_GEN</code>	Bits	[CKF_HW] CKF_GENERATE_KEY_PAIR CKF_EC_F_P ¹ CKF_EC_NAMEDCURVE ² CKF_EC_UNCOMPRESS
<code>CKM_DSA_PARAMETER_GEN</code>	Bits	[CKF_HW] CKF_GENERATE
<code>CKM_DH_PKCS_PARAMETER_GEN</code>	Bits	[CKF_HW] CKF_GENERATE
<code>CKM_BLOWFISH_KEY_GEN</code> ⁷	Bytes	[CKF_HW] CKF_GENERATE
<code>CKM_RC4_KEY_GEN</code> ⁷	Bits	[CKF_HW] CKF_GENERATE
<code>CKM_SSL3_PRE_MASTER_KEY_GEN</code> ⁷	Bytes	[CKF_HW] CKF_GENERATE
<code>CKM_TLS_PRE_MASTER_KEY_GEN</code> ⁷	Bytes	[CKF_HW] CKF_GENERATE
<code>CKM_GENERIC_SECRET_KEY_GEN</code> ³	Bits	[CKF_HW] CKF_GENERATE
<code>CKM_BLOWFISH_CBC</code> ^{4,7}	Bytes	CKF_ENCRYPT CKF_DECRYPT
<code>CKM_RC4</code> ^{4,7}	Bits	CKF_ENCRYPT CKF_DECRYPT
<code>CKM_DSA_SHA1</code>	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
<code>CKM_DSA</code>	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
<code>CKM_ECDSA_SHA1</code>	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY CKF_EC_F_P ¹ CKF_EC_NAMEDCURVE ² CKF_EC_UNCOMPRESS
<code>CKM_ECDSA</code>	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY CKF_EC_F_P ¹ CKF_EC_NAMEDCURVE ² CKF_EC_UNCOMPRESS
<code>CKM_MD5_HMAC</code>	not applicable	CKF_SIGN CKF_VERIFY
<code>CKM_SHA_1_HMAC</code>	not applicable	[CKF_HW] CKF_SIGN CKF_VERIFY
<code>CKM_SHA224_HMAC</code>	not applicable	[CKF_HW] CKF_SIGN CKF_VERIFY
<code>CKM_SHA256_HMAC</code>	not applicable	[CKF_HW] CKF_SIGN CKF_VERIFY
<code>CKM_SHA384_HMAC</code>	not applicable	[CKF_HW] CKF_SIGN CKF_VERIFY
<code>CKM_SHA512_HMAC</code>	not applicable	[CKF_HW] CKF_SIGN CKF_VERIFY
<code>CKM_SSL3_MD5_MAC</code> ⁷	Bits	CKF_SIGN CKF_VERIFY
<code>CKM_SSL3_SHA1_MAC</code> ⁷	Bits	CKF_SIGN CKF_VERIFY
<code>CKM_DH_PKCS_DERIVE</code>	Bits	[CKF_HW] CKF_DERIVE
<code>CKM_ECDH1_DERIVE</code>	Bits	[CKF_HW] CKF_DERIVE CKF_EC_F_P ¹ CKF_EC_NAMEDCURVE ² CKF_EC_UNCOMPRESS
<code>CKM_SSL3_MASTER_KEY_DERIVE</code> ⁷	Bytes	CKF_DERIVE
<code>CKM_SSL3_MASTER_KEY_DERIVE_DH</code> ⁷	Bytes	CKF_DERIVE
<code>CKM_SSL3_KEY_AND_MAC_DERIVE</code> ⁷	not applicable	CKF_DERIVE
<code>CKM_TLS_MASTER_KEY_DERIVE</code> ⁷	Bytes	CKF_DERIVE
<code>CKM_TLS_MASTER_KEY_DERIVE_DH</code> ⁷	Bytes	CKF_DERIVE
<code>CKM_TLS_KEY_AND_MAC_DERIVE</code> ⁷	not applicable	CKF_DERIVE
<code>CKM_TLS_PRFF</code> ⁷	not applicable	CKF_DERIVE

Table 4. Mechanism information as returned by C_GetMechanismInfo (CK_MECHANISM_INFO) (continued)

Type (CK_MECHANISM_TYPE)	Size factor	Flags
CKM_IBM_ATTRIBUTEBOUND_WRAP ⁸ (vendor specific mechanism - 0x80020004). IBM proprietary wrap/unwrap mechanism that includes the Boolean usage attributes along with the key data. Only supported for secure keys that have the CKA_IBM_ATTRIBUTEBOUND attribute set TRUE	not applicable	[CKF_HW] CKF_WRAP CKF_UNWRAP
CKM_PBE_SHA1_DES3_EDE_CBC	not applicable	[CKF_HW] CKF_GENERATE
CKM_RSA_PKCS_PSS ⁹	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA1_RSA_PKCS_PSS ⁹	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA224_RSA_PKCS_PSS ⁹	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA256_RSA_PKCS_PSS ⁹	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA384_RSA_PKCS_PSS ⁹	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA512_RSA_PKCS_PSS ⁹	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY

Footnotes for Table 4 on page 23

¹ The PKCS #11 standard designates two ways of implementing Elliptic Curve Cryptography, nicknamed F_p and F_2^m . z/OS PKCS #11 supports the F_p variety only.

² ANSI X9.62 has the following ASN.1 definition for Elliptic Curve domain parameters:

```
Parameters ::= CHOICE {
    ecParameters  ECParameters,
    namedCurve    OBJECT IDENTIFIER,
    implicitlyCA  NULL }
```

z/OS PKCS #11 supports the specification of CKA_EC_PARAMS attribute using the namedCurve CHOICE. The following NIST-recommended named curves are supported:

- secp192r1 – { 1 2 840 10045 3 1 1 }
- secp224r1 – { 1 3 132 0 33 }
- secp256r1 – { 1 2 840 10045 3 1 7 }
- secp384r1 – { 1 3 132 0 34 }
- secp521r1 – { 1 3 132 0 35 }

The following Brainpool-defined named curves are supported:

- brainpoolP160r1 – { 1 3 36 3 3 2 8 1 1 1 }
- brainpoolP192r1 – { 1 3 36 3 3 2 8 1 1 3 }
- brainpoolP224r1 – { 1 3 36 3 3 2 8 1 1 5 }
- brainpoolP256r1 – { 1 3 36 3 3 2 8 1 1 7 }
- brainpoolP320r1 – { 1 3 36 3 3 2 8 1 1 9 }
- brainpoolP384r1 – { 1 3 36 3 3 2 8 1 1 11 }
- brainpoolP512r1 – { 1 3 36 3 3 2 8 1 1 13 }

In addition, z/OS PKCS #11 has limited support for the ecParameters CHOICE. When specified, the DER encoding must contain the optional cofactor field and must not contain the optional Curve.seed field. Also, calls to C_GetAttributeValue

to retrieve the CKA_EC_PARAMS attribute will always return the value in the namedCurve form regardless of how the attribute was specified when the object was created. Due to these limitations, the CKF_EC_ECPARAMETERS flag is not turned on for the applicable mechanisms.

³ Mechanism not present on a system that is export controlled.

⁴ Mechanism limited to 56-bit on a system that is export controlled.

⁵ In general, z/OS PKCS #11 expects RSA private keys to be in Chinese Remainder Theorem (CRT) format. However, for Decrypt, Sign, or UnwrapKey (z890, z990 or higher only) where one of the following is true, the shorter Modulus Exponent (ME) is permitted:

- There is an accelerator present and the key is less than or equal to 2048 bits in length.
- There is a coprocessor present and the key is less than or equal to 1024 bits in length and FIPS restrictions don't apply.

⁶ RSA public or private keys that have a public exponent greater than 8 bytes in length can only be used when a coprocessor or accelerator is present.

⁷ Mechanism supported for clear keys only.

⁸ Mechanism supported for secure keys only.

⁹ PARAM field restrictions for PSS algorithms:

```
typedef struct CK_RSA_PKCS_PSS_PARAMS {  
    CK_MECHANISM_TYPE hashAlg;  
    CK_RSA_PKCS_MGF_TYPE mgf;  
    CK_ULONG sLen;  
} CK_RSA_PKCS_PSS_PARAMS;
```

- For mechanisms other than CKM_RSA_PKCS_PSS, the hashAlg must match the mechanism specified. For mechanism CKM_RSA_PKCS_PSS, the hashAlg must be a supported SHA algorithm.
- mgf must match the algorithm specified by hashAlg
- sLen must be either 0 or the size of the output of the hashAlg specified.

The following table lists the mechanisms supported by specific cryptographic hardware. When a particular mechanism is not available in hardware, ICSF will use the software implementation of the mechanism.

Table 5. Mechanisms supported by specific cryptographic hardware

Machine type and cryptographic hardware	Mechanisms supported	Notes
z890, z990 - PCIXCC	CKM_DES_KEY_GEN CKM_DES2_KEY_GEN CKM_DES3_KEY_GEN CKM_RSA_PKCS CKM_RSA_X_509 CKM_MD5_RSA_PKCS CKM_SHA1_RSA_PKCS CKM_DES_CBC CKM_DES_CBC_PAD CKM_DES3_CBC CKM_DES3_CBC_PAD CKM_SHA_1 CKM_BLOWFISH_KEY_GEN CKM_RC4_KEY_GEN CKM_AES_KEY_GEN CKM_SSL3_PRE_MASTER_KEY_GEN CKM_TLS_PRE_MASTER_KEY_GEN CKM_GENERIC_SECRET_KEY_GEN CKM_RSA_PKCS_KEY_PAIR_GEN CKM_EC_KEY_PAIR_GEN CKM_DES_ECB CKM_DES3_ECB	This is the base set. RSA private key operations limited to 1024 bits in length (maximum) and no key pair generation capability.
z890, z990 - CEX2C	PCIXCC set plus: CKM_RSA_PKCS_KEY_PAIR_GEN CKM_DES_ECB CKM_DES3_ECB	RSA private key operations limited to 40496 bits in length (maximum).
z9 - CEX2C	PCIXCC set plus: CKM_RSA_PKCS_KEY_PAIR_GEN CKM_DES_ECB CKM_DES3_ECB CKM_SHA224_RSA_PKCS CKM_SHA256_RSA_PKCS CKM_SHA224 CKM_SHA256 CKM_AES_CBC CKM_AES_CBC_PAD CKM_AES_ECB	AES key operations limited to 128 bits in length (maximum).
z10 - CEX2C or CEX3C	z9 CEX2C set plus: CKM_SHA384_RSA_PKCS CKM_SHA512_RSA_PKCS CKM_SHA384 CKM_SHA512	AES key operations limited to 256 bits in length (maximum).
IBM zEnterprise 196 - CEX3C	z9 CEX2C set plus: CKM_SHA384_RSA_PKCS CKM_SHA512_RSA_PKCS CKM_SHA384 CKM_SHA512	AES key operations limited to 256 bits in length (maximum). RSA private key operations limited to 4096 bits in length (maximum).

Table 5. Mechanisms supported by specific cryptographic hardware (continued)

Machine type and cryptographic hardware	Mechanisms supported	Notes
IBM zEnterprise EC12 or IBM zEnterprise BC12 with an Enterprise PKCS #11 coprocessor	z10 set plus: CKM_IBM_ATTRIBUTEBOUND_WRAP CKM_PBE_SHA1_DES3_EDE_CBC CKM_DSA_PARAMETER_GEN CKM_DH_PKCS_KEY_PAIR_GEN CKM_DH_PKCS_DERIVE CKM_ECDH1_DERIVE CKM_RSA_PKCS_PSS CKM_SHA1_RSA_PKCS_PSS CKM_SHA224_RSA_PKCS_PSS CKM_SHA256_RSA_PKCS_PSS CKM_SHA384_RSA_PKCS_PSS CKM_SHA512_RSA_PKCS_PSS	Requires the Sept. 2013 or later licensed internal code (LIC)
IBM z13	zEC12 and zBC12 set	

The following table lists the algorithms and uses (by mechanism) that are not allowed when operating in compliance with FIPS 140-2. For more information about how the z/OS PKCS #11 services can be configured to operate in compliance with the FIPS 140-2 standard, refer to “Operating in compliance with FIPS 140-2” on page 11.

Table 6. Restricted algorithms and uses when running in compliance with FIPS 140-2

Algorithm	Mechanisms	Usage disallowed
RIPEMD	CKM_RIPEMD160	All
MD2	CKM_MD2, CKM_MD2_RSA_PKCS	All
MD5	CKM_MD5, CKM_MD5_RSA_PKCS, CKM_MD5_HMAC	All
SSL3	CKM_SSL3_MD5_MAC, CKM_SSL3_SHA1_MAC, CKM_SSL3_MASTER_KEY_DERIVE, CKM_SSL3_MASTER_KEY_DERIVE_DH, CKM_SSL3_KEY_AND_MAC_DERIVE	All
TLS	CKM_TLS_MASTER_KEY_DERIVE, CKM_TLS_MASTER_KEY_DERIVE_DH, CKM_TLS_KEY_AND_MAC_DERIVE	Base key sizes less than 10 bytes
Diffie Hellman	CKM_DH_PKCS_DERIVE	Prime size less than 1024 bits
	CKM_DH_PKCS_PARAMETER_GEN	Prime sizes other than 1024 or 2048 bits
DSA	CKM_DSA_SHA1, CKM_DSA	Prime sizes less than 1024 bits
DSA	CKM_DSA_PARAMETER_GEN, CKM_DSA_KEY_PAIR_GEN or Sign	Combinations other than the following: <ul style="list-style-type: none"> • Prime size = 1024 bits, subprime size = 160 bits • Prime size = 2048 bits, subprime size = 224 bits, or 256 bits
Single DES	CKM_DES_ECB, CKM_DES_CBC, CKM_DES_CBC_PAD	All
Triple DES	CKM_DES3_ECB, CKM_DES3_CBC, CKM_DES3_CBC_PAD	Two key Triple DES

Table 6. Restricted algorithms and uses when running in compliance with FIPS 140-2 (continued)

Algorithm	Mechanisms	Usage disallowed
Blowfish	CKM_BLOWFISH_KEY_GEN, CKM_BLOWFISH_CBC	All
RC4	CKM_RC4	All
RSA	CKM_RSA_X_509	All
RSA	CKM_RSA_PKCS	Key sizes less than 1024 bits
RSA	CKM_RSA_PKCS_KEY_PAIR_GEN or Sign without an active accelerator	Key sizes that are less than 1024 bits or not a multiple of 256 bits or public key exponents less than 0x010001
HMAC	CKM_SHA_1, CKM_SHA224, CKM_SHA256, CKM_SHA384, CKM_SHA512	Base key sizes less than one half the output size
AES GCM	CKM_AES_GCM	GCM encryption or GMAC generation with externally generated initialization vectors. Initialization vector lengths other than 12 bytes. Tag byte sizes 4 and 8

Objects and attributes supported

ICSF supports the following PKCS #11 object types (CK_OBJECT_CLASS):

- CKO_DATA
- CKO_CERTIFICATE - CKC_X_509 only
- CKO_DOMAIN_PARAMETERS - CKK_DSA and CKK_DH only
- CKO_PUBLIC_KEY - CKK_RSA, CKK_EC (CKK_ECDSA), CKK_DSA, and CKK_DH only
- CKO_PRIVATE_KEY - CKK_RSA, CKK_EC (CKK_ECDSA), CKK_DSA, and CKK_DH only
- CKO_SECRET_KEY - CKK_DES, CKK_DES2, CKK_DES3, CKK_AES, CKK_BLOWFISH, and CKK_RC4, CKK_GENERIC_SECRET only

The footnotes described in Table 7 are taken from the PKCS #11 specification and apply to the attribute tables that follow.

Table 7. Common footnotes for object attribute tables

Footnote number	Footnote meaning
1	Must be specified when object is created with C_CreateObject .
2	Must <i>not</i> be specified when object is created with C_CreateObject .
3	Must be specified when object is generated with C_GenerateKey or C_GenerateKeyPair .
4	Must <i>not</i> be specified when object is generated with C_GenerateKey or C_GenerateKeyPair .
5	Must be specified when object is unwrapped with C_UnwrapKey .
6	Must <i>not</i> be specified when object is unwrapped with C_UnwrapKey .
7	Cannot be revealed if object has its CKA_SENSITIVE attribute set to TRUE or its CKA_EXTRACTABLE attribute set to FALSE.

Table 7. Common footnotes for object attribute tables (continued)

Footnote number	Footnote meaning
8	May be modified after object is created with a C_SetAttributeValue call, or in the process of copying object with a C_CopyObject call. However, it is possible that a particular token may not permit modification of the attribute, or may not permit modification of the attribute during the course of a C_CopyObject call.
9	Default value is token-specific, and may depend on the values of other attributes.
10	Can only be set to TRUE by the SO user.
11	May be changed during a C_CopyObject call but not on a C_SetAttributeValue call
12	Attribute cannot be changed once set to CK_TRUE. It becomes a read only attribute.
13	May be changed to CK_TRUE during the course of a copy operation but only if the source object is not already a secure key. When the source object is a secure key, the attribute is read only.
14	If CKA_PRIVATE=TRUE , can only be set TRUE by a user who is a Strong SO or a Weak USER who is also an SO.
15	Only modifiable with an Enterprise PKCS #11 coprocessor configured with August 2013 or later licensed internal code (LIC).

Table 8. Data object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_CLASS ¹	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created (or generated) only.
CKA_TOKEN ¹¹	CK_BBOOL	Default value on create is FALSE. Object hardened to the TKDS if TRUE. An application can specify the value when the object is created (or generated) only.
CKA_PRIVATE ¹¹	CK_BBOOL	Default value on create is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_MODIFIABLE ¹¹	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_LABEL	Printable EBCDIC string	Application-specific nickname. Limited to 32 characters. Default is empty. The string is assumed to come from the IBM1047 code page. An application can set or change the value at any time.
CKA_ID	Byte array	Key or other identifier. Default is empty. An application can set or change the value at any time.
CKA_VALUE	Byte array	Any value. Default is empty. An application can set or change the value at any time.

Table 8. Data object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_APPLICATION	Printable EBCDIC string	Description of the application that created the object. Default is empty. The string is assumed to come from the IBM1047 code page. An application can set or change the value at any time.
CKA_OBJECT_ID	Byte array	DER-encoded OID. Default is empty. An application can set or change the value at any time.
CKA_IBM_ICSF_HANDLE (vendor specific attribute - 0x80010009)	Printable EBCDIC string	44-character ICSF record locator value. This attribute may be retrieved via C_GetAttributeValue. It is not valid in the attribute list of any other function or callable service.

Table 9. X.509 certificate object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_CLASS ¹	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created (or generated) only.
CKA_TOKEN ¹¹	CK_BBOOL	Default value on create is FALSE. Object hardened to the TKDS if TRUE. An application can specify the value when the object is created (or generated) only.
CKA_PRIVATE ¹¹	CK_BBOOL	Default value on create is FALSE. An application can specify the value when the object is created (or generated) only.
CKA_MODIFIABLE ¹¹	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_LABEL	Printable EBCDIC string	Application-specific nickname. Limited to 32 characters. Default is empty. The string is assumed to come from the IBM1047 code page. An application can set or change the value at any time.
CKA_CERTIFICATE_TYPE	CK_CERTIFICATE_TYPE	Always CKC_X_509. An application can specify the value when the object is created (or generated) only.
CKA_TRUSTED	CK_BBOOL	Always set to TRUE. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.

Table 9. X.509 certificate object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_SUBJECT	Byte array	DER-encoding as found in certificate. If not specified, ICSF sets it from the certificate. If specified, ICSF enforces that it matches the subject in the certificate. An application can specify the value when the object is created (or generated) only.
CKA_ID	Byte array	Key identifier. Default is empty. An application can set or change the value at any time.
CKA_ISSUER	Byte array	DER-encoding as found in certificate. If not specified, ICSF sets from the certificate. If specified, ICSF enforces that it matches the issuer in the certificate An application can specify the value when the object is created (or generated) only.
CKA_SERIAL_NUMBER	Byte array	DER-encoding as found in certificate. If not specified, ICSF sets from the certificate. If specified, ICSF enforces that it matches the serial number in the certificate. An application can specify the value when the object is created (or generated) only.
CKA_VALUE	Byte array	This is the DER-encoding of the certificate. (Required.) An application can specify the value when the object is created (or generated) only.
CKA_CERTIFICATE_CATEGORY	CK_ULONG	Categorization of the certificate: 1 Token user 2 Certificate authority 3 Other entity If not specified, ICSF sets it to 2 if the certificate has the BasicConstraints CA flag on. Otherwise it is not set. Note: If specified (or defaulted) to 2, the certificate is considered a CA certificate. The user must have appropriate authority. An application can set or change the value at any time.
CKA_APPLICATION	Printable EBCDIC string	Description of the application that created the object. Default is empty. The string is assumed to come from the IBM1047 code page. An application can specify the value when the object is created (or generated) only.
CKA_IBM_DEFAULT (vendor specific attribute - 0x80000002)	CK_BBOOL	Default flag. Default is FALSE. An application can set or change the value at any time.

Table 9. X.509 certificate object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_IBM_ICSF_HANDLE (vendor specific attribute - 0x80010009)	Printable EBCDIC string	44-character ICSF record locator value. This attribute may be retrieved via C_GetAttributeValue. It is not valid in the attribute list of any other function or callable service.

Table 10. Secret key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_CLASS ¹	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created (or generated) only.
CKA_TOKEN ¹¹	CK_BBOOL	Default value on create is FALSE. Object hardened to the TKDS if TRUE. An application can specify the value when the object is created (or generated) only.
CKA_PRIVATE ¹¹	CK_BBOOL	Default value on create is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_MODIFIABLE ¹¹	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_LABEL	Printable EBCDIC string	Application-specific nickname. Limited to 32 characters. Default is empty. The string is assumed to come from the IBM1047 code page. An application can set or change the value at any time.
CKA_ID	Byte array	Default is empty. An application can set or change the value at any time.
CKA_KEY_TYPE ^{1,5}	CK_KEY_TYPE	Type of key: CKK_DES, CKK_DES2, CKK_DES3, CKK_BLOWFISH, CKK_RC4, CKK_GENERIC_SECRET, or CKK_AES. An application can specify the value when the object is created (or generated) only.
CKA_START_DATE ⁸	CK_DATE	Start date for the key. Default is empty. An application can set or change the value at any time.
CKA_END_DATE ⁸	CK_DATE	End date for the key. Default is empty. An application can set or change the value at any time.

Table 10. Secret key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_DERIVE ⁸	CK_BBOOL	TRUE if key supports key derivation (other keys can be derived from this one). Default is TRUE. An application can set or change the value at any time.
CKA_LOCAL ^{2, 4, 6}	CK_BBOOL	TRUE only if key was generated locally. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_GEN_MECHANISM ^{2, 4, 6}	CK_MECHANISM_TYPE	Identifier of the mechanism used to generate the key. Always CK_UNAVAILABLE_INFORMATION. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_ENCRYPT ⁸	CK_BBOOL	TRUE if key supports encryption ⁹ . Default is TRUE. An application can set or change the value at any time.
CKA_VERIFY ⁸	CK_BBOOL	TRUE if key supports verification where the signature is an appendix to the data. Default is TRUE. An application can set or change the value at any time.
CKA_WRAP ⁸	CK_BBOOL	TRUE if key supports wrapping (can be used to wrap other keys). ⁹ Default is TRUE. An application can set or change the value at any time.
CKA_DECRYPT ⁸	CK_BBOOL	TRUE if key supports decryption. ⁹ Default is TRUE. An application can set or change the value at any time.
CKA_SIGN ⁸	CK_BBOOL	TRUE if key supports signatures where the signature is an appendix to the data. ⁹ Default is TRUE. An application can set or change the value at any time.
CKA_UNWRAP ⁸	CK_BBOOL	TRUE if key supports unwrapping (can be used to unwrap other keys). ⁹ Default is TRUE. An application can set or change the value at any time.
CKA_EXTRACTABLE ⁸	CK_BBOOL	TRUE if key is extractable. Caller can change from TRUE to FALSE only. Default is TRUE. An application can set or change the value, as per PKCS #11 restrictions.

Table 10. Secret key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_SENSITIVE ⁸	CK_BBOOL	TRUE if key is sensitive. Caller can change from FALSE to TRUE only. Default is FALSE. An application can set or change the value, as per PKCS #11 restrictions.
CKA_ALWAYS_SENSITIVE ^{2, 4, 6}	CK_BBOOL	TRUE if key has always had the CKA_SENSITIVE attribute set to TRUE. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_NEVER_EXTRACTABLE ^{2, 4, 6}	CK_BBOOL	TRUE if key has never had the CKA_EXTRACTABLE attribute set to TRUE. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_VALUE ^{1, 4, 6, 7}	Byte array	The key. Sensitive key part. An application can specify the value when the object is created (or generated) only.
CKA_VALUE_LEN ^{2, 3}	CK_ULONG	Length of the key in bytes (AES, Blowfish, RC4, and Generic secret keys only). An application can specify the value when the object is generated only.
CKA_APPLICATION	Printable EBCDIC string	Description of the application that created the object. Default is empty. The string is assumed to come from the IBM1047 code page. An application can specify the value when the object is created (or generated) only.
CKA_IBM_FIPS140 (vendor specific attribute 0x80000005)	CK_BBOOL	TRUE if the key must only be used in a FIPS 140-2 compliant fashion. The default value is FALSE. An application can specify the value when the object is created (or generated) only.
CKA_TRUSTED ^{10, 14}	CK_BBOOL	The wrapping key can be used to wrap keys with CKA_WRAP_WITH_TRUSTED set to CK_TRUE. Always set CK_FALSE when the key is created or generated. May be set to CK_TRUE via C_SetAttributeValue, with restrictions
CKA_WRAP_WITH_TRUSTED ¹²	CK_BBOOL	CK_TRUE if the key can only be wrapped with a wrapping key that has CKA_TRUSTED set to CK_TRUE. Default is CK_FALSE

Table 10. Secret key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes								
CKA_IBM_SECURE ^{6, 11, 12, 13} (vendor specific attribute - 0x80000006)	CK_BBOOL	<p>CK_TRUE if the key is an Enterprise PKCS #11 coprocessor secure key.</p> <p>A secure key may be requested by setting this attribute CK_TRUE during key creation or generation. For key generation, if set CK_FALSE or not specified at all, ICSF will determine the security.</p> <p>When this is set TRUE by the caller or ICSF, the key is treated as sensitive (CKA_SENSITIVE is set TRUE).</p>								
CKA_IBM_ALWAYS_SECURE ^{2, 4, 6} (vendor specific attribute - 0x80000008)	CK_BBOOL	<p>CK_TRUE if key has always had the CKA_IBM_SECURE attribute set to CK_TRUE. Only applicable to secure keys. This attribute will not have a value for clear keys.</p>								
CKA_IBM_CARD_COMPLIANCE ^{2, 4, 6, 15} (vendor specific attribute - 0x80000007)	CK_ULONG	<p>A bit mask field indicating the Enterprise PKCS #11 coprocessor compliance mode of the secure key:</p> <table style="margin-left: 20px;"> <tr> <td>FIPS2009</td> <td>1</td> </tr> <tr> <td>BSI2009</td> <td>2</td> </tr> <tr> <td>FIPS2011</td> <td>4</td> </tr> <tr> <td>BSI2011</td> <td>8</td> </tr> </table> <p>Only applicable to secure keys. This attribute will not have a value for clear keys. When changing the value, the new value must specify the current mode or modes and any additional new modes. Changing the value to 0 results in the compliance mode being reset to the current compliance mode of the coprocessor.</p>	FIPS2009	1	BSI2009	2	FIPS2011	4	BSI2011	8
FIPS2009	1									
BSI2009	2									
FIPS2011	4									
BSI2011	8									
CKA_IBM_ATTRBOUND ^{2, 6} (vendor specific attribute - 0x80010004)	CK_BBOOL	<p>CK_TRUE has the following meaning:</p> <ul style="list-style-type: none"> • The key must only be exported with its boolean usage attributes. • The key may be used as a signing or verification key for attribute bound wrap/unwrap. • The key may be used as a wrapping or unwrapping key for attribute bound wrap/unwrap. • The key may not be used as a wrapping key for non-attribute bound wrap. <p>May only be set CK_TRUE during key generation. The default value is CK_FALSE.</p> <p>When this is set TRUE by the caller, the key will be a secure key (CKA_IBM_SECURE is set TRUE).</p> <p>Only applicable to secure keys. This attribute will not have a value for clear keys.</p>								

Table 10. Secret key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_IBM_ICSF_HANDLE (vendor specific attribute - 0x80010009)	Printable EBCDIC string	44-character ICSF record locator value. This attribute may be retrieved via C_GetAttributeValue. It is not valid in the attribute list of any other function or callable service.

Table 11. Public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_CLASS ¹	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created (or generated) only.
CKA_TOKEN ¹¹	CK_BBOOL	Default value on create is FALSE. Object hardened to the TKDS if TRUE. An application can specify the value when the object is created (or generated) only.
CKA_PRIVATE ¹¹	CK_BBOOL	Default value on create is FALSE. An application can specify the value when the object is created (or generated) only.
CKA_MODIFIABLE ¹¹	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_LABEL	Printable EBCDIC string	Application-specific nickname. Limited to 32 characters. Default is empty. The string is assumed to come from the IBM1047 code page. An application can set or change the value at any time.
CKA_TRUSTED ^{10, 14}	CK_BBOOL	The wrapping key can be used to wrap keys with CKA_WRAP_WITH_TRUSTED set to CK_TRUE. Always set CK_FALSE when the key is created or generated. May be set to CK_TRUE via C_SetAttributeValue, with restrictions
CKA_SUBJECT	Byte array	DER-encoding. Default empty. An application can set or change the value at any time.
CKA_ID	Byte array	Key identifier. Default empty. An application can set or change the value at any time.
CKA_KEY_TYPE ^{1, 5}	CK_KEY_TYPE	Type of key. CKK_RSA, CKK_EC, CKK_DSA, and CKK_DH only. An application can specify the value when the object is created (or generated) only.

Table 11. Public key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_START_DATE ⁸	CK_DATE	Start date for the key. Default empty. An application can set or change the value at any time.
CKA_END_DATE ⁸	CK_DATE	End date for the key. Default empty. An application can set or change the value at any time.
CKA_DERIVE ⁸	CK_BBOOL	TRUE if key supports key derivation (if other keys can be derived from this one). Default is TRUE. An application can set or change the value at any time.
CKA_LOCAL ^{2, 4, 6}	CK_BBOOL	TRUE only if key was generated locally. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_KEY_GEN_MECHANISM ^{2, 4, 6}	CK_MECHANISM_TYPE	Identifier of the mechanism used to generate the key. Always CK_UNAVAILABLE_INFORMATION. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_ENCRYPT ⁸	CK_BBOOL	TRUE if key supports encryption. ⁹ Default is TRUE. An application can set or change the value at any time.
CKA_VERIFY ⁸	CK_BBOOL	TRUE if key supports verification where the signature is an appendix to the data. Default is TRUE. An application can set or change the value at any time.
CKA_VERIFY_RECOVER ⁸	CK_BBOOL	TRUE if key supports verification where the data is recovered from the signature. ⁹ Default is TRUE. An application can set or change the value at any time.
CKA_WRAP ⁸	CK_BBOOL	TRUE if key supports wrapping (can be used to wrap other keys). ⁹ Default is TRUE. An application can set or change the value at any time.
CKA_APPLICATION	Printable EBCDIC string	Description of the application that created the object. Default is empty. The string is assumed to come from the IBM1047 code page. An application can specify the value when the object is created (or generated) only.
CKA_IBM_FIPS140 (vendor specific attribute 0x80000005)	CK_BBOOL	TRUE if the key must only be used in a FIPS 140-2 compliant fashion. The default value is FALSE. An application can specify the value when the object is created (or generated) only.

Table 11. Public key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes								
CKA_IBM_SECURE ^{11, 12, 13} (vendor specific attribute - 0x80000006)	CK_BBOOL	<p>CK_TRUE if the key is an Enterprise PKCS #11 coprocessor secure key.</p> <p>A secure key may be requested by setting this attribute CK_TRUE during key creation or generation.</p> <p>For key-pair generation, if set CK_FALSE or not specified at all, ICSF will determine the security. If set CK_TRUE by the caller or ICSF, the matching private key will also be a secure key.</p>								
CKA_IBM_CARD_COMPLIANCE ^{2, 4, 6, 15} (vendor specific attribute - 0x80000007)	CK_ULONG	<p>A bit mask field indicating the Enterprise PKCS #11 coprocessor compliance mode of the secure key:</p> <table border="0"> <tr> <td>FIPS2009</td> <td>1</td> </tr> <tr> <td>BSI2009</td> <td>2</td> </tr> <tr> <td>FIPS2011</td> <td>4</td> </tr> <tr> <td>BSI2011</td> <td>8</td> </tr> </table> <p>Only applicable to secure keys. This attribute will not have a value for clear keys. When changing the value, the new value must specify the current mode or modes and any additional new modes. Changing the value to 0 results in the compliance mode being reset to the current compliance mode of the coprocessor.</p>	FIPS2009	1	BSI2009	2	FIPS2011	4	BSI2011	8
FIPS2009	1									
BSI2009	2									
FIPS2011	4									
BSI2011	8									
CKA_IBM_ATTRBOUND ^{11, 12, 13} (vendor specific attribute - 0x80010004)	CK_BBOOL	<p>CK_TRUE if the key may be used as a wrapping key to export attribute bound secret keys. (Such public keys may not be used for non-attribute bound wrap.) CK_TRUE also means that the key may be used as an attribute bound unwrap verification key. The default value is CK_FALSE.</p> <p>An attribute bound key may be requested by setting this attribute CK_TRUE during key creation or generation.</p> <p>When this is set TRUE by the caller, the key will be a secure key (CKA_IBM_SECURE is set TRUE).</p> <p>Only applicable to secure keys. This attribute will not have a value for clear keys.</p>								
CKA_IBM_ICSF_HANDLE (vendor specific attribute - 0x80010009)	Printable EBCDIC string	<p>44-character ICSF record locator value. This attribute may be retrieved via C_GetAttributeValue. It is not valid in the attribute list of any other function or callable service.</p>								

Table 12. RSA public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_MODULUS ^{1, 4}	Big integer	<p>Modulus n</p> <p>An application can specify the value when the object is created (or generated) only.</p>

Table 12. RSA public key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_MODULUS_BITS ^{2,3}	CK_ULONG	Length in bits of modulus n An application can specify the value when the object is created (or generated) only.
CKA_PUBLIC_EXPONENT ¹	Big integer	Public exponent e An application can specify the value when the object is created (or generated) only.

Table 13. DSA public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_PRIME ^{1,3}	Big integer	Prime p (512 to 2048 bits in steps of 64 bits)
CKA_SUBPRIME ^{1,3}	Big integer	Subprime q (160 bits for $p \leq 1024$ bits, 224 bits or 256 bits for $p > 1024$ bits)
CKA_BASE ^{1,3}	Big integer	Base g
CKA_VALUE ^{1,4}	Big integer	Public value y

Table 14. Diffie-Hellman public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_PRIME ^{1,3}	Big integer	Prime p (512 to 2048 bits in steps of 64 bits)
CKA_BASE ^{1,3}	Big integer	Base g
CKA_VALUE ^{1,4}	Big integer	Public value y

Table 15. Elliptic Curve public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_EC_PARAMS ^{1,3} (CKA_ECDSA_PARAMS)	Byte Array	DER-encoding of an ANSI X9.62 Parameters value
CKA_EC_POINT ^{1,4}	Byte Array	DER-encoding of an ANSI X9.62 ECPoint value Q

Table 16. Private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_CLASS ¹	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created (or generated) only.
CKA_TOKEN ¹¹	CK_BBOOL	Default value on create is FALSE. Object hardened to the TKDS if TRUE. An application can specify the value when the object is created (or generated) only.

Table 16. Private key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_PRIVATE ¹¹	CK_BBOOL	Default value on create is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_MODIFIABLE ¹¹	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_LABEL	Printable EBCDIC string	Application-specific nickname. Limited to 32 characters. Default is empty. The string is assumed to come from the IBM1047 code page. An application can set or change the value at any time.
CKA_SUBJECT	Byte array	DER-encoding. An application can set or change the value at any time.
CKA_ID	Byte array	Default is empty. An application can set or change the value at any time.
CKA_KEY_TYPE ^{1, 5}	CK_KEY_TYPE	Type of key. CKK_EC, CKK_RSA, CKK_DSA, and CKK_DH only. An application can specify the value when the object is created (or generated) only.
CKA_START_DATE ⁸	CK_DATE	Start date for the key. Default empty. An application can set or change the value at any time.
CKA_END_DATE ⁸	CK_DATE	End date for the key. Default empty. An application can set or change the value at any time.
CKA_DERIVE ⁸	CK_BBOOL	TRUE if key supports key derivation (if other keys can be derived from this one). Default is TRUE. An application can set or change the value at any time.
CKA_LOCAL ^{2, 4, 6}	CK_BBOOL	TRUE only if key was generated locally. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_KEY_GEN_MECHANISM ^{2, 4, 6}	CK_MECHANISM_TYPE	Identifier of the mechanism used to generate the key material. Always CK_UNAVAILABLE_INFORMATION. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_DECRYPT ⁸	CK_BBOOL	TRUE if key supports decryption. ⁹ Default is TRUE. An application can set or change the value at any time.

Table 16. Private key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_SIGN ⁸	CK_BBOOL	TRUE if key supports signatures where the signature is an appendix to the data. ⁹ Default is TRUE. An application can set or change the value at any time.
CKA_SIGN_RECOVER ⁸	CK_BBOOL	TRUE if key supports signatures where the data can be recovered from the signature. ⁹ Default is TRUE. An application can set or change the value at any time.
CKA_UNWRAP ⁸	CK_BBOOL	TRUE if key supports unwrapping (can be used to unwrap other keys). ⁹ Default is TRUE. An application can set or change the value at any time.
CKA_EXTRACTABLE ⁸	CK_BBOOL	TRUE if key is extractable. Default is TRUE. An application can set or change the value, as per PKCS #11 restrictions. Caller can change from TRUE to FALSE only.
CKA_SENSITIVE ⁸	CK_BBOOL	TRUE if key is sensitive. Default is FALSE. An application can set or change the value, as per PKCS #11 restrictions. Caller can change from FALSE to TRUE only.
CKA_ALWAYS_SENSITIVE ^{2,4,6}	CK_BBOOL	TRUE if key has always had the CKA_SENSITIVE attribute set to TRUE. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_NEVER_EXTRACTABLE ^{2,4,6}	CK_BBOOL	TRUE if key has never had the CKA_EXTRACTABLE attribute set to TRUE. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_APPLICATION	Printable EBCDIC string	Description of the application that created the object. Default is empty. The string is assumed to come from the IBM1047 code page. An application can specify the value when the object is created (or generated) only.
CKA_IBM_FIPS140 (vendor specific attribute 0x80000005)	CK_BBOOL	TRUE if the key must only be used in a FIPS 140-2 compliant fashion. The default value is FALSE. An application can specify the value when the object is created (or generated) only.
CKA_WRAP_WITH_TRUSTED ¹²	CK_BBOOL	CK_TRUE if the key can only be wrapped with a wrapping key that has CKA_TRUSTED set to CK_TRUE. Default is CK_FALSE

Table 16. Private key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes								
CKA_IBM_SECURE ^{6, 11, 12, 13} (vendor specific attribute - 0x80000006)	CK_BBOOL	<p>CK_TRUE if the key is an Enterprise PKCS #11 coprocessor secure key.</p> <p>A secure key may be requested by setting this attribute CK_TRUE during key creation or generation.</p> <p>For key-pair generation, if set CK_TRUE by the caller or ICSF, the matching public key will also be a secure key. If set CK_FALSE or not specified at all, ICSF will determine the security.</p> <p>When this is set TRUE by the caller or ICSF, the key is treated as sensitive (CKA_SENSITIVE is set TRUE).</p> <p>May be changed from FALSE to TRUE during C_CopyObject.</p>								
CKA_IBM_ALWAYS_SECURE ^{2, 4, 6} (vendor specific attribute - 0x80000008)	CK_BBOOL	<p>CK_TRUE if key has always had the CKA_IBM_SECURE attribute set to CK_TRUE. Only applicable to secure keys. This attribute will not have a value for clear keys.</p>								
CKA_IBM_CARD_COMPLIANCE ^{2, 4, 6, 15} (vendor specific attribute - 0x80000007)	CK_ULONG	<p>A bit mask field indicating the compliance mode of the Enterprise PKCS #11 coprocessor at the time the secure key was created:</p> <table border="0"> <tr> <td>FIPS2009</td> <td>1</td> </tr> <tr> <td>BSI2009</td> <td>2</td> </tr> <tr> <td>FIPS2011</td> <td>4</td> </tr> <tr> <td>BSI2011</td> <td>8</td> </tr> </table> <p>Only applicable to secure keys. This attribute will not have a value for clear keys. When changing the value, the new value must specify the current mode or modes and any additional new modes. Changing the value to 0 results in the compliance mode being reset to the current compliance mode of the coprocessor.</p>	FIPS2009	1	BSI2009	2	FIPS2011	4	BSI2011	8
FIPS2009	1									
BSI2009	2									
FIPS2011	4									
BSI2011	8									
CKA_IBM_ATTRBOUND ^{2, 6} (vendor specific attribute - 0x80010004)	CK_BBOOL	<p>CK_TRUE has the following meaning:</p> <ul style="list-style-type: none"> The key must only be exported with its boolean usage attributes. The key may be used as a signing key for attribute bound wrap. The key may be used as an unwrapping key for attribute bound unwrap. <p>May only be set CK_TRUE during key generation. The default value is CK_FALSE.</p> <p>When this is set TRUE by the caller, the key will be a secure key (CKA_IBM_SECURE is set TRUE).</p> <p>Only applicable to secure keys. This attribute will not have a value for clear keys.</p>								

Table 16. Private key object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_IBM_ICSF_HANDLE (vendor specific attribute - 0x80010009)	Printable EBCDIC string	44-character ICSF record locator value. This attribute may be retrieved via C_GetAttributeValue. It is not valid in the attribute list of any other function or callable service.

Table 17. RSA private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_MODULUS ^{1, 4, 6}	Big integer	Modulus n An application can specify the value when the object is created (or generated) only.
CKA_PUBLIC_EXPONENT ^{4, 6}	Big integer	Public exponent e An application can specify the value when the object is created (or generated) only.
CKA_PRIVATE_EXPONENT ^{1, 4, 6, 7}	Big integer	Private exponent d Sensitive key part. An application can specify the value when the object is created (or generated) only.
CKA_PRIME_1 ^{4, 6, 7}	Big integer	Prime p Sensitive key part. An application can specify the value when the object is created (or generated) only.
CKA_PRIME_2 ^{4, 6, 7}	Big integer	Prime q Sensitive key part. An application can specify the value when the object is created (or generated) only.
CKA_EXPONENT_1 ^{4, 6, 7}	Big integer	Private exponent d modulo $p-1$ Sensitive key part. An application can specify the value when the object is created (or generated) only.
CKA_EXPONENT_2 ^{4, 6, 7}	Big integer	Private exponent d modulo $q-1$ Sensitive key part. An application can specify the value when the object is created (or generated) only.
CKA_COEFFICIENT ^{4, 6, 7}	Big integer	CRT coefficient $q^{-1} \bmod p$ Sensitive key part. An application can specify the value when the object is created (or generated) only.

Table 18. DSA private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_PRIME ^{1,4,6}	Big integer	Prime p (512 to 2048 bits in steps of 64 bits)
CKA_SUBPRIME ^{1,4,6}	Big integer	Subprime q (160 bits for $p \leq 1024$ bits, 224 bits or 256 bits for $p > 1024$ bits)
CKA_BASE ^{1,4,6}	Big integer	Base g
CKA_VALUE ^{1,4,6,7}	Big integer	Private value x

Table 19. Diffie-Hellman private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_PRIME ^{1,4,6}	Big integer	Prime p (512 to 2048 bits in steps of 64 bits)
CKA_BASE ^{1,4,6}	Big integer	Base g
CKA_VALUE ^{1,4,6,7}	Big integer	Private value x
CKA_VALUE_BITS ^{2,6}	CK_ULONG	Length in bits of private value x . For non-FIPS or when prime bit size = 1024, the default is 160. For FIPS prime bit size = 2048, the default is 256

Table 20. Elliptic Curve private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_EC_PARAMS ^{1,4,6} (CKA_ECDSA_PARAMS)	Byte Array	DER-encoding of an ANSI X9.62 Parameters value
CKA_VALUE ^{1,4,6,7}	Big integer	ANSI X9.62 private value d

Table 21. Domain parameter object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_CLASS ¹	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created (or generated) only.
CKA_TOKEN ¹¹	CK_BBOOL	Default value on create is FALSE. Object hardened to the TKDS if TRUE.
CKA_PRIVATE ¹¹	CK_BBOOL	Default value on create is FALSE
CKA_MODIFIABLE ¹¹	CK_BBOOL	Default value is TRUE
CKA_LABEL	Printable EBCDIC string	Application specific nickname. Limit to 32 chars. Default is empty. The string is assumed to come from the IBM1047 code page.
CKA_KEY_TYPE ¹	CK_KEY_TYPE	Type of key the domain parameters can be used to generate. CKK_DSA and CKK_DH only in this release
CKA_LOCAL ^{2,4}	CK_BBOOL	TRUE only if the parameters were generated locally
CKA_APPLICATION	Printable EBCDIC string	Description of the application that created the object. Default is empty. The string is assumed to come from the IBM1047 code page.

Table 21. Domain parameter object attributes that ICSF supports (continued). For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_IBM_ICSF_HANDLE (vendor specific attribute - 0x80010009)	Printable EBCDIC string	44-character ICSF record locator value. This attribute may be retrieved via C_GetAttributeValue. It is not valid in the attribute list of any other function or callable service.

Table 22. DSA domain parameter object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_PRIME ^{1,4}	Big integer	Prime p (512 to 2048 bits in steps of 64 bits)
CKA_SUBPRIME ^{1,4}	Big integer	Subprime q (160 bits for $p \leq 1024$ bits, 224 bits or 256 bits for $p > 1024$ bits)
CKA_BASE ^{1,4}	Big integer	Base g
CKA_PRIME_BITS ^{2,3}	CK_ULONG	Length of the prime value

Table 23. Diffie-Hellman domain parameter object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 30.

Attribute	Data type	Notes
CKA_PRIME ^{1,4}	Big integer	Prime p (512 to 2048 bits in steps of 64 bits)
CKA_BASE ^{1,4}	Big integer	Base g
CKA_PRIME_BITS ^{2,3}	CK_ULONG	Length of the prime value

Library, slot, and token information

PKCS #11 maintains information about the library code, slots, and tokens, which can be set and queried by calling the library functions. For z/OS, this information is as follows:

CK_INFO - Returned by C_GetInfo. not modifiable by applications.

- cryptokiVersion - 2.20
- manufacturerID - "IBM Corp. "
- libraryDescription - "z/OS PKCS11 library "
- libraryVersion - 7.70

CK_SLOT_INFO - Returned by C_GetSlotInfo. not modifiable by applications

- slotDescription - "z/OS PKCS11 - virtual smart card "
- manufacturerID - "IBM Corp. "
- flags - for any slot returned by C_GetSlotList the following flags are set:
 - CKF_TOKEN_PRESENT=ON
 - CKF_REMOVABLE_DEVICE=ON
 - CKF_HW_SLOT=OFF
- hardwareVersion - 7.70
- firmwareVersion - 7.70

CK_TOKEN_INFO - Set by C_InitToken, Returned by C_GetTokenInfo

- label - As specified

- manufacturerID - "z/OS PKCS11 API" (Might be set to other values if the token was initialized outside of the C API.)
- model - "HCR7770" (coincides with the release that the token was created) (Might be set to other values if the token was initialized outside of the C API.)
- serialNumber - "0" (Might be set to other values if the token was initialized outside of the C API.)
- flags - the following flags are set ON for any initialized token. All others are OFF:
 - CKF_RNG
 - CKF_PROTECTED_AUTHENTICATION_PATH
 - CKF_TOKEN_INITIALIZED
 - CKF_USER_PIN_INITIALIZED
- ulMaxSessionCount - CK_UNAVAILABLE_INFORMATION
- ulSessionCount - CK_UNAVAILABLE_INFORMATION
- ulMaxRwSessionCount - CK_UNAVAILABLE_INFORMATION
- ulRwSessionCount - CK_UNAVAILABLE_INFORMATION
- ulMaxPinLen - CK_UNAVAILABLE_INFORMATION
- ulMinPinLen - 0
- ulTotalPublicMemory - CK_UNAVAILABLE_INFORMATION
- ulFreePublicMemory - CK_UNAVAILABLE_INFORMATION
- ulTotalPrivateMemory - CK_UNAVAILABLE_INFORMATION
- ulFreePrivateMemory - CK_UNAVAILABLE_INFORMATION
- hardwareVersion - 7.70
- firmwareVersion - 7.70
- utcTime - GMT date and time that the token was last updated

CK_SESSION_INFO - Returned by C_GetSessionInfo

- slotId - The slot in question
- state - CK_UNAVAILABLE_INFORMATION
- flags - As defined by the PKCS #11 specification
- ulDeviceError - A mapping of the last failing ICSF return and reason code values related to this session. For more information see "Function return codes" on page 62.

Functions supported

ICSF supports a subset of the standard PKCS #11 functions, and several non-standard functions.

Standard functions supported

Table 24 lists the standard PKCS #11 functions that ICSF supports. Any function not listed is not supported and returns the CKR_FUNCTION_NOT_SUPPORTED return code.

Table 24. Standard PKCS #11 functions that ICSF supports. The Standard PKCS #11 functions that ICSF supports table is presented by type of function.

Function	Usage notes
General purpose functions:	

Table 24. Standard PKCS #11 functions that ICSF supports (continued). The Standard PKCS #11 functions that ICSF supports table is presented by type of function.

Function	Usage notes
C_Initialize()	<ul style="list-style-type: none"> The library always uses OS locking for thread serialization. Therefore, if C_Initialize is called with the CreateMutex, DestroyMutex, LockMutex, and UnlockMutex function pointer arguments set and the CKF_OS_LOCKING_OK flag is not set, C_Initialize fails and returns the value CKR_CANT_LOCK. When C_Initialize is called, the application-specific set of (virtual) slot IDs is allocated, one for each preexisting token that the application is authorized to use. (See the descriptions of C_GetSlotList and C_WaitForSlotEvent for information on how this set can increase in size.) The one exception to this occurs when C_Initialize is called by a child process after fork. If the PKCS #11 environment is inherited by the child process, the slot list and token state is not refreshed.
C_Finalize()	dlclose() cannot be used as an implicit C_Finalize(). If an application uses dlclose() without calling C_Finalize(), and reinitializes PKCS #11, a subsequent call to C_Initialize() will result in error CKR_FUNCTION_FAILED being returned.
C_GetInfo()	
C_GetFunctionList()	
Slot and token management functions:	
C_GetSlotList()	<ul style="list-style-type: none"> If the pSlotList argument is NULL, this function returns only the number of allocated slots. In the process of returning this number C_GetSlotList searches for new tokens to which the application has access. If new tokens are found, slot IDs are allocated for them. This search is only performed if at least 5 seconds has passed since the last search was made. If the pSlotList argument is non-NULL, this function returns the current list of virtual slot IDs. No attempt is made to discover new tokens created by other applications. The tokenPresent argument flag is meaningless as all allocated slots have a token present.
C_GetSlotInfo()	
C_GetTokenInfo()	
C_WaitForSlotEvent()	<ul style="list-style-type: none"> This function is used to dynamically allocate an additional slot in order to create a new token. There are no other slot events. The newly allocated slot ID is returned as the pSlot argument. The CKF_DONT_BLOCK argument flag is meaningless because this function never blocks. The dynamic slot allocation occurs synchronously.

Table 24. Standard PKCS #11 functions that ICSF supports (continued). The Standard PKCS #11 functions that ICSF supports table is presented by type of function.

Function	Usage notes
C_GetMechanismList()	The list of functions returned reflects the capabilities of the current cryptographic hardware configuration. Note: The loss or addition of hardware on the fly is not detected or reflected. (For example, on a z9-109, if the only CEX2C present is deactivated, this function still returns the mechanisms that require an active CEX2C to function.)
C_GetMechanismInfo()	The output of this function reflects the capabilities of the current cryptographic hardware configuration.
C_InitToken()	Tokens are protected by the security manager through profiles in the CRYPTOZ class. PINs are not used. The pPin and ulPinLen arguments are ignored.
C_InitPIN()	Tokens are protected by the security manager through profiles in the CRYPTOZ class. PINs are not used. This function performs no operation and always returns CKR_OK.
C_SetPIN()	Tokens are protected by the security manager through profiles in the CRYPTOZ class. PINs are not used. This function performs no operation and always returns CKR_OK.
Session management functions:	
C_OpenSession()	The Notify and pApplication arguments are ignored.
C_CloseSession()	
C_CloseAllSessions()	
C_GetSessionInfo()	The state field returned is meaningless. It is always set to CK_UNAVAILABLE_INFORMATION.
C_GetOperationState()	Returns CKR_STATE_UNSAVEABLE if a find is active or more than one cryptographic operation is active.
C_SetOperationState()	
C_Login()	Tokens are protected by the security manager through profiles in the CRYPTOZ class. Applications are always logged in to the security manager. PINs are not used. This function has no effect on the session state and always returns CKR_OK.
C_Logout()	Tokens are protected by the security manager through profiles in the CRYPTOZ class. Applications are always logged in to the security manager. PINs are not used. This function has no effect on the session state and always returns CKR_OK.
Object management functions:	
C_CreateObject()	
C_CopyObject()	
C_DestroyObject()	
C_GetObjectSize()	
C_GetAttributeValue()	
C_SetAttributeValue()	

Table 24. Standard PKCS #11 functions that ICSF supports (continued). The Standard PKCS #11 functions that ICSF supports table is presented by type of function.

Function	Usage notes
C_FindObjectsInit()	
C_FindObjects()	Sensitive attributes cannot be used as search criteria when the object is marked sensitive or not exportable. Doing so results in no match found.
C_FindObjectsFinal()	
Encryption functions:	
C_EncryptInit()	The following mechanisms are supported: <ul style="list-style-type: none"> • CKM_DES_ECB • CKM_DES_CBC • CKM_DES_CBC_PAD • CKM_DES3_ECB • CKM_DES3_CBC • CKM_DES3_CBC_PAD • CKM_RSA_PKCS • CKM_RSA_X_509 • CKM_AES_CBC • CKM_AES_ECB • CKM_AES_CBC_PAD • CKM_AES_GCM (Limited to single part encryption only and for no more than 1048576 bytes of clear text.) • CKM_BLOWFISH_CBC • CKM_RC4 • A secure key may not be used for processing rule GCM or GCMIVGEN.
C_Encrypt()	
C_EncryptUpdate()	Multiple-part encryption is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms
C_EncryptFinal()	Multiple-part encryption is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
Decryption functions:	
C_DecryptInit()	The following mechanisms are supported: <ul style="list-style-type: none"> • CKM_DES_ECB • CKM_DES_CBC • CKM_DES_CBC_PAD • CKM_DES3_ECB • CKM_DES3_CBC • CKM_DES3_CBC_PAD • CKM_RSA_PKCS • CKM_RSA_X_509 • CKM_AES_CBC • CKM_AES_ECB • CKM_AES_CBC_PAD • CKM_AES_GCM (Limited to single part decryption only and for no more than 1048576 bytes of clear text.) • CKM_BLOWFISH_CBC • CKM_RC4 • A secure key may not be used for processing rule GCM.
C_Decrypt()	
C_DecryptUpdate()	Multiple-part decryption is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.

Table 24. Standard PKCS #11 functions that ICSF supports (continued). The Standard PKCS #11 functions that ICSF supports table is presented by type of function.

Function	Usage notes
C_DecryptFinal()	Multiple-part decryption is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
Message digesting functions:	
C_DigestInit()	The following mechanisms are supported: <ul style="list-style-type: none"> • CKM_MD2 • CKM_MD5 • CKM_SHA_1 • CKM_SHA224 • CKM_SHA256 • CKM_SHA384 • CKM_SHA512 • CKM_RIPEMD160
C_Digest()	
C_DigestUpdate()	
C_DigestFinal()	
Signing and message authentication coding (MACing) functions:	
C_SignInit()	The following mechanisms are supported: <ul style="list-style-type: none"> • CKM_RSA_X_509 • CKM_RSA_PKCS • CKM_MD5_RSA_PKCS • CKM_SHA1_RSA_PKCS • CKM_SHA224_RSA_PKCS • CKM_SHA256_RSA_PKCS • CKM_SHA384_RSA_PKCS • CKM_SHA512_RSA_PKCS • CKM_DSA • CKM_DSA_SHA1 • CKM_MD5_HMAC • CKM_SHA_1_HMAC • CKM_SHA224_HMAC • CKM_SHA256_HMAC • CKM_SHA384_HMAC • CKM_SHA512_HMAC • CKM_SSL3_MD5_MAC • CKM_SSL3_SHA1_MAC • CKM_MD2_RSA_PKCS • CKM_ECDSA • CKM_ECDSA_SHA1 • CKM_RSA_PKCS_PSS • CKM_SHA1_RSA_PKCS_PSS • CKM_SHA224_RSA_PKCS_PSS • CKM_SHA256_RSA_PKCS_PSS • CKM_SHA384_RSA_PKCS_PSS • CKM_SHA512_RSA_PKCS_PSS
C_Sign()	
C_SignUpdate()	Multiple-part signature is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
C_SignFinal()	Multiple-part signature is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.

Table 24. Standard PKCS #11 functions that ICSF supports (continued). The Standard PKCS #11 functions that ICSF supports table is presented by type of function.

Function	Usage notes
Functions for verifying signatures and message authentication codes (MACs):	
C_VerifyInit()	<p>The following mechanisms are supported:</p> <ul style="list-style-type: none"> • CKM_RSA_X_509 • CKM_RSA_PKCS • CKM_MD5_RSA_PKCS • CKM_SHA1_RSA_PKCS • CKM_SHA224_RSA_PKCS • CKM_SHA256_RSA_PKCS • CKM_SHA384_RSA_PKCS • CKM_SHA512_RSA_PKCS • CKM_DSA • CKM_DSA_SHA1 • CKM_MD5_HMAC • CKM_SHA_1_HMAC • CKM_SHA224_HMAC • CKM_SHA256_HMAC • CKM_SHA384_HMAC • CKM_SHA512_HMAC • CKM_SSL3_MD5_MAC • CKM_SSL3_SHA1_MAC • CKM_MD2_RSA_PKCS • CKM_ECDSA • CKM_ECDSA_SHA1 • CKM_RSA_PKCS_PSS • CKM_SHA1_RSA_PKCS_PSS • CKM_SHA224_RSA_PKCS_PSS • CKM_SHA256_RSA_PKCS_PSS • CKM_SHA384_RSA_PKCS_PSS • CKM_SHA512_RSA_PKCS_PSS
C_Verify()	
C_VerifyUpdate()	Multiple-part verify is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
C_VerifyFinal()	Multiple-part verify is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
Key management functions:	
C_DeriveKey()	<p>The following mechanisms are supported:</p> <ul style="list-style-type: none"> • CKM_DH_PKCS_DERIVE • CKM_SSL3_MASTER_KEY_DERIVE • CKM_SSL3_MASTER_KEY_DERIVE_DH • CKM_SSL3_KEY_AND_MAC_DERIVE • CKM_TLS_MASTER_KEY_DERIVE • CKM_TLS_MASTER_KEY_DERIVE_DH • CKM_TLS_KEY_AND_MAC_DERIVE • CKM_TLS_PRF (It is the caller's responsibility to supply an ASCII value for the seed) • CKM_ECDH1_DERIVE <p>The service does not support the derivation of secure keys. For mechanisms CKM_ECDH1_DERIVE and CKM_DH_PKCS_DERIVE only, the input base key may be a secure key.</p>

|
|
|
|

Table 24. Standard PKCS #11 functions that ICSF supports (continued). The Standard PKCS #11 functions that ICSF supports table is presented by type of function.

Function	Usage notes
C_GenerateKey()	<p>The following mechanisms are supported:</p> <ul style="list-style-type: none"> • CKM_DES_KEY_GEN • CKM_DES2_KEY_GEN • CKM_DES3_KEY_GEN • CKM_PBE_SHA1_DES3_EDE_CBC • CKM_AES_KEY_GEN • CKM_DSA_PARAMETER_GEN • CKM_DH_PKCS_PARAMETER_GEN • CKM_BLOWFISH_KEY_GEN • CKM_RC4_KEY_GEN • CKM_GENERIC_SECRET_KEY_GEN • CKM_SSL3_PRE_MASTER_KEY_GEN • CKM_TLS_PRE_MASTER_KEY_GEN
C_GenerateKeyPair()	<p>The following mechanisms are supported:</p> <ul style="list-style-type: none"> • CKM_RSA_PKCS_KEY_PAIR_GEN • CKM_DSA_KEY_PAIR_GEN • CKM_DH_PKCS_KEY_PAIR_GEN • CKM_EC_KEY_PAIR_GEN
C_WrapKey()	<p>The following mechanisms are supported for wrapping secret keys:</p> <ul style="list-style-type: none"> • CKM_RSA_PKCS • CKM_DES_CBC_PAD • CKM_DES3_CBC_PAD • CKM_AES_CBC_PAD • CKM_IBM_ATTRIBUTEBOUND_WRAP <p>The following mechanisms are supported for wrapping private keys:</p> <ul style="list-style-type: none"> • CKM_DES_CBC_PAD • CKM_DES3_CBC_PAD • CKM_AES_CBC_PAD • CKM_IBM_ATTRIBUTEBOUND_WRAP <p>Clear keys may not be used to wrap secure keys and secure keys may not be used to wrap clear keys. One exception: Clear RSA public keys may be used to perform a non-attribute bound wrap of secure secret keys.</p>
C_UnwrapKey()	<p>The following mechanisms are supported for unwrapping secret keys:</p> <ul style="list-style-type: none"> • CKM_RSA_PKCS • CKM_DES_CBC_PAD • CKM_DES3_CBC_PAD • CKM_AES_CBC_PAD • CKM_IBM_ATTRIBUTEBOUND_WRAP <p>The following mechanisms are supported for unwrapping private keys:</p> <ul style="list-style-type: none"> • CKM_DES_CBC_PAD • CKM_DES3_CBC_PAD • CKM_AES_CBC_PAD • CKM_IBM_ATTRIBUTEBOUND_WRAP

Table 24. Standard PKCS #11 functions that ICSF supports (continued). The Standard PKCS #11 functions that ICSF supports table is presented by type of function.

Function	Usage notes
Random number generation functions:	
C_SeedRandom()	This function always returns the value CKR_RANDOM_SEED_NOT_SUPPORTED because the z/OS hardware random number generator is self-seeding.
C_GenerateRandom()	

Non-standard functions supported

The following non-standard functions are also supported:

- CSN_FindALLObjects()

Because they are non-standard, they do not appear in the PKCS #11 CK_FUNCTION_LIST structure returned by C_GetFunctionList(). To invoke these functions, the caller must either locate the desired function in the main DLL using dlsym(), or link the application program with the main DLL's sidedeck.

CSN_FindALLObjects()

CSN_FindALLObjects() is identical to C_FindObjects(), except that it uses the ALL rule array keyword when invoking the ICSF CSFPTRL callable service. This can result in CSN_FindALLObjects() returning handles to private objects even if the caller has insufficient SAF authority to view such objects. CSN_FindALLObjects() returns a private key handle (and C_FindObjects does not) when the following conditions are all met:

1. The private object matches the search criteria.
2. No sensitive attributes were specified in the search criteria. The sensitive values for this service are:
 - For a secret key object: CKA_VALUE
 - For Diffie Hellman, DSA, and Elliptic Curve private key objects: CKA_VALUE
 - For an RSA private key object: CKA_PRIVATE_EXPONENT, CKA_PRIME_1, CKA_PRIME_2, CKA_EXPONENT_1, CKA_EXPONENT_2, CKA_COEFFICIENT
3. The caller has only Weak SO or SO R/W permission to the token.

Table 25. Syntax of the CK_RV CSN_FindALLObjects() function

CSN_FindALLObjects (
CK_SESSION_HANDLE	hSession,
CK_OBJECT_HANDLE_PTR	phObject,
CK_ULONG	ulMaxObjectCount
CK_ULONG_PTR	pu1ObjectCount
);	

For more information about CSFPTRL processing with respect to the ALL rule array keyword, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

Non-standard mechanisms supported

The CKM_IBM_ATTRIBUTEBOUND_WRAP function is for wrapping and unwrapping private and secret keys in an IBM proprietary format, where the key's boolean usage attributes are included with the key material in the cryptogram. In

addition to the format, the package is also signed, which means that the unwrapping party must have the matching verifying key.

CKM_IBM_ATTRIBUTEBOUND_WRAP has the following restrictions:

- Only works with secure keys (CKA_IBM_SECURE=TRUE)
- All keys involved (target, wrapping/unwrapping, signature/verification) must be attribute bound keys (CKA_IBM_ATTRBOUND=TRUE), otherwise
 - For the target key on C_WrapKey, CKR_KEY_NOT_WRAPPABLE is returned.
 - For the wrapping/unwrapping, signature/verification keys, CKR_KEY_FUNCTION_NOT_PERMITTED is returned.
- An attribute template, if specified, may not contain key usage attributes. If such a template is specified, CKR_TEMPLATE_INCONSISTENT is returned.
- On C_WrapKey, the signing private key must be capable of signing (CKA_SIGN=TRUE), otherwise CKR_KEY_FUNCTION_NOT_PERMITTED is returned.
- On C_UnwrapKey, the verification public key must be capable of verifying (CKA_VERIFY=TRUE), otherwise CKR_KEY_FUNCTION_NOT_PERMITTED is returned.

The CKM_IBM_ATTRIBUTEBOUND_WRAP function takes a parameter, used to specify the signature or verification key handle.

Table 26. Syntax of the CKM_IBM_ATTRIBUTEBOUND_WRAP function

```
typedef struct CKM_IBM_ATTRIBUTEBOUND_WRAP {  
    CK_OBJECT_HANDLE hSignVerifyKey;  
}CK_IBM_ATTRBOUND_WRAP_PARAMS;
```

Note: If attribute bound wrapping is used to import a key, the resulting key object may have certain usage attribute flags set FALSE even though they were set TRUE on the source key. This happens for the following key types:

CKK_GENERIC_SECRET secret keys - CKA_ENCRYPT, CKA_DECRYPT, CKA_WRAP, and CKA_UNWRAP will be set FALSE

CKK_DSA, CKK_EC or CKK_DH private keys - CKA_SIGN_RECOVER, CKA_DECRYPT, and CKA_UNWRAP will be set FALSE

This behavior, though inconsistent, does not cause a loss of function as these key types are not physically capable of performing the negated operations.

Enterprise PKCS #11 coprocessors

Key algorithms/usages that are unsupported or disallowed by the Enterprise PKCS #11 coprocessors

The following table lists the key algorithms/usages that are not supported by the Enterprise PKCS #11 coprocessors or disallowed due to FIPS restrictions that are always enforced. The results of requesting an unsupported algorithm depend on what is being requested. All these results assume the system is properly configured to use secure PKCS #11. Improper configuration would result in different errors:

1. Key generation or creation – Explicitly requesting the generation or creation of an unsupported/disallowed secure key type results in CKR_TEMPLATE_INCONSISTENT being returned.
2. Key derivation – Explicitly requesting the derivation of a secure key using a clear base key results CKR_TEMPLATE_INCONSISTENT being returned.

Attempting key derivation using a secure base key results in CKR_IBM_CLEAR_KEY_REQ being returned.

3. Standard unwrap key – The target key always has the security of the unwrapping key. Specifying the CKA_IBM_SECURE attribute in the unwrap template results in CKR_ATTRIBUTE_READ_ONLY being returned. Requesting the unwrapping of an unsupported/disallowed key type using a secure unwrapping key results in CKR_IBM_CLEAR_KEY_REQ being returned.
4. Otherwise, requesting an unsupported/disallowed algorithm using a secure key results in CKR_IBM_CLEAR_KEY_REQ being returned.

Table 27. List of algorithms/uses not supported/disallowed by Enterprise PKCS #11 coprocessors

Algorithm	PKCS #11 Mechanisms or key types	Comments
MD2	CKM_MD2_RSA_PKCS	Secure private key use for signing disallowed
MD5	CKM_MD5_RSA_PKCS, CKM_MD5_HMAC	Secure private or secret key use for signing disallowed
SSL3	CKM_SSL3_MD5_MAC, CKM_SSL3_SHA1_MAC, CKM_SSL3_MASTER_KEY_DERIVE, CKM_SSL3_MASTER_KEY_DERIVE_DH, CKM_SSL3_KEY_AND_MAC_DERIVE	
TLS	CKM_TLS_MASTER_KEY_DERIVE, CKM_TLS_MASTER_KEY_DERIVE_DH, CKM_TLS_KEY_AND_MAC_DERIVE	
Diffie Hellman	CKK_DH keys	Prime size less than 1024 bits
DSA	CKK_DSA keys	Combinations other than the following are not supported: <ul style="list-style-type: none"> • Prime size = 1024 bits, subprime size = 160 bits • Prime size = 2048 bits, subprime size = 224 bits or 256 bits
Single DES	CKK_DES keys	
Triple DES	CKK_DES2 keys	
Blowfish	CKK_BLOWFISH keys	
RC4	CKK_RC4 keys	
RSA	CKK_RSA CKM_RSA_PKCS_KEY_PAIR_GEN CKM_RSA_X_509	Key sizes less than 1024 bits Key sizes that are less than 1024 bits or not a multiple of 256 bits or public key exponents less than 0x010001 Secure private key use for signing/decryption disallowed
HMAC	CKK_GENERIC_SECRET CKM_SHA_1_HMAC, CKM_SHA224_HMAC, CKM_SHA256_HMAC, CKM_SHA384_HMAC, CKM_SHA512_HMAC	Key sizes less than 10 bytes Base key sizes less than ½ the output size are not supported.
AES GCM	CKM_AES_GCM	

PKCS #11 Coprocessor Access Control Points

The following table lists the Access Control Points that are available on the Enterprise PKCS #11 coprocessors and the PKCS #11 mechanisms or functions that would be disabled for secure keys if the control point is deactivated. A new or a zeroized Enterprise PKCS #11 coprocessor (or domain) comes with an initial set of Access Control Points (ACPs) that are enabled by default. All other ACPs, representing potential future support, are left disabled. When a firmware upgrade is applied to an existing Enterprise PKCS #11 coprocessor, the upgrade may introduce new ACPs. The firmware upgrade does not retroactively enable these ACPs, so they are disabled by default. These ACPs must be enabled via the TKE (or subsequent zeroize) in order to utilize the new support they govern.

See the Enabling Access Control Points for PKCS #11 coprocessor firmware section in the Migration topic of the *z/OS Cryptographic Services ICSF System Programmer's Guide* for the list of default ACPs and those ACPs that need to be enabled via TKE for PKCS #11 coprocessor firmware upgrades.

The following table lists the Access Control Points that are available on the Enterprise PKCS #11 coprocessors and the PKCS #11 mechanisms or functions that would be disabled for secure keys if the control point is deactivated.

Table 28. PKCS #11 Access Control Points. The information is presented by Access Control Points name or group.

Access Control Point name or group	Mechanism/Function requiring enablement	Number
<i>Control Point Management</i>		
Allow addition (activation) of Control Points	not applicable	0
Allow removal (deactivation) of Control Points	not applicable	1
<i>Cryptographic Operations</i>		
Sign with private keys	Sign using CKK_RSA, CKK_DSA, of CKK_ECDSA keys.	2
Sign with HMAC or CMAC	Sign using CKM_SHA_1_HMAC, CKM_SHA224_HMAC, CKM_SHA256_HMAC, CKM_SHA384_HMAC, or CKM_SHA512_HMAC.	3
Verify with HMAC or CMAC	Verify using CKM_SHA_1_HMAC, CKM_SHA224_HMAC, CKM_SHA256_HMAC, CKM_SHA384_HMAC, or CKM_SHA512_HMAC.	4
Encrypt with symmetric keys	Encrypt with CKK_DES3 of CKK_AES keys. Create Object or Copy Object where source is a clear key.	5
Decrypt with private keys	Decrypt with CKK_RSA keys.	6
Decrypt with symmetric keys	Decrypt with CKK_DES3 of CKK_AES keys.	7
Key export with public keys	Wrap Key using a CKK_RSA wrapping key.	8
Key export with symmetric keys	Wrap Key using a CKK_DES3 or CKK_AES wrapping key.	9
Key import with private keys	Unwrap Key using a CKK_RSA unwrapping key.	10

Table 28. PKCS #11 Access Control Points (continued). The information is presented by Access Control Points name or group.

Access Control Point name or group	Mechanism/Function requiring enablement	Number
Key import with symmetric keys	Unwrap Key using a CKK_DES3 or CKK_AES unwrapping key. Create Object or Copy Object where source is a clear key.	11
Generate asymmetric key pairs	Generate Key Pair for CKK_RSA, CKK_DSA, or CKK_ECDSA keys	12
Generate symmetric keys	Generate key for CKK_DES2 or CKK_AES keys	13
Allow key derivation	Derive key using a CKK_DH key	47
<i>Cryptographic Algorithms</i>		
RSA private-key use	Generate Key Pair for CKK_RSA Sign or Decrypt using a CKK_RSA key	30
DSA private-key use	Generate Key Pair for CKK_DSA Sign using a CKK_DSA key	31
EC private-key use	Generate Key Pair for CKK_EC Sign or Derive Key using a CKK_EC key	32
DH private-key use	Generate Key Pair for CKK_DH	46
Brainpool (E.U.) EC curves	Sign or Verify using the Brainpool curves	33
NIST/SECG EC curves	Sign or Verify using the NIST EC curves	34
Allow non-BSI algorithms (as of 2009)	not applicable	21
Allow non-FIPS-approved algorithms (as of 2011)	not applicable	35
Allow non-BSI algorithms (as of 2011)	not applicable	36
<i>Key Size</i>		
Allow 80 to 111-bit algorithms	Any use of CKK_GENERIC_SECRET keys smaller than 112 bits, or 160 or 192 bit CKK_ECDSA keys If in a BSI mode: Any use of CKK_DSA keys, or CKK_RSA keys smaller than 2432 bits If not in a BSI mode: Any use of CKK_DSA, or CKK_RSA keys smaller than 2048 bits	24
Allow 112 to 127-bit algorithms	Any use of 2048 bit CKK_DSA keys, CKK_GENERIC_SECRET keys larger than 111 bits but less than 128 bits, 224 bit CKK_ECDSA keys, or CKK_DES3 keys If in a BSI mode: Any use of CKK_RSA keys larger than 2431 bits but less than 3248 bits If not in a BSI mode: Any use of CKK_RSA keys larger than 2047 bits but less than 3072 bits	25

Table 28. PKCS #11 Access Control Points (continued). The information is presented by Access Control Points name or group.

Access Control Point name or group	Mechanism/Function requiring enablement	Number
Allow 128 to 191-bit algorithms	Any use of CKK_GENERIC_SECRET keys larger than 127 bits but less than 192 bits, 128 bit CKK_AES keys, or 256 bit CKK_ECDSA keys If in a BSI mode: Any use of CKK_RSA keys larger than 3247 bits If not in a BSI mode: Any use of CKK_RSA keys larger than 3071 bits	26
Allow 192 to 255-bit algorithms	Any use of CKK_GENERIC_SECRET keys larger than 191 bits, 192 bit CKK_AES keys or 384 bit CKK_ECDSA keys.	27
Allow 256-bit algorithms	Any coprocessor use other than random number generation.	28
Allow RSA public exponents below 0x10001	Generate Key or Generate Key Pair for CKK_RSA where the exponent is 3.	29
<i>Miscellaneous</i>		
Allow backend to save semi-retained keys	not applicable	14
Allow keywrap without attribute-binding	Wrap Key or Unwrap Key using CKM_RSA_PKCS, CKM_AES_CBC_PAD, or CKM_DES3_CBC_PAD Create Object or Copy Object where source is a clear key.	16
Allow changes to key objects (usage flags only)	Set Attribute Value or Copy Object where the key usage flags are modified	17
Allow mixing external seed to RNG	not applicable	18
Allow non-administrators to mark key objects TRUSTED	Set Attribute Value where CKA_TRUSTED is set TRUE	37
Do not double-check sign/decrypt operations	not applicable	38
Allow dual-function keys - key wrapping and data encryption	Generate Key or Generate Key Pair where CKA_WRAP / CKA_UNWRAP and CKA_ENCRYPT / CKA_DECRYPT combinations are requested (or defaulted) Wrap Key, Unwrap Key, Encrypt or Decrypt with a previously created key containing the above combination. Create Object or Copy Object where source is a clear key.	39
Allow dual-function keys - digital signature and data encryption	Create Object, Generate Key or Generate Key Pair where CKA_SIGN / CKA_VERIFY and CKA_ENCRYPT / CKA_DECRYPT combinations are requested (or defaulted) Sign, Verify, Encrypt or Decrypt with a previously created key containing the above combination	40

Table 28. PKCS #11 Access Control Points (continued). The information is presented by Access Control Points name or group.

Access Control Point name or group	Mechanism/Function requiring enablement	Number
Allow dual-function keys - key wrapping and digital signature	Create Object, Generate Key or Generate Key Pair where CKA_SIGN / CKA_VERIFY and CKA_WRAP / CKA_UNWRAP combinations are requested (or defaulted) Sign, Verify, Wrap Key or Unwrap Key with a previously created key containing the above combination	41
Allow non-administrators to mark public key objects ATTRBOUND	Create Object where CKA_IBM_ATTRBOUND is set TRUE	42
Allow clear passphrases for password-based-encryption	Generate Key using CKM_PBE_SHA1_DES3_EDE_CBC	43
Allow wrapping of stronger keys by weaker keys	Wrap Key where the to-be-wrapped key is stronger than the wrapping key.	44
Allow clear public keys as non-attribute bound wrapping keys	Wrap Key where the wrapping key is an CKK_RSA clear public key and the to-be-wrapped key is a secure CKK_DES3, CKK_AES, or CKK_GENERIC_SECRET key.	45

Standard compliance modes

Enterprise PKCS #11 coprocessors are designed to always operate in a FIPS compliant fashion. There are four optional compliances modes that a given domain can be in. The FIPS modes correspond to the FIPS 140-2 requirements as of 2009 and 2011. The BSI modes correspond to the BSI HSM protection profile, and German Bundesnetzagentur algorithms as of 2009 and 2011.

These compliance modes are not mutually exclusive and are set ON by disabling certain access control points:

1. **FIPS 2009** – Card adheres to FIPS restrictions that went into effect in 2009 – equivalent to ICSF’s FIPS140 mode. This is the default mode. A domain cannot be set to be less restrictive than this mode. This mode has the following policy/restrictions:
 - a. Algorithms and keys below 80-bit of strength are not permitted.
 - b. RSA private-keys may not be use without padding.
 - c. Newly generated asymmetric keys always undergo selftests.
 - d. The minimum keysize on HMAC is 1/2 the algorithm's output size.
 - e. Only FIPS-approved algorithms (as of 2009) are present.
2. **FIPS 2011** – Card adheres to FIPS restrictions that went into effect in 2011. (More restrictive than FIPS 2009.) The following access control points would need to be disabled:
 - a. Allow 80 to 111-bit algorithms.
 - b. Allow non-FIPS-approved algorithms (as of 2011).
 - c. Allow RSA public exponents below 0x10001.
3. **BSI 2009** – Card adheres to BSI restrictions that went into effect in 2009. The following access control point would need to be disabled:
 - a. Allow keywrap without attribute-binding.
 - b. Allow non-BSI-approved algorithms (as of 2009).

4. **BSI 2011** – Card adheres to BSI restrictions that went into effect in 2011. (More restrictive than BSI 2009.) The following access control points would need to be disabled:
 - a. Allow keywrap without attribute-binding.
 - b. Allow 80 to 111-bit algorithms.
 - c. Allow non-BSI-approved algorithms (as of 2011).

Whenever a secure key is created, the current compliance mode of the Enterprise PKCS #11 coprocessor is recorded inside the secure key. If the compliance mode of the coprocessor is subsequently changed, all previously created secure keys become unusable until their compliance modes are updated. See “Steps for running the pre-compiled version of testpkcs11” on page 66 for information on how to modify the compliance mode of a secure key using a sample program distributed by IBM. The compliance mode of a key may also be updated by using the PKCS #11 Token Browser ISPF panels. See the *z/OS Security Server RACF Security Administrator’s Guide* for more information on these panels.

Function return codes

In general, the PKCS #11 function return codes are defined in the PKCS #11 specification. However, the following function return codes have a meaning specific to z/OS:

CKR_TOKEN_NOT_PRESENT

ICSF is not running or the TKDS is not properly configured. Note that this return code has no relationship to the slot flag CKF_TOKEN_PRESENT.

CKR_TOKEN_NOT_RECOGNIZED

The caller is not authorized to perform the action requested.

CKR_MECHANISM_INVALID

The specified mechanism is either unknown or not supported by the current cryptographic hardware configuration.

CKR_DEVICE_REMOVED

The token no longer exists. When this error is detected, the token flags are cleared indicating that the token is no longer initialized. It can be re-initialized as a new token, if desired.

Other ICSF-related errors are returned as vendor-defined error codes (CKR_VENDOR_DEFINED). The ICSF return and reason codes are combined into the single return code as follows:

```
#define CKR_IBM_ICSF_ERROR 0xC0000000 /* High order byte mask indicating ICSF error */
#define CKR_IBM_ICSF_ERROR_RET 0x00FF0000 /* Second byte is the return code */
#define CKR_IBM_ICSF_ERROR_RSN 0x0000FFFF /* low order half word is reason code */
```

This mapping is also used to store the ICSF return and reason code values in the CK_SESSION_INFO ulDeviceError field.

The following constants are defined for select ICSF return reason codes:

```
/* ICSF not configured for FIPS mode OR system does not
support FIPS mode */
#define CKR_IBM_ICSF_NOT_FIPS_MODE 0xC0080BFD

/* Algorithm or key size is not valid in FIPS mode */
#define CKR_IBM_ICSF_NOT_VALID_FIPS 0xC0080BFE

/* FIPS known answer tests failed */
#define CKR_IBM_ICSF_FIPS_KAT_FAILED 0xC00C8D3C
```

```

/* Service or algorithm not available on current system */
#define CKR_IBM_ICSF_SERV_NOTAVAIL 0xC00C0008

/* A clear key is required for this operation, but a secure
key was supplied */
#define CKR_IBM_CLEAR_KEY_REQ 0xC0080C81

/* Clear key creation denied by policy */
#define CKR_IBM_DENIED_BY_POLICY 0xC0083E88

/* Key object in more restrictive compliance mode than
current setting of the Enterprise PKCS #11 coprocessors */
#define CKR_IBM_KEY_MODE_ERROR 0xC00C3200

```

Troubleshooting PKCS #11 applications

Note: The information and techniques described in this topic are for use primarily by IBM service personnel in determining the cause of a problem with the ICSF PKCS #11 C API.

You can capture trace data using environment variables. To do this, the trace environment variables CSN_PKCS11_TRACE and CSN_PKCS11_TRACE_FILE must be exported prior to the application's first call to any of the PKCS #11 functions.

Table 29. Environment variables for capturing trace data

Environment variable	Usage	Valid values
CSN_PKCS11_TRACE	Specifies the level of tracing to be performed.	<p>An integer value, 1-7. The higher the value, the greater the number of conditions traced. Each level includes the conditions of the levels below it:</p> <p>7 Debug information</p> <p>6 Informational conditions</p> <p>5 Normal but significant conditions</p> <p>4 Warning conditions</p> <p>3 Error conditions</p> <p>2 Critical conditions</p> <p>1 Immediate action required</p> <p>Any other value causes tracing to be inactive (the default).</p>

Table 29. Environment variables for capturing trace data (continued)

Environment variable	Usage	Valid values
CSN_PKCS11_TRACE_FILE	<p>Specifies the name of the trace file. Defaults to /tmp/csnpkcs11.%.trc.</p> <p>The current process identifier is included as part of the trace file name when the name contains a percent sign (%). For example, if CSN_PKCS11_TRACE_FILE is set to:</p> <pre>/tmp/csnpkcs11.%.trc</pre> <p>and the current process identifier is 247, the trace file name is</p> <pre>/tmp/csnpkcs11.247.trc</pre>	<p>Must be set to the full path name of an HFS file in a directory for which the executing application has write permission. The maximum length for the path name is 255 bytes. Values longer than 255 bytes are truncated.</p>

You can also use the utility program, testpkcs11, for troubleshooting. For information about running testpkcs11, see “Running the pre-compiled version of testpkcs11” on page 66.

Chapter 3. Sample PKCS #11 C programs

IBM provides sample PKCS #11 C programs. The source code for the sample programs is provided in `/usr/lpp/pkcs11/samples/`. See “Building sample PKCS #11 applications from source code” on page 67 for instructions on how to build and run a sample program.

- **Sample testpkcs11:** This program is passed the name of a PKCS #11 token, and performs the following tasks:
 1. Creates a token that has the name passed
 2. Generates an RSA key-pair
 3. Encrypts some test data using the public part of the key-pair
 4. Decrypts the data using the private part of the key-pair
 5. Deletes the key-pair and the token

You can use this program in several ways:

As a utility program to test the system configuration for PKCS #11 and troubleshoot problems.

As a sample application to learn how to build and run a PKCS #11 application.

IBM provides a pre-compiled version of this program installed in `/usr/lpp/pkcs11/bin`. For the source code for this program, see Appendix B, “Source code for the testpkcs11 sample program,” on page 73.

- **Sample updatecomp:** This sample program uses the C API for updating the FIPS/BSI compliance mode of all keys in a token. With this program, users can update all the keys in a token to the current compliance mode of the Enterprise PKCS # 11 coprocessor or to some other mode. Users will find this program a useful aid when changing the FIPS/BSI compliance mode of an Enterprise PKCS #11 coprocessor where increasing the coprocessor compliance mode makes all existing Secure PKCS #11 keys unusable. This program creates a backup copy of each key (under the same token) before updating its compliance mode. The label of the copy will indicate the sequence number of the original key. The copied keys act as a backup, allowing one to revert to the previous compliance mode, if necessary. Users can examine the backup and updated keys via the PKCS #11 Token Browser ISPF panels. See “Standard compliance modes” on page 61 for more information.
 - Usage: **updatecomp** { `-t token-name` [`-c comp-mode`] | `-h` }
 - `-t token-name` = The name of the token to be updated.
 - `-c comp-mode` = The numeric value (0-15) to use as the new compliance mode. Optional, ICSF will reset to the current mode of the coprocessor if not specified.
 - `-h` = Displays this help.
 - IBM recommends that you use **updatecomp** only to reset your keys to match the compliance mode of the coprocessor. This can be accomplished by specifying `-c 0` or letting the `-c` switch default to 0. Use caution when specifying a non-zero compliance mode via the `-c` switch. Specifying a value that doesn't match the coprocessor could disable use of the keys permanently.

Running the pre-compiled version of testpkcs11

If you are testing the system configuration for PKCS #11, or troubleshooting problems with the configuration, you can run the pre-compiled version of testpkcs11.

Steps for running the pre-compiled version of testpkcs11

About this task

Before you begin: You need to know how to use z/OS UNIX shells.

Perform the following steps to run the pre-compiled version of testpkcs11.

Procedure

1. Change to the PKCS #11 bin directory by entering the following command:

```
cd /usr/lpp/pkcs11/bin
```

2. Choose a temporary token name to use. If you need to review the rules for token names, see "Tokens" on page 1.

3. Run testpkcs11, passing it your token name on the -t option. For example, to use a temporary token name of my.temp.token, enter the following command:

```
./testpkcs11 -t my.temp.token
```

Results

If z/OS PKCS #11 has been set up properly and the you have sufficient authority to the token label specified, you should see the following output:

```
Getting the PKCS11 function list...
Initializing the PKCS11 environment...
Creating the temporary token...
Opening a session...
Generating keys. This may take a while...
Enciphering data...
Deciphering data...
Destroying keys...
Closing the session...
Deleting the temporary token...
Test completed successfully!
```

If you see different messages, there is an error in either your PKCS #11 set up or in the token label that you specified.

The most common user error is specifying token label that is unacceptable to ICSF or already in use. In that case the following is displayed:

```
Getting the PKCS11 function list...
Initializing the PKCS11 environment...
Creating the temporary token...
C_InitToken #1 returned 7 (0x07) CKR_ARGUMENTS_BAD
Make sure your the token name you specified meets ICSF rules:
Contains only alphanumeric characters, nationals (@#$), and periods.
The first character cannot be a numeric or a period.
```

If you see other error messages, there is probably an error in the setup for the PKCS #11 environment. Determine the error represented by the PKCS #11 error code returned. For information about error codes, see “Function return codes” on page 62.

To display the help text for the testpkcs11 program, run the program with the -h option:

```
cd /usr/lpp/pkcs11/bin
./testpkcs11 -h
```

Building sample PKCS #11 applications from source code

If you are learning how to build and run a PKCS #11 application, you can use the source code for testpkcs11 to build and run a sample application.

Before you begin: You need to know in which directory the PKCS #11 header file is located. By default it is located in the standard include subdirectory under /usr. If your standard include subdirectory is in a different location, you will need to modify the Makefile in step 2. You also need to know how to use z/OS UNIX shells.

Makefiles for sample programs: Makefiles for 31, 64, and 31 XPLINK addressing for the sample programs are provided in /usr/lpp/pkcs11/samples. The naming convention of the makefiles are:

```
Makefile.samplepgmname,
Makefile64.samplepgmname, and
Makefile3X.samplepgmname
```

where *samplepgmname* is the name of the sample C program.

Example: Perform the following steps to build the sample application, testpkcs11. (Other samples would be built in a similar fashion.) Issue the commands from the z/OS UNIX command shell.

1. Copy the testpkcs11.c program and appropriate Makefile to the current directory. For example, for 64-bit addressing enter the following commands:

```
cp /usr/lpp/pkcs11/samples/testpkcs11.c testpkcs11.c
cp /usr/lpp/pkcs11/samples/Makefile64.testpkcs11 Makefile
```

2. If the standard include subdirectory is not located under /usr, edit the Makefile copied in step 1 and change the PKCS11_INSTALL_DIR variable as required.

3. Enter the following command to compile and link and produce the executable, testpkcs11:

```
make
```

4. Update your C/C++ environment variable _CEE_RUNOPTS to include XPLINK(ON) if it does not already include it. For example, execute the following command from a UNIX shell:

```
export _CEE_RUNOPTS=$_CEE_RUNOPTS' XPLINK(ON)'
```

When you are done, you have built the testpkcs1164 application and can run it in your directory. For example, to run testpkcs1164 in your directory and display its help text, enter the following command:

```
./testpkcs1164 -h
```

To run a test, choose a temporary token name and enter it with the -t option. If you need to review the rules for token names, see “Tokens” on page 1. For example, to use a temporary token name of my.temp.token, enter the following command:

```
./testpkcs1164 -t my.temp.token
```

For the output that should appear, see “Steps for running the pre-compiled version of testpkcs11” on page 66.

Chapter 4. ICSF PKCS #11 callable services

The PKCS #11 C language API (described in Chapter 2, “The C API,” on page 21) requires a Language Environment (LE) runtime to operate. Although an LE is normally provided with C application programs, if you are coding your application in some other language (for example, Assembler), acquiring an LE runtime may not be desirable. For these situations, ICSF provides a base set of PKCS #11 callable services that you can use. (In fact, the C API itself uses these services.) These callable services do not require an LE runtime. The ICSF PKCS #11 callable services include:

- Derive key (CSFPDVK)
- Derive multiple keys (CSFPDMK)
- Generate HMAC (CSFPHMG)
- Generate key pair (CSFPGKP)
- Generate secret key (CSFPGSK)
- Get attribute value (CSFPGAV)
- One-way hash generate (CSFPOWH)
- Private key sign (CSFPPKS)
- Pseudo-random function (CSFPPRF)
- Public key verify (CSFPPKV)
- Secret key decrypt (CSFPSKD)
- Secret key encrypt (CSFPSKE)
- Set attribute value (CSFPSAV)
- Token record create (CSFPTRC)
- Token record delete (CSFPTRD)
- Token record list (CSFPTRL)
- Unwrap key (CSFPUWK)
- Verify HMAC (CSFPHMV)
- Wrap key (CSFPWPK)

Calls to the system authorization facility (SAF) determine access authorization for the callable services. The CSFSERV class controls access to the PKCS #11 callable services.

For details about the PKCS #11 callable services, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

Appendix A. SMP/E installation data sets, directories, and files

The following dynamic link libraries (DLLs) are linked into SYS1.SIEALNKE:

CSNPCAPI

The main DLL invoked by applications to use PKCS #11 functions. Also shipped as an HFS file at /usr/lpp/pkcs11/lib/csnpcapi.so.

CSNPCA64

64-bit addressing mode version of CSNPCAPI. Also shipped as an HFS file at /usr/lpp/pkcs11/lib/csnpca64.so.

CSNPCA3X

31-bit addressing mode version of CSNPCAPI with XPLINK. Also shipped as an HFS file at /usr/lpp/pkcs11/lib/csnpca3x.so

CSNPCINT

An internal DLL loaded by CSNPCAPI.

CSNPCI64

64-bit addressing mode version of CSNPCINT.

CSNPCI3X

31-bit addressing mode version of CSNPCINT with XPLINK.

CSNPCUTL

An internal DLL implicitly loaded for utilities.

CSNPCU64

64-bit addressing mode version of CSNPCUTL.

CSNPCU3X

31-bit addressing mode version of CSNPCUTL with XPLINK.

CSFDLL31

CSFDLL in 31-bit addressing mode

CSFDLL64

CSFDLL in 64-bit addressing mode.

CSFDLL3X

31-bit addressing mode version of CSFDLL31 with XPLINK.

SMP/E installs the product files into the HFS directory /usr/lpp/pkcs11. This directory contains the following subdirectories and files:

- /usr/lpp/pkcs11/include subdirectory (members are symbolically linked to /usr/include)

csnpdefs.h

A header file that applications must include to use PKCS #11 functions. Also copied to SYS1.SIEAHDR.H(CSNPDEFS).

csfbext.h

A header file that applications must include to use the CSFDLLs. Also copied to SYS1.SIEAHDR.H(CSFBEXT).

- /usr/lpp/pkcs11/lib subdirectory (members are symbolically linked to /usr/lib)

CSNPCAPI.x

Side deck for CSNPCAPI. Also copied to SYS1.SIEASID(CSNPCAPI).

CSNPCA64.x

Side deck for CSNPCA64. Also copied to SYS1.SIEASID(CSNPCA64).

CSNPCA3X.x

Side deck for CSNPCA3X. Also copied to SYS1.SIEASID(CSNPCA3X).

csnpcapi.so**csnpca64.so****csnpca3x.so****CSFDLL31.x**

Side deck for CSFDLL31. Also copied to SYS1.SIEASID(CSFDLL31).

CSFDLL64.x

Side deck for CSFDLL64. Also copied to SYS1.SIEASID(CSFDLL64).

CSFDLL3X.x

Side deck for CSFDLL3X. Also copied to SYS1.SIEASID(CSFDLL3X).

- `usr/lpp/pkcs11/bin` subdirectory

testpkcs11

Program to test system configuration for PKCS #11.

- `/usr/lpp/pkcs11/samples` subdirectory

testpkcs11.c

Source code for the testpkcs11 program.

Makefile.testpkcs11

Makefile for the testpkcs11 program.

Makefile3X.testpkcs11

Makefile for 31-bit addressing mode version of the testpkcs11 program with XPLINK.

Makefile64.testpkcs11

Makefile for 64-bit addressing mode version of the testpkcs11 program.

updatecomp.c

Source code for the updatecomp.c program.

Makefile.updatecomp

Makefile for the updatecomp.c program.

Makefile3X.updatecomp

Makefile for 31-bit addressing mode version of updatecomp.c program with XPLINK.

Makefile64.updatecomp

Makefile for 64-bit addressing mode version of the updatecomp.c program.

Appendix B. Source code for the testpkcs11 sample program

For information about building and using the testpkcs11 sample program, see Chapter 3, "Sample PKCS #11 C programs," on page 65.

```
/*
*****
*/
/* COMPONENT_NAME: testpkcs11.c
*/
/*
*/
/* Licensed Materials - Property of IBM
*/
/* 5650-ZOS
*/
/* Copyright IBM Corp. 2007, 2013
*/
/* Status = HCR7770
*/
/*
*/
*****
*****
*/
/*
*/
/* This file contains sample code. IBM PROVIDES THIS CODE ON AN
*/
/* 'AS IS' BASIS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR
*/
/* IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
*/
/* OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
*/
/*
*/
*****
*/
/*
*/
/* Change Activity:
*/
/* $L0=P11C1 ,HCR7740, 060124,PDJS: PKCS11 initial release
*/
/* $D1=MG08269 ,HCR7740, 061114,PDJS: Misc fixes
*/
/* $D2=MG08740 ,HCR7740, 070302,PDGL: XPLINK
*/
/* $P1=MG13406 ,HCR7770, 090812,PDGL: fix XPLINK define
*/
/* $P2=MG13431 ,HCR7770, 090826,PDER: update prolog
*/
/*
*/
*****

#ifdef IBM
/* Customers may remove this copyright statement */
#pragma comment (copyright,"\
Licensed Materials - Property of IBM \
5650-ZOS Copyright IBM Corp. 2007, 2013 \
All Rights Reserved \
US Government Users Restricted Rights - \
Use, duplication or disclosure restricted by \
GSA ADP Schedule Contract with IBM Corp.")
#endif

/* Install verification test for PKCS #11 */

#define _UNIX03_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <memory.h>
#include <dlfcn.h>
#include <sys/timeb.h>
#include <csnpdefs.h>

int skip_token_obj;

CK_FUNCTION_LIST *funcs;
CK_SLOT_ID slotID = CK_UNAVAILABLE_INFORMATION;
CK_BYTE tokenName[32];

void ProcessRetCode( CK_RV rc )
{
    switch (rc) {
        case CKR_OK: printf(" CKR_OK"); break;
        case CKR_CANCEL: printf(" CKR_CANCEL"); break;
        case CKR_HOST_MEMORY: printf(" CKR_HOST_MEMORY"); break;
        case CKR_SLOT_ID_INVALID: printf(" CKR_SLOT_ID_INVALID"); break;
        case CKR_GENERAL_ERROR: printf(" CKR_GENERAL_ERROR"); break;
        case CKR_FUNCTION_FAILED: printf(" CKR_FUNCTION_FAILED"); break;
        case CKR_ARGUMENTS_BAD: printf(" CKR_ARGUMENTS_BAD"); break;
    }
}
```

```

case CKR_NO_EVENT:                printf(" CKR_NO_EVENT");                break;
case CKR_NEED_TO_CREATE_THREADS:  printf(" CKR_NEED_TO_CREATE_THREADS");  break;
case CKR_CANT_LOCK:               printf(" CKR_CANT_LOCK");               break;
case CKR_ATTRIBUTE_READ_ONLY:     printf(" CKR_ATTRIBUTE_READ_ONLY");     break;
case CKR_ATTRIBUTE_SENSITIVE:     printf(" CKR_ATTRIBUTE_SENSITIVE");     break;
case CKR_ATTRIBUTE_TYPE_INVALID:  printf(" CKR_ATTRIBUTE_TYPE_INVALID");  break;
case CKR_ATTRIBUTE_VALUE_INVALID: printf(" CKR_ATTRIBUTE_VALUE_INVALID");  break;
case CKR_DATA_INVALID:            printf(" CKR_DATA_INVALID");            break;
case CKR_DATA_LEN_RANGE:          printf(" CKR_DATA_LEN_RANGE");          break;
case CKR_DEVICE_ERROR:            printf(" CKR_DEVICE_ERROR");            break;
case CKR_DEVICE_MEMORY:           printf(" CKR_DEVICE_MEMORY");           break;
case CKR_DEVICE_REMOVED:          printf(" CKR_DEVICE_REMOVED");          break;
case CKR_ENCRYPTED_DATA_INVALID:   printf(" CKR_ENCRYPTED_DATA_INVALID");   break;
case CKR_ENCRYPTED_DATA_LEN_RANGE: printf(" CKR_ENCRYPTED_DATA_LEN_RANGE");  break;
case CKR_FUNCTION_CANCELED:       printf(" CKR_FUNCTION_CANCELED");       break;
case CKR_FUNCTION_NOT_PARALLEL:   printf(" CKR_FUNCTION_NOT_PARALLEL");   break;
case CKR_FUNCTION_NOT_SUPPORTED:  printf(" CKR_FUNCTION_NOT_SUPPORTED");  break;
case CKR_KEY_HANDLE_INVALID:      printf(" CKR_KEY_HANDLE_INVALID");      break;
case CKR_KEY_SIZE_RANGE:          printf(" CKR_KEY_SIZE_RANGE");          break;
case CKR_KEY_TYPE_INCONSISTENT:   printf(" CKR_KEY_TYPE_INCONSISTENT");   break;
case CKR_KEY_NOT_NEEDED:          printf(" CKR_KEY_NOT_NEEDED");          break;
case CKR_KEY_CHANGED:             printf(" CKR_KEY_CHANGED");             break;
case CKR_KEY_NEEDED:              printf(" CKR_KEY_NEEDED");              break;
case CKR_KEY_INDIGESTIBLE:        printf(" CKR_KEY_INDIGESTIBLE");        break;
case CKR_KEY_FUNCTION_NOT_PERMITTED: printf(" CKR_KEY_FUNCTION NOT PERMITTED"); break;
case CKR_KEY_NOT_WRAPPABLE:       printf(" CKR_KEY_NOT_WRAPPABLE");       break;
case CKR_KEY_UNEXTRACTABLE:       printf(" CKR_KEY_UNEXTRACTABLE");       break;
case CKR_MECHANISM_INVALID:       printf(" CKR_MECHANISM_INVALID");       break;
case CKR_MECHANISM_PARAM_INVALID: printf(" CKR_MECHANISM_PARAM_INVALID");  break;
case CKR_OBJECT_HANDLE_INVALID:   printf(" CKR_OBJECT_HANDLE_INVALID");   break;
case CKR_OPERATION_ACTIVE:        printf(" CKR_OPERATION_ACTIVE");        break;
case CKR_OPERATION_NOT_INITIALIZED: printf(" CKR_OPERATION NOT INITIALIZED"); break;
case CKR_PIN_INCORRECT:           printf(" CKR_PIN_INCORRECT");           break;
case CKR_PIN_INVALID:             printf(" CKR_PIN_INVALID");             break;
case CKR_PIN_LEN_RANGE:           printf(" CKR_PIN_LEN_RANGE");           break;
case CKR_PIN_EXPIRED:             printf(" CKR_PIN_EXPIRED");             break;
case CKR_PIN_LOCKED:              printf(" CKR_PIN_LOCKED");              break;
case CKR_SESSION_CLOSED:          printf(" CKR_SESSION_CLOSED");          break;
case CKR_SESSION_COUNT:           printf(" CKR_SESSION_COUNT");           break;
case CKR_SESSION_HANDLE_INVALID:  printf(" CKR_SESSION_HANDLE_INVALID");  break;
case CKR_SESSION_PARALLEL_NOT_SUPPORTED: printf(" CKR_SESSION_PARALLEL NOT SUPPORTED"); break;
case CKR_SESSION_READ_ONLY:       printf(" CKR_SESSION_READ_ONLY");       break;
case CKR_SESSION_EXISTS:          printf(" CKR_SESSION_EXISTS");          break;
case CKR_SESSION_READ_ONLY_EXISTS: printf(" CKR_SESSION_READ_ONLY EXISTS"); break;
case CKR_SESSION_READ_WRITE_SO_EXISTS: printf(" CKR_SESSION_READ_WRITE SO EXISTS"); break;
case CKR_SIGNATURE_INVALID:       printf(" CKR_SIGNATURE_INVALID");       break;
case CKR_SIGNATURE_LEN_RANGE:     printf(" CKR_SIGNATURE_LEN_RANGE");     break;
case CKR_TEMPLATE_INCOMPLETE:     printf(" CKR_TEMPLATE_INCOMPLETE");     break;
case CKR_TEMPLATE_INCONSISTENT:   printf(" CKR_TEMPLATE_INCONSISTENT");   break;
case CKR_TOKEN_NOT_PRESENT:       printf(" CKR_TOKEN NOT PRESENT - ICSF is not active or not configured for TKDS operations"); break;
case CKR_TOKEN_NOT_RECOGNIZED:    printf(" CKR_TOKEN NOT RECOGNIZED - You are not authorized to perform the token operation"); break;
case CKR_TOKEN_WRITE_PROTECTED:   printf(" CKR_TOKEN WRITE PROTECTED");   break;
case CKR_UNWRAPPING_KEY_HANDLE_INVALID: printf(" CKR_UNWRAPPING_KEY_HANDLE INVALID"); break;
case CKR_UNWRAPPING_KEY_SIZE_RANGE: printf(" CKR_UNWRAPPING_KEY_SIZE RANGE"); break;
case CKR_UNWRAPPING_KEY_TYPE_INCONSISTENT: printf(" CKR_UNWRAPPING_KEY_TYPE INCONSISTENT"); break;
case CKR_USER_ALREADY_LOGGED_IN:  printf(" CKR_USER_ALREADY_LOGGED_IN");  break;
case CKR_USER_NOT_LOGGED_IN:      printf(" CKR_USER_NOT_LOGGED_IN");      break;
case CKR_USER_PIN_NOT_INITIALIZED: printf(" CKR_USER_PIN_NOT_INITIALIZED");  break;
case CKR_USER_TYPE_INVALID:        printf(" CKR_USER_TYPE_INVALID");        break;
case CKR_USER_ANOTHER_ALREADY_LOGGED_IN: printf(" CKR_USER_ANOTHER_ALREADY_LOGGED_IN"); break;
case CKR_USER_TOO_MANY_TYPES:     printf(" CKR_USER_TOO_MANY_TYPES");     break;
case CKR_WRAPPED_KEY_INVALID:     printf(" CKR_WRAPPED_KEY_INVALID");     break;
case CKR_WRAPPED_KEY_LEN_RANGE:   printf(" CKR_WRAPPED_KEY_LEN_RANGE");   break;
case CKR_WRAPPING_KEY_HANDLE_INVALID: printf(" CKR_WRAPPING_KEY_HANDLE INVALID"); break;
case CKR_WRAPPING_KEY_SIZE_RANGE: printf(" CKR_WRAPPING_KEY_SIZE_RANGE");  break;
case CKR_WRAPPING_KEY_TYPE_INCONSISTENT: printf(" CKR_WRAPPING_KEY_TYPE INCONSISTENT"); break;
case CKR_RANDOM_SEED_NOT_SUPPORTED: printf(" CKR_RANDOM_SEED_NOT_SUPPORTED"); break;
case CKR_RANDOM_NO_RNG:           printf(" CKR_RANDOM_NO_RNG");           break;
case CKR_BUFFER_TOO_SMALL:        printf(" CKR_BUFFER_TOO_SMALL");        break;
case CKR_SAVED_STATE_INVALID:     printf(" CKR_SAVED_STATE_INVALID");     break;
case CKR_INFORMATION_SENSITIVE:   printf(" CKR_INFORMATION_SENSITIVE");   break;
case CKR_STATE_UNSAVEABLE:        printf(" CKR_STATE_UNSAVEABLE");        break;
case CKR_CRYPTOKI_NOT_INITIALIZED: printf(" CKR_CRYPTOKI_NOT_INITIALIZED"); break;
case CKR_CRYPTOKI_ALREADY_INITIALIZED: printf(" CKR_CRYPTOKI_ALREADY_INITIALIZED"); break;
case CKR_MUTEX_BAD:               printf(" CKR_MUTEX_BAD");               break;
case CKR_MUTEX_NOT_LOCKED:        printf(" CKR_MUTEX_NOT_LOCKED");        break;
/* Otherwise - Value does not match a known PKCS11 return value */

```

```

}
}

void showError( char *str, CK_RV rc )
{
    printf("%s returned:  %d (0x%0x)", str, rc, rc );
    ProcessRetCode( rc );
    printf("\n");
}

CK_RV createToken( void )
{
    CK_VOID_PTR      p = NULL; // @D1C
    CK_RV            rc;
    CK_FLAGS         flags = 0;

    printf("Creating the temporary token... \n");
    /* wait for slot event. On z/OS this creates a new slot synchronously */
    rc = funcs->C_WaitForSlotEvent(flags, &slotID, p);
    if (rc != CKR_OK) {
        showError(" C_WaitForSlotEvent #1", rc );
        return !CKR_OK;
    }
    /* The slot has been created. Now initialize the token in the slot */
    /* On z/OS no PIN is required, so we will pass NULL. */
    rc= funcs->C_InitToken(slotID, NULL, 0, tokenName);
    if (rc != CKR_OK) {
        showError(" C_InitToken #1", rc );
        if (rc == CKR_ARGUMENTS_BAD) {
            printf(" Make sure your the token name you specified meets ICSF rules:\n");
            printf(" Contains only alphanumeric characters, nationals (@#$), and periods.\n");
            printf(" The first character cannot be a numeric or a period.\n");
        }
        return !CKR_OK;
    }

    return CKR_OK;
}

CK_RV deleteToken( void )
{
    CK_VOID_PTR      p;
    CK_RV            rc;
    CK_FLAGS         flags = 0;

    if (slotID != CK_UNAVAILABLE_INFORMATION) {
        printf("Deleting the temporary token... \n");
        /* C_InitToken with the reserved label $$DELETE-TOKEN$$ is the way to delete a token */
        /* on z/OS */
        memset(tokenName, ' ', sizeof(tokenName));
        memcpy(tokenName, DEL_TOK, sizeof(DEL_TOK));
        rc= funcs->C_InitToken(slotID, NULL, 0, tokenName);
        if (rc != CKR_OK) {
            showError(" C_InitToken #2 (for delete)", rc );
            return !CKR_OK;
        }
    }

    return CKR_OK;
}

CK_RV encryptRSA( void )
{
    CK_BYTE          data1[100];
    CK_BYTE          data2[256];
    CK_BYTE          cipher[256];
    CK_SLOT_ID      slot_id;
    CK_SESSION_HANDLE session;
    CK_MECHANISM     mech;
    CK_OBJECT_HANDLE publ_key, priv_key;
    CK_FLAGS         flags;
    CK_ULONG         i;
    CK_ULONG         len1, len2, cipherlen;
    CK_RV            rc;
    static CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
    static CK_KEY_TYPE  type= CKK_RSA;
    static CK_OBJECT_CLASS privclass = CKO_PRIVATE_KEY;

```

```

static CK_BBOOL      true = TRUE;
static CK_BBOOL      false = FALSE;

static CK_ULONG      bits = 1024;
static CK_BYTE       pub_exp[] = { 0x01, 0x00, 0x01 };

/* Attributes for the public key to be generated */
CK_ATTRIBUTE pub_tmpl[] = {
    {CKA_MODULUS_BITS,    &bits,    sizeof(bits)    },
    {CKA_ENCRYPT,         &>true,     sizeof(true)   },
    {CKA_VERIFY,        &>true,     sizeof(true)   },
    {CKA_PUBLIC_EXPONENT, &pub_exp, sizeof(pub_exp) }
};

/* Attributes for the private key to be generated */
CK_ATTRIBUTE priv_tmpl[] =
{
    {CKA_DECRYPT, &true, sizeof(true) },
    {CKA_SIGN,  &true, sizeof(true) }
};

slot_id = slotID;
flags = CKF_SERIAL_SESSION | CKF_RW_SESSION;
printf("Opening a session... \n");
rc = funcs->C_OpenSession( slot_id, flags, (CK_VOID_PTR) NULL, NULL, &session );
if (rc != CKR_OK) {
    showError(" C_OpenSession #1", rc );
    return !CKR_OK;
}

printf("Generating keys. This may take a while... \n");
mech.mechanism      = CKM_RSA_PKCS_KEY_PAIR_GEN;
mech.ulParameterLen = 0;
mech.pParameter     = NULL;

rc = funcs->C_GenerateKeyPair( session, &mech,
                               pub_tmpl,  4,
                               priv_tmpl, 2,
                               &publ_key, &priv_key );

if (rc != CKR_OK) {
    showError(" C_GenerateKeyPair #1", rc );
    return !CKR_OK;
}

/* now, encrypt some data */
len1      = sizeof(data1);
len2      = sizeof(data2);
cipherlen = sizeof(cipher);

for (i=0; i < len1; i++)
    data1[i] = (i) % 255;

mech.mechanism      = CKM_RSA_PKCS;
mech.ulParameterLen = 0;
mech.pParameter     = NULL;

printf("Enciphering data... \n");
rc = funcs->C_EncryptInit( session, &mech, publ_key );
if (rc != CKR_OK) {
    showError(" C_EncryptInit #1", rc );
    funcs->C_CloseSession( session );
    return !CKR_OK;
}

rc = funcs->C_Encrypt( session, data1, len1, cipher, &cipherlen );
if (rc != CKR_OK) {
    showError(" C_Encrypt #1", rc );
    funcs->C_CloseSession( session );
    return !CKR_OK;
}

/* now, decrypt the data */
printf("Deciphering data... \n");
rc = funcs->C_DecryptInit( session, &mech, priv_key );
if (rc != CKR_OK) {
    showError(" C_DecryptInit #1", rc );
    funcs->C_CloseSession( session );
}

```

```

    return !CKR_OK;
}

rc = funcs->C_Decrypt( session, cipher, cipherlen, data2, &len2 );
if (rc != CKR_OK) {
    showError("    C_Decrypt #1", rc);
    funcs->C_CloseSession( session );
    return !CKR_OK;
}

/* PKCS - returns clear data as is */
if (len1 != len2) {
    printf("    ERROR: lengths don't match\n");
    printf("    Length of original data = %d, after decryption = %d\n",len1, len2);
    funcs->C_CloseSession( session );
    return !CKR_OK;
}

for (i=0; i < len1; i++) {
    if (data1[i] != data2[i]) {
        printf("    ERROR: mismatch at byte %d\n", i );
        funcs->C_CloseSession( session );
        return !CKR_OK;
    }
}

printf("Destroying keys... \n");
rc = funcs->C_DestroyObject( session, priv_key );
if (rc != CKR_OK) {
    showError("    C_DestroyObject #1", rc );
    funcs->C_CloseSession( session );
    return !CKR_OK;
}

rc = funcs->C_DestroyObject( session, publ_key );
if (rc != CKR_OK) {
    showError("    C_DestroyObject #2", rc );
    funcs->C_CloseSession( session );
    return !CKR_OK;
}

printf("Closing the session... \n");
rc = funcs->C_CloseSession( session );
if (rc != CKR_OK) {
    showError("    C_CloseSession #1", rc );
    return !CKR_OK;
}

return CKR_OK;
}

```

```

CK_RV getFunctionList( void )
{
    CK_RV      rc;
    CK_RV      (*pFunc)();
    void      *d;
#ifdef _LP64
    char      e[]="CSNPCA64";
#elif _XPLINK_
    char      e[]="CSNPCA3X";          /* @P1C */
#else
    char      e[]="CSNPCA3I";          /* @D2A */
#endif

    printf("Getting the PKCS11 function list...\n");

    d = dlopen(e,RTLD_NOW);
    if ( d == NULL ) {
        printf("%s not found in linklist or LIBPATH\n",e); // @D1A
        return !CKR_OK;
    }

    pFunc = (CK_RV (*)())dlsym(d,"C_GetFunctionList");
    if (pFunc == NULL) {
        printf("C_GetFunctionList() not found in module %s\n",e); // @D1A
    }
}

```

```

    return !CKR_OK;
}
rc = pFunc(&funcs);

if (rc != CKR_OK) {
    showError("    C_GetFunctionList", rc );
    return !CKR_OK;
}

return CKR_OK;
}

void displaySyntax(char *pgm) {
    printf("usage:  %s { -t <token-name> | -h }\n\n", pgm );
    printf(" -t <token-name> = The name of a temporary token to create for the test. The\n");
    printf("   name must be less than 33 characters in length and contains only alphanumeric\n");
    printf("   characters, nationals (@#$), and periods. The first character cannot be a\n");
    printf("   numeric or a period. The token will be deleted when the test is complete.\n\n");
    printf(" -h = Displays this help.\n\n");
}

void main( int argc, char **argv )
{
    CK_C_INITIALIZE_ARGS  cinit_args;
    CK_RV                rc, i;

    memset(tokenName, ' ', sizeof(tokenName)); /* Token name is left justified, padded with blanks */

    if (argc == 3) {
        if (strcmp(argv[1], "-t") == 0)
            if (strlen(argv[2]) > 0 && strlen(argv[2]) < 33) {
                memcpy(tokenName, argv[2], strlen(argv[2]));
            }
        else {
            displaySyntax(argv[0]);
            return;
        }
    }
    else {
        displaySyntax(argv[0]);
        return;
    }
}

rc = getFunctionList();
if (rc != CKR_OK) {
    printf("getFunctionList failed!\n"); // @D1C
    return;
}

memset( &cinit_args, 0x0, sizeof(cinit_args) );
cinit_args.flags = CKF_OS_LOCKING_OK;

printf("Initializing the PKCS11 environment...\n");
rc = funcs->C_Initialize( &cinit_args );
if (rc != CKR_OK) {
    showError("    C_Initialize", rc );
    return;
}

rc = createToken();
if (rc != CKR_OK) {
    funcs->C_Finalize( NULL );
    return;
}

rc = encryptRSA();
if (rc != CKR_OK) {
    deleteToken();
    funcs->C_Finalize( NULL );
    return;
}
}

```

```
rc = deleteToken();
if (rc != CKR_OK) {
    funcs->C_Finalize( NULL );
    return;
}

rc = funcs->C_Finalize( NULL );
if (rc != CKR_OK) {
    showError(" C_Initialize", rc );
    return;
}
printf("Test completed successfully!\n");
}
```

Appendix C. Accessibility

Accessible publications for this product are offered through the z/OS Information Center, which is available at www.ibm.com/systems/z/os/zos/bkserv/.

If you experience difficulty with the accessibility of any z/OS information, please send a detailed message to mhvrcfs@us.ibm.com or to the following mailing address:

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Accessibility features

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 *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Vol I* 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.

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users accessing the z/OS Information Center using a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line, because they can be considered as a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that your screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually

exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, you know that your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol can be used next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol giving information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, this indicates a reference that is defined elsewhere. The string following the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you should refer to separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:

- ? means an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.
- ! means a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicates that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP will be applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1!

(KEEP), and 2.1.1 (DELETE), the default option KEEP only applies to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

- * means a syntax element that can be repeated 0 or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3*, 3 HOST, and 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Note:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
 2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you could write HOST STATE, but you could not write HOST HOST.
 3. The * symbol is equivalent to a loop-back line in a railroad syntax diagram.
- + means a syntax element that must be included one or more times. A dotted decimal number followed by the + symbol indicates that this syntax element must be included one or more times; that is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can only repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loop-back line in a railroad syntax diagram.

Notices

This information was developed for products and services offered in the U.S.A. or elsewhere.

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
U.S.A

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, 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:

Site Counsel
IBM Corporation
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.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

COPYRIGHT LICENSE:

This information might contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS™, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted

for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (<http://www.ibm.com/software/support/systemsz/lifecycle/>)
- For information about currently-supported IBM hardware, contact your IBM representative.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available at Copyright and Trademark information (<http://www.ibm.com/legal/copytrade.shtml>).

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 Terminology located at: IBM Glossary of Computing Terms (<http://www.ibm.com/software/globalization/terminology/>)

This glossary includes terms and definitions from:

- IBM Terminology definitions are identified by the symbol (D) after the definition.
- The American National Standard Dictionary for Information Technology, ANSI INCITS 172, 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:"

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 X9.19

An ANSI standard that specifies an optional double-MAC procedure which requires a double-length MAC key.

application program

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.

A program used to connect and communicate with stations in a network, enabling users to perform application-oriented activities. (D)

application program interface (API)

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)

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)

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.

Central Credit Committee

The official English name for *Zentraler Kreditausschuss*, also known as ZKA. ZKA was founded in 1932 and was renamed in August 2011 to *Die Deutsche Kreditwirtschaft*, also known as DK. DK is an association of the German banking industry. The hybrid term in English for DK is 'German Banking Industry Committee'.

CEX2A

Crypto Express2 Accelerator

CEX2C

Crypto Express2 Coprocessor

CEX3A

Crypto Express3 Accelerator

CEX3C

Crypto Express3 Coprocessor

checksum

The sum of a group of data associated with the group and used for checking purposes. (T)

In ICSF, the data used is a key part. The resulting checksum is a two-digit value you enter when you enter a master key part.

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 both of 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

In computer security, text produced by encryption.

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 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)
- control-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**
A storage area used by a computer program to hold control information. (I)
Synonymous with control area.

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 tamper responding, programmable, cryptographic PCI card, containing CPU, encryption hardware, RAM, persistent memory, hardware random number generator, time of day clock, infrastructure firmware, and software.

cryptographic key data set (CKDS)

A data set that contains the encrypting keys used by an installation. (D)

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

The transformation of data to conceal its meaning.

In computer security, the principles, means, and methods for encrypting plaintext and decrypting ciphertext. (D)

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.

CUSP (Cryptographic Unit Support Program)

The IBM cryptographic offering, program product 5740-XY6, using the channel-attached 3848. CUSP is no longer in service.

CUSP/PCF conversion program

A program, for use during migration from CUSP or PCF to ICSF, that converts a CUSP or PCF cryptographic key data set into a 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.

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

A key used to encipher, decipher, or authenticate data. (D)

In ICSF, a 64-bit encryption key used to protect data privacy using the DES 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

To convert enciphered data in order to restore the original data. (T)

In computer security, to convert ciphertext into plaintext by means of a cipher system.

To convert enciphered data into clear data. Contrast with encipher. Synonymous with decrypt. (D)

decode

To convert data by reversing the effect of some previous encoding. (I) (A)

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 Algorithm (DSA)

A public key algorithm for digital signature generation and verification used with the Digital Signature Standard.

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).

DK *Die Deutsche Kreditwirtschaft* (German Banking Industry Committee). Formerly known as ZKA.

domain

That part of a network in which the data processing resources are under common control. (T)

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.

ECB Electronic codebook.

ECI Euroch \hat{u} que International S.C., a financial institution consortium that has defined three PIN block formats.

EID Environment Identification.

electronic codebook (ECB) operation

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)

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

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.

Contrast with decipher. (D)

enciphered data

Data whose meaning is concealed from unauthorized users or observers. (D)

encode

To convert data by the use of a code in such a manner that reconversion to the original form is possible. (T)

In computer security, to convert plaintext into an unintelligible form by means of a code system. (D)

In ICSF, to encipher data by use of a clear key.

encrypt

See encipher.

exit

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)

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.

file A named set of records stored or processed as a unit. (T)

GBP German Bank Pool.

German Bank Pool (GBP)

A German financial institution consortium that defines specific methods of PIN calculation.

German Banking Industry Committee

A hybrid term in English for *Die Deutsche Kreditwirtschaft*, also known as DK, an association of the German banking industry. Prior to August 2011, DK was named ZKA for *Zentraler Kreditausschuss*, or Central Credit Committee. ZKA was founded in 1932.

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)

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)

The initialization procedure that causes an operating system to commence operation.

The process by which a configuration image is loaded into storage at the beginning of a work day or after a system malfunction.

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.

job control language (JCL)

A control language used to identify a job to an operating system and to describe the job's requirements. (D)

key-encrypting key (KEK)

In computer security, a key used for encryption and decryption of other keys. (D)

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 a cryptographic coprocessor that accumulates key parts as they are entered via TKE.

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.

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.

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

In computer security, the top-level key in a hierarchy of key-encrypting keys.

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.

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)

The cryptographic result of block cipher operations on text or data using the cipher block chain (CBC) mode of operation. (D)

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)

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.

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 a cryptographic coprocessor 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.

OAEP Optimal asymmetric encryption padding.

offset The process of exclusively ORing a counter to a key.

old master key register

A register in a cryptographic coprocessor 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.

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)

PCICA

PCI Cryptographic Accelerator.

PCI X Cryptographic Coprocessor

An asynchronous cryptographic

coprocessor available on the IBM eServer zSeries 990 and IBM eServer 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-multiplexer channel capability, and supports the VM/XA System Product enhancement for Multiple Preferred Guests.

Programmed Cryptographic Facility (PCF)

An IBM licensed program that provides facilities for enciphering and deciphering data and for creating, maintaining, and managing cryptographic keys. (D)

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.

RACE Integrity Primitives Evaluatiuon 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

A code used to influence the execution of succeeding instructions. (A)

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 Electronic Transaction

A standard created by Visa International and MasterCard for safe-guarding payment card purchases made over open networks.

secure key

A key that is encrypted under a master key. When ICSF uses a secure key, it is passed to a cryptographic coprocessor where the coprocessor decrypts the key and performs the function. The secure key never appears in the clear outside of the cryptographic coprocessor.

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.

special secure mode

An alternative form of security that allows you to enter clear keys with the key generator utility program or generate clear PINs.

SSL Secure Sockets Layer.**supervisor state**

A state during which a processing unit can execute input/output and other privileged instructions. (D)

System Authorization Facility (SAF)

An interface to a system security system like the Resource Access Control Facility (RACF).

system key

A key that ICSF creates and uses for internal processing.

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)

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.

UAT UDX Authority Table.

UDF User-defined function.

UDK User-derived key.

UDP User Developed Program.

UDX User Defined Extension.

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.

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. The NSP includes a 4755 cryptographic adapter in a workstation which is channel attached to a S/390 host computer.

4758 The IBM PCI Cryptographic processor provides a secure programming and hardware environment where DES and RSA processes are performed.

Index

A

accessibility 81
 contact IBM 81
 features 81
assistive technologies 81
auditing PKCS #11 functions 10

C

C application program interface (API),
 using 21
CCA (Common Cryptographic
 Architecture) 1
CK_RV CSN_FindAllObjects()
 function 55
Common Cryptographic Architecture
 (CCA) 1
component trace entries for TKDS
 events 10
constants, manifest
 where defined 21
crypto education x
Cryptoki 1
CRYPTOZ class 3
CSFSERV class
 resources for token services 5

D

data object
 attributes that ICSF supports 30
deleting token 21
Diffie-Hellman domain parameter object
 attributes that ICSF supports 30
Diffie-Hellman private key object
 attributes that ICSF supports 30
Diffie-Hellman public key object
 attributes that ICSF supports 30
DLLs provided by ICSF 71
domain parameter object
 attributes that ICSF supports 30
DSA domain parameter object
 attributes that ICSF supports 30
DSA private key object
 attributes that ICSF supports 30
DSA public key object
 attributes that ICSF supports 30
dynamic link libraries (DLLs) provided
 by ICSF 71

E

Elliptic Curve private key object
 attributes that ICSF supports 30
Elliptic Curve public key object
 attributes that ICSF supports 30
environment variables for tracing 63

F

FIPS 140-2
 algorithms restricted when complying
 with 29
 operating in compliance with 11
function
 non-standard PKCS #11 supported by
 ICSF 55
 standard PKCS #11 supported by
 ICSF 48
function return code
 unique to z/OS, list of 62

H

header file for C API 71

K

key types supported 23
keyboard
 navigation 81
 PF keys 81
 shortcut keys 81

L

Language Environment 21
library
 information that can be set and
 queried 47

M

manifest constants
 where defined 21
mechanism
 information returned by
 C_GetMechanismInfo 23
 which cryptographic hardware
 supports 27

N

navigation
 keyboard 81
Notices 85

O

object
 data
 attributes that ICSF supports 30
 Diffie-Hellman domain parameter
 attributes that ICSF supports 30
 Diffie-Hellman private key
 attributes that ICSF supports 30

object (*continued*)

 Diffie-Hellman public key
 attributes that ICSF supports 30
 domain parameter
 attributes that ICSF supports 30
 DSA domain parameter
 attributes that ICSF supports 30
 DSA private key
 attributes that ICSF supports 30
 DSA public key
 attributes that ICSF supports 30
 Elliptic Curve private key
 attributes that ICSF supports 30
 Elliptic Curve public key
 attributes that ICSF supports 30
 private key
 attributes that ICSF supports 30
 RSA private key
 attributes that ICSF supports 30
 RSA public key
 attributes that ICSF supports 30
 secret key
 attributes that ICSF supports 30
 X.509 certificate
 attributes that ICSF supports 30
object type
 supported by ICSF, list of 30

P

PIN 3
PKCS (Public Key Cryptography
 Standards) 1
private key object
 attributes that ICSF supports 30
program, sample
 building and using 65
 source code for 73
Public Key Cryptography Standards
 (PKCS) 1
public key object
 attributes that ICSF supports 30

R

RSA private key object
 attributes that ICSF supports 30
RSA public key object
 attributes that ICSF supports 30

S

sample C program, PKCS #11
 building and using 65
sample program, testpkcs11
 source code for 73
secret key object
 attributes that ICSF supports 30
sending comments to IBM xi
session object 11

- shortcut keys 81
- slot 1
 - information that can be queried 47
- slot ID 1
- SMF records written for PKCS #11
 - functions 10
- SO R/W
 - description 4
- SO role 3
- SRB mode 21
- Strong SO
 - description 4

T

- tasks
 - running the sample program
 - steps 66
- TCB mode 21
- testpkcs11 program
 - building and using 65
 - source code for 73
- TKDS (token data set)
 - description 2
- token
 - access levels 4
 - deleting 21
 - information that can be set and queried 47
 - managing, interfaces for 7
- token data set (TKDS)
 - description 2
- token object 11
- token, on z/OS
 - description 1
 - rules for name 1
- trace data 63
- trace entries for TKDS events 10
- trademarks 87
- troubleshooting applications 63

U

- updatecomp program
 - building and using 65
- user interface
 - ISPF 81
 - TSO/E 81
- User R/O
 - description 4
- User R/W
 - description 4
- User role 3

V

- V2R1 changed information FMID
 - HCR77A1 xiv
- V2R1 changed information FMID
 - HCR77B0 xiii
- V2R1 deleted information FMID
 - HCR77A1 xiv
- V2R1 deleted information FMID
 - HCR77B0 xiii
- V2R1 new information FMID
 - HCR77A1 xiii

- V2R1 new information FMID
 - HCR77B0 xiii

W

- Weak SO
 - description 4
- Weak User
 - description 4

X

- X.509 certificate object
 - attributes that ICSF supports 30

Z

- z/OS PKCS #11 token
 - deleting 21
 - description 1
 - rules for name 1



Printed in USA

SC14-7510-01

