Cryptographic Services
System Secure Sockets Layer
Programming
Cryptographic Services
System Secure Sockets Layer
Programming
Note

Before using this information and the product it supports, be sure to read the general information under Appendix D, "Notices," on page 495.

Ninth Edition (September 2009)

This edition applies to Version 1 Release 11 of z/OS (program number 5694-A01) and all subsequent releases and modifications until otherwise indicated in new editions.

This is a major revision of SC24-5901-07.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this document, or you may address your comments to the following address:

International Business Machines Corporation
MHVRCFS, Mail Station P181
2455 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+845+432-9405
FAX (Other Countries): Your International Code +1+845+432-9405

IBMLink (United States customers only): IBMUSM10(MHVRCFS)
Internet e-mail: mhvrdfs@us.ibm.com

If you would like a reply, be sure to include your name, address, telephone number, or FAX number. Make sure to include the following in your comment or note:
• Title and order number of this book
• Page number or topic related to your comment

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

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

Figures ............................................................... ix

Tables. ............................................................... xi

Preface. .................................................................... xiii
Who should use this information ................................ xiii
How this information is organized .............................. xiii
Conventions used in this information ............................ xiv
Where to find more information ................................... xv

Summary of changes .................................................. xvii

Chapter 1. Introduction ............................................... 1
Software Dependencies .............................................. 1
Installation Information ............................................. 2

Chapter 2. How System SSL Works for Secure Socket Communication
Using System SSL on z/OS ........................................... 5
System SSL Application Overview ............................... 6

Chapter 3. Using Hardware Cryptographic Features with System SSL ............ 9

Chapter 4. System SSL and FIPS 140-2.......................... 13
Algorithms and key sizes. .......................................... 13
Certificates ............................................................. 13
SSL/TLS Protocol .................................................... 14
System SSL Module Verification Setup ......................... 14
Certificate Stores ...................................................... 17
Application Changes ................................................ 17
SSL Started Task ..................................................... 18

Chapter 5. Writing and Building a z/OS System SSL Application .................. 19
Writing a System SSL Source Program ........................ 19
Building a z/OS System SSL Application ..................... 24
Running a z/OS System SSL Application ...................... 24
Additional Topics ..................................................... 25

Chapter 6. Migrating from Deprecated SSL Interfaces .......................... 33

Chapter 7. API Reference ............................................ 35
gsk_attribute_get_buffer() ......................................... 37
gsk_attribute_get_cert_info() ..................................... 40
gsk_attribute_get_data() ............................................. 45
gsk_attribute_get_enum() ............................................. 47
gsk_attribute_get_numeric_value() ............................. 50
gsk_attribute_set_buffer() ......................................... 52
gsk_attribute_set_callback() ...................................... 55
gsk_attribute_set_enum() ............................................. 59
gsk_attribute_set_numeric_value() ............................. 62
gsk_attribute_set_tls_extension() ............................... 64
gsk_environment_close() ............................................ 67
gsk_environment_init() .............................................. 68
gsk_environment_open() ............................................. 70
Chapter 8. Certificate Management Services (CMS) API Reference

gsk_add_record() .................................. 105

gsk_change_database_password() ......................... 107

gsk_change_database_record_length() ........................ 109

gsk_close_database() ................................ 110

gsk_close_directory() ................................ 111

gsk_construct_certificate() ................................ 112

gsk_construct_private_key_rsa() ........................... 115

gsk_construct_public_key_rsa() ............................ 116

gsk_construct_renewal_request() ........................... 117

gsk_construct_self_signed_certificate() .................. 119

gsk_construct_signed_certificate() ........................ 121

gsk_copy_attributes_signers() ............................ 124

gsk_copy_buffer() ................................... 125

gsk_copy_certificate() .................................. 126

gsk_copy_certificate_extension() .......................... 127

gsk_copy_certification_request() .......................... 128

gsk_copy_content_info() ................................ 129

gsk_copy_crl() ...................................... 130

gsk_copy_name() ..................................... 131

gsk_copy_private_key_info() ............................. 132

gsk_copy_public_key_info() .............................. 133

gsk_copy_record() .................................... 134

gsk_create_certification_request() ......................... 135

gsk_create_database() .................................. 138

gsk_create_database_renewal_request() ..................... 140

gsk_create_database_signed_certificate() .................. 143

gsk_create_renewal_request() ............................. 147

gsk_create_self_signed_certificate() ....................... 149

gsk_create_signed_certificate() ........................... 152

gsk_create_signed_certificate_record() .................... 155

gsk_create_signed_certificate_set() ....................... 158

gsk_create_signed_crl() ................................ 162

gsk_create_signed_crl_record() .......................... 164

gsk_decode_base64() ................................... 167

gsk_decode_certificate() ................................ 168

gsk_decode_certificate_extension() ........................ 169

gsk_decode_certification_request() ......................... 171

gsk_decode_crl() ..................................... 172

gsk_decode_import_certificate() .......................... 173

gsk_decode_import_key() ................................ 174
gsk_decode_name() ............................................. 176
gsk_decode_private_key() ...................................... 177
gsk_decode_public_key() ........................................ 178
gsk_delete_record() ............................................. 179
gsk_dn_to_name() ................................................ 180
gsk_encode_base64() ............................................... 183
gsk_encode_certificate_extension() ............................ 184

gsk_encode_certificate() ........................................ 186
gsk_encode_export_certificate() .................................. 186

gsk_encode_export_key() ......................................... 188

gsk_encode_export_request() ..................................... 190

gsk_encode_name() ............................................... 191

gsk_encode_private_key() ......................................... 192

gsk_encode_public_key() ......................................... 193

gsk_encode_signature() ........................................... 194

gsk_export_certificate() .......................................... 195

gsk_export_certification_request() ............................. 197

gsk_export_key() ................................................ 199

gsk_factor_private_key_rsa() .................................... 202

gsk_factor_public_key_rsa() ..................................... 203

gsk_fips_state_query() .......................................... 204

gsk_fips_state_set() ............................................. 205

gsk_free_attributes_signers() ................................... 206

gsk_free_buffer() ................................................ 207

gsk_free_certificate() ............................................ 208

gsk_free_certificates() ........................................... 209

gsk_free_certificate_extension() ............................... 210

gsk_free_certification_request() ............................... 211

gsk_free_content_info() .......................................... 212

gsk_free_crl() .................................................... 213

gsk_free_crls() ................................................... 214

gsk_free_decoded_extension() .................................... 215

gsk_free_name() .................................................. 216

gsk_free_private_key_info() ..................................... 217

gsk_free_public_key_info() ....................................... 218

gsk_free_record() ................................................ 219

gsk_free_records() ............................................... 220

gsk_free_string() ................................................ 221

gsk_free_strings() ............................................... 222

gsk_generate_key_agreement_pair() ............................. 223

gsk_generate_key_pair() .......................................... 224

gsk_generate_key_parameters() .................................. 226

gsk_generate_random_bytes() .................................... 228

gsk_generate_secret() ............................................. 229

gsk_get_certificate_algorithms() ............................... 230

gsk_get_cms_vector() ............................................. 231

gsk_get_default_key() ............................................ 232

gsk_get_default_label() ........................................... 233

gsk_get_directory_certificates() ............................... 234

gsk_get_directory_crls() ........................................ 235

gsk_get_directory_enum() ....................................... 237

gsk_get_record_by_id() .......................................... 238

gsk_get_record_by_index() ....................................... 239

gsk_get_record_by_label() ....................................... 240

gsk_get_record_by_subject() ..................................... 241

gsk_get_record_labels() ........................................... 242

gsk_get_update_code() .......................................... 243
## Figures

1. Sockets Programming Model Using System SSL ........................................ 8
2. Database Menu .................................................................................. 376
3. Key Management Menu/Token Management Menu .......................... 379
4. Key and Certificate Menus .............................................................. 380
5. Certificate Menus ........................................................................... 382
6. Request Menus ................................................................................. 383
7. Starting Menu for gskkyman ........................................................... 388
8. Creating a New Key Database .......................................................... 389
9. Key Management Menu for gskkyman ....................................... 390
10. Opening an Existing Key Database File ........................................ 391
11. Key Management Menu .................................................................. 391
12. Deleting an Existing Key Database .............................................. 392
13. Changing a Key Database Password ............................................ 393
14. Key Management Menu ................................................................. 394
15. Creating a New z/OS PKCS #11 Token .......................................... 395
16. Opening a z/OS PKCS #11 Token via token name ..................... 396
17. Opening a z/OS PKCS #11 Token via token list ......................... 396
18. Token Management Menu ............................................................... 397
19. Deleting an existing z/OS PKCS #11 Token .................................. 398
20. Deleting an existing z/OS PKCS #11 Token .................................. 399
21. Creating a Self-Signed Certificate ............................................... 400
22. Creating a Self-Signed Certificate .............................................. 401
23. Key Management Menu/Token Management Menu ................ 402
24. Creating a Certificate Request ..................................................... 403
25. Specifying subject alternate names ............................................. 404
27. Creating a key parameter file to be used with Diffie-Hellman .... 406
28. Creating a certificate to be used with Diffie_Hellman ................ 407
29. Receiving a Certificate Issued for your Request ......................... 408
30. Key and Certificate List ................................................................. 409
31. Key and Certificate Menu .............................................................. 410
32. Certificate Information ................................................................. 411
33. Certificate Extensions List ............................................................. 411
34. Key Usage Information ................................................................. 412
35. Key Information menu ................................................................. 412
36. Marking a Certificate (and Private Key) as the Default Certificate. 413
37. Copying a Certificate Without its Private Key ................................ 414
38. Copying a Certificate and Private key to a Different Key Database 415
39. Copying a Certificate with its Private Key to a Key Database on the Same System 416
40. Copying a Certificate with its Private Key to a z/OS PKCS #11 Token on the Same System 417
41. Delete Certificate and Key ........................................................... 418
42. Changing a Certificate Label ....................................................... 418
43. Select 10 to Create a Signed Certificate and Key ....................... 419
44. Certificate Type ........................................................................... 420
45. Subject Alternate Name Type ....................................................... 420
46. Select 11 to Create a Certificate Renewal Request ..................... 421
47. Certificate List (part 1) ................................................................. 422
48. Certificate List (part 2) ................................................................. 422
49. Importing a Certificate from a File .............................................. 423
50. Importing a Certificate and Private Key from a File ................... 424
## Tables

1. FMID Encryption Capabilities .............................................. 1
2. Hardware cryptographic functions exploited by System SSL .......................... 9
3. Hardware cryptographic functions used by System SSL ............................. 10
4. Algorithm support: FIPS and non-FIPS .................................. 13
5. Server Communicating with Clients Via a Socket ................................ 25
6. Using the select() Routine .................................................. 25
7. SAF Access Levels ............................................................. 375
8. SSL-Specific Environment Variables ........................................ 481
9. System Environment Variables used by SSL .................................. 490
Preface

This contains information about the System SSL product. This information consists of primarily two sets of APIs and a Certificate Management utility. The first set of APIs support the Secure Sockets Layer protocols (SSL V2.0, SSL 3.0, TLS V1.0, and TLS V1.1) which can be utilized by C/C++ applications to communicate securely across an open communications network. The other set of APIs (Certificate Management) provide the ability to exploit function other than the SSL protocols. These functions include the ability to create/manage key database files in a similar function to the SSL Certificate Management utility, use certificates stored in a key database file, SAF key ring or z/OS PKCS #11 token for purposes other than SSL and basic PKCS #7 message support to provide application writers a mechanism to communicate with another application through the PKCS #7 standard.

This information also provides guidance on how to write a client and server secure sockets layer application. The client and server may both reside on z/OS™ systems or reside on different systems.

Who should use this information

This information is intended to assist system administrators in setting up the system to use System SSL support and application programmers in writing System SSL applications.

How this information is organized

The format and organization of this information:

Chapter 1, “Introduction,” on page 1 describes Secure Sockets Layer (SSL) and lists the software dependencies and installation information you need to use the System SSL support.

Chapter 2, “How System SSL Works for Secure Socket Communication,” on page 5 provides a general overview of System SSL and the basic structure of a z/OS application using System SSL.

Chapter 3, “Using Hardware Cryptographic Features with System SSL,” on page 9 describes the Hardware Cryptographic features.

Chapter 4, “System SSL and FIPS 140-2,” on page 13 describes how to execute System SSL securely in a mode designed to meet FIPS 140-2 Level 1 criteria.

Chapter 5, “Writing and Building a z/OS System SSL Application,” on page 19 describes how to write a System SSL source program and build the System SSL application.

Chapter 6, “Migrating from Deprecated SSL Interfaces,” on page 33 describes how to migrate an existing application which uses the deprecated SSL interfaces to the latest SSL interfaces.

Chapter 7, “API Reference,” on page 35 describes the System SSL program interfaces.


Chapter 9, “Deprecated Secure Sockets Layer APIs,” on page 345 describes the deprecated System SSL program interfaces.

Chapter 10, “Certificate/Key Management,” on page 373 describes how to use the gskkyman utility to create a key database file, a z/OS PKCS #11 token, a public/private key pair, a certificate request, and other tasks.
Chapter 11, “SSL Started Task,” on page 435 provides sysplex session cache support and dynamic trace support.

Chapter 12, “Obtaining Diagnostic Information,” on page 439 provides debugging information.

Chapter 13, “Messages and Codes,” on page 445 contains various messages and codes you may encounter using System SSL.

Appendix A, “Environment Variables,” on page 481 lists the environment variables used by System SSL.

Appendix B, “Sample C++ SSL Files,” on page 491 describes the sample set of files shipped to provide an example of what is needed to build a C++ System SSL application.

Appendix C, “Accessibility,” on page 493 describes accessibility features provided with this product to help a user who has a physical disability.

Appendix D, “Notices,” on page 495 lists various trademark and licensing notices.

---

**Conventions used in this information**

This information uses these typographic conventions:

**Bold**  **Bold** words or characters

**Highlighting1**

Words or characters highlighted in this manner represent system elements that you must enter into the system literally, such as commands, options, or path names.

**Italic**  **Italic** words or characters

**Highlighting2**

Words or characters highlighted in this manner represent values for variables that you must supply.

**Example font**

Examples and information displayed by the system appear in constant width type style.

[]  Brackets enclose optional items in format and syntax descriptions.

{}  Braces enclose a list from which you must choose an item in format and syntax descriptions.

|  A vertical bar separates items in a list of choices.

<>  Angle brackets enclose the name of a key on the keyboard.

...  Horizontal ellipsis points indicate that you can repeat the preceding item one or more times.

\  A backslash is used as a continuation character when entering commands from the shell that exceed one line (255 characters). If the command exceeds one line, use the backslash character \ as the last non blank character on the line to be continued, and continue the command on the next line.

This information uses these keying conventions:

<ALT-c>

The notation <Alt-c> followed by the name of a key indicates a control character sequence.

<Return>

The notation <Return> refers to the key on your keyboard that is labeled with the word Return or Enter, or with a left arrow.

**Entering commands**

When instructed to enter a command, type the command name and then press <Return>. 

xiv  z/OS V1R11.0 System SSL Programming
Where to find more information

Where necessary, this information references information in other books, using shortened versions of the book title. For complete titles and order numbers of the books for all products that are part of z/OS, see the z/OS Information Roadmap. For a list of titles and order numbers of the books that are useful for the SSL Services, see z/OS Security Server Publications.

Softcopy Publications

The z/OS Cryptographic Services library is available on a CD-ROM, z/OS Collection, SK3T-4269. The CD-ROM online library collection is a set of unlicensed books for z/OS and related products that includes the IBM Library Reader. This is a program that enables you to view the BookManager files. This CD-ROM also contains the Portable Document Format (PDF) files. You can view or print these files with the Adobe Acrobat reader.

Internet Sources

The softcopy z/OS publications are also available for web-browsing and for viewing or printing PDFs using the following URL:

You can also provide comments about this book and any other z/OS documentation by visiting that URL. Your feedback is important in helping to provide the most accurate and high-quality information.

Accessing Licensed Books on the Web

z/OS licensed documentation in PDF format is available on the Internet at the IBM Resource Link site:
http://www.ibm.com/servers/resourcelink

Licensed books are available only to customers with a z/OS license. Access to these books requires an IBM Resource Link user ID, password, and z/OS licensed book key code. The z/OS order that you received provides a memo that includes your key code.

To obtain your IBM Resource Link user ID and password, logon to:
http://www.ibm.com/servers/resourcelink

To register for access to the z/OS licensed books:
1. Logon to Resource Link using your Resource Link user ID and password.
2. Select User Profiles located on the left-hand navigation bar.
5. Supply your key code where requested and select the Submit button.

If you supplied the correct key code you will receive confirmation that your request is being processed.

After your request is processed you will receive an e-mail confirmation.

Note: You cannot access the z/OS licensed books unless you have registered for access to them and received an e-mail confirmation informing you that your request has been processed.

To access the licensed books:
1. Logon to Resource Link using your Resource Link user ID and password.
2. Select Library.
3. Select zSeries.
4. Select Software.
5. Select z/OS.
6. Access the licensed book by selecting the appropriate element.
Summary of changes

Summary of Changes
for SC24-5901-08
z/OS Version 1 Release 11


New information

• Added SSL API:
  – gsk_attribute_set_tls_extension

• Added Certificate Management Services (CMS) APIs:
  – gsk_fips_state_query
  – gsk_fips_state_set
  – gsk_perform_kat
  – gsk_validate_certificate_mode

• Added Function Return Codes:
  – 445
  – 446
  – 447
  – 448
  – 449
  – 450
  – 707
  – 708

• Added Deprecated Function Return Code:
  – -105

• Added CMS Status Codes:
  – 03353067
  – 03353068
  – 03353069
  – 0335306A
  – 0335306B
  – 0335306C
  – 0335306D
  – 0335306E
  – 0335306F
  – 03353070
  – 03353071
  – 03353072
  – 03353073
  – 03353074

• Added Messages:
  – GSK01053A
  – GSK01054E

© Copyright IBM Corp. 1999, 2009
- GSK01057I
- Added GSK_CERT_VALIDATION_MODE in Appendix A. Environment Variables
- Added GSK_PROTOCOL_TLSV1_1 in Appendix A. Environment Variables
- Added new chapter - Chapter 4, “System SSL and FIPS 140-2,” on page 13

**Changed information**

- Changed SSL API:
  - gsk_attribute_get_buffer
  - gsk_attribute_get_cert_info
  - gsk_attribute_get_data
  - gsk_attribute_get_enum
  - gsk_attribute_set_buffer
  - gsk_attribute_set_callback
  - gsk_attribute_set_enum
  - gsk_attribute_set_numeric_value
  - gsk_environment_open
  - gsk_get_cert_by_label
  - gsk_get_cipher_suites
  - gsk_get_ssl_vector
  - gsk_secure_socket_init
  - gsk_secure_socket_misc
  - gsk_secure_socket_shutdown

- Changed Certificate Management Services (CMS) APIs:
  - gsk_add_record
  - gsk_change_database_password
  - gsk_change_database_record_length
  - gsk_construct_certificate
  - gsk_construct_renewal_request
  - gsk_construct_self_signed_certificate
  - gsk_construct_signed_certificate
  - gsk_create_certification_request
  - gsk_create_database
  - gsk_create_database_renewal_request
  - gsk_create_database_signed_certificate
  - gsk_create_renewal_request
  - gsk_create_self_signed_certificate
  - gsk_create_signed_certificate
  - gsk_create_signed_certificate_record
  - gsk_create_signed_certificate_set
  - gsk_create_signed_crl
  - gsk_create_signed_crl_record
  - gsk_decode_certificate
  - gsk_decode_certificate_extensions
  - gsk_decode_crl
  - gsk_decode_import_certificate
- gsk_decode_name
- gsk_decode_public_key
- gsk_delete_record
- gsk_dn_to_name
- gsk_encode_certificate_extensions
- gsk_encode_export_certificate
- gsk_encode_export_key
- gsk_encode_name
- gsk_encode_public_key
- gsk_export_key
- gsk_generate_key_agreement_pair
- gsk_generate_key_pair
- gsk_generate_key_parameters
- gsk_generate_secret
- gsk_get_cms_vector
- gsk_get_directory_certificates
- gsk_get_directory_crls
- gsk_import_certificate
- gsk_import_key
- gsk_make_encrypted_data_content
- gsk_make_encrypted_data_msg
- gsk_make_enveloped_data_content
- gsk_make_enveloped_data_content_extended
- gsk_make_enveloped_data_msg
- gsk_make_enveloped_data_msg_extended
- gsk_make_signed_data_content
- gsk_make_signed_data_content_extended
- gsk_make_signed_data_msg
- gsk_make_signed_data_msg_extended
- gsk_name_to_dn
- gsk_open_database
- gsk_open_database_using_stash_file
- gsk_open_keyring
- gsk_query_crypto_level
- gsk_read_encrypted_data_content
- gsk_read_encrypted_data_msg
- gsk_read_enveloped_data_content
- gsk_read_enveloped_data_content_extended
- gsk_read_enveloped_data_msg
- gsk_read_enveloped_data_msg_extended
- gsk_read_signed_data_content
- gsk_read_signed_data_content_extended
- gsk_read_signed_data_msg
- gsk_read_signed_data_msg_extended
- gsk_replace_record
- gsk_set_default_key
- gsk_sign_certificate
- gsk_sign_crl
- gsk_sign_data
- gsk_validate_certificate
- gsk_verify_certificate_signature
- gsk_verify_crl_signature
- gsk_verify_data_signature

- Changed Deprecated Secure Socket Layer APIs:
  - gsk_get_cipher_info
  - gsk_initialize
  - gsk_secure_soc_init
  - gsk_srb_initialize

- Changed Function Return Codes:
  - 8
  - 9
  - 402
  - 407
  - 412
  - 428
  - 601

- Changed Deprecated Function Return Codes:
  - -27
  - -35
  - -36

- Changed CMS Status Codes:
  - 03353003
  - 03353010
  - 03353034

- Updated gskyman
- Updated GSKSRVR started task

- Changed GSK_PROTOCOL_SSLV2 in Appendix A. Environment Variables
- Changed GSK_PROTOCOL_SSLV3 in Appendix A. Environment Variables
- Changed GSK_V3_CIPHER_SPECS in Appendix A. Environment Variables
- Changed GSK_V3_SESSION_TIMEOUT in Appendix A. Environment Variables
- Changed GSK_V3_SIDCACHE_SIZE in Appendix A. Environment Variables

Summary of Changes
for SC24-5901-07
z/OS Version 1 Release 10


New information
- Added Certificate Management Services (CMS) APIs:
  - gsk_create_database_renewal_request()
- gsk_create_database_signed_certificate()
- gsk_create_signed_certificate_record()
- gsk_create_signed_crl_record()

**Added Function Return Codes:**
- 444

**Added Deprecated Function Return Codes:**
- -104

**Added CMS Status Codes:**
- 03353064
- 03353065
- 03353066
- 03353068

**Added Function Return Codes:**
- 444

**The Key Management Menu displays the expiration date for the key database**

**Added support for certificates with SHA-224, SHA-256, SHA-384 and SHA-512 based signatures.**

**Changed information**

**Changed Certificate Management Services (CMS) APIs**
- gsk_construct_certificate()
- gsk_construct_renewal_request()
- gsk_construct_self_signed_certificate()
- gsk_construct_signed_certificate()
- gsk_create_certification_request()
- gsk_create_renewal_request()
- gsk_create_self_signed_certificate()
- gsk_create_signed_certificate()
- gsk_create_signed_certificate_set()
- gsk_create_signed_crl()
- gsk_generate_random_bytes()
- gsk_make_signed_data_content()
- gsk_make_signed_data_content_extended()
- gsk_make_signed_data_msg()
- gsk_make_signed_data_msg_extended()
- gsk_read_signed_data_content()
- gsk_read_signed_data_content_extended()
- gsk_read_signed_data_msg()
- gsk_read_signed_data_msg_extended()
- gsk_sign_certificate()
- gsk_sign_crl()
- gsk_sign_data()
- gsk_verify_certificate_signature()
- gsk_verify_crl_signature()
- gsk_verify_data_signature()

**Changed Message:**
- GSK01049A

- Changed CMS Status Codes:
  - 0335303D
  - 03353040

- Changed Function Return Codes:
  - 7
  - 109
  - 435
  - 440
  - 444

- Changed Deprecated Function Return Codes:
  - 18
  - 19
  - -15
  - -18
  - -19
  - -55

- Updated gskyyman

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

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

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

Summary of Changes for SC24-5901-06
z/OS Version 1 Release 9

This book contains information previously presented in z/OS System Secure Sockets Layer Programming, SC24-5901-05, which supports z/OS Version 1 Release 8.

New information
- Added Certificate Management Services (CMS) APIs:
  - gsk_construct_certificate()
  - gsk_construct_private_key_rsa()
  - gsk_construct_public_key_rsa()
  - gsk_construct_renewal_request()
  - gsk_construct_self_signed_certificate()
  - gsk_construct_signed_certificate()
  - gsk_decode_import_certificate()
  - gsk_decode_import_key()
  - gsk_encode_export_certificate()
  - gsk_encode_export_key()
- `gsk_encode_export_request()`
- `gsk_factor_private_key_rsa()`
- `gsk_factor_public_key_rsa()`
- `gsk_get_directory_enum()`
- `gsk_set_directory_enum()`
- `gsk_validate_hostname()`

- **CMS Status Codes:**
  - `0335304F`
  - `03353050`
  - `03353051`
  - `03353052`
  - `03353053`
  - `03353054`
  - `03353055`
  - `03353056`
  - `03353057`
  - `03353058`
  - `03353059`
  - `0335305A`
  - `0335305B`
  - `0335305C`
  - `0335305D`
  - `0335305E`
  - `0335305F`
  - `03353060`
  - `03353061`
  - `03353062`

- **Added Function Return Codes:**
  - `442`
  - `443`

- **Added Deprecated Function Return Codes:**
  - `-56`
  - `-57`

- **Added messages:**
  - `GSK00008E`
  - `GSK01051E`
  - `GSK01052W`

- **Added** `GSK_CRL_SECURITY_LEVEL` in Appendix A. Environment Variables
- **Added** `GSK_RNG_ALLOW_ZERO_BYTES` in Appendix A. Environment Variables

**Changed information**

- **Changed SSL APIs:**
  - `gsk_attribute_get_buffer()`
  - `gsk_attribute_get_enum()`
  - `gsk_attribute_set_buffer()`
- `gsk_attribute_set_callback()`
- `gsk_attribute_set_enum()`
- `gsk_environment_init()`
- `gsk_environment_open()`
- `gsk_get_cert_by_label()`
- `gsk_get_cipher_suites()`
- `gsk_get_ssl_vector()`
- `gsk_get_update()`
- `gsk_secure_socket_init()`
- `gsk_secure_socket_read()`
- `gsk_secure_socket_write()`

* Changed Certificate Management Services (CMS) APIs
  - `gsk_create_self_signed_certificate()`
  - `gsk_generate_random_bytes()`
  - `gsk_get_certificate_algorithms()`
  - `gsk_get_cms_vector()`
  - `gsk_get_default_key()`
  - `gsk_get_default_label()`
  - `gsk_get_record_by_label()`
  - `gsk_open_keyring()`
  - `gsk_query_crypto_level()`
  - `gsk_query_database_label()`
  - `gsk_validate_certificate()`
  - `gsk_validate_server()`

* Changed Deprecated APIs:
  - `gsk_get_dn_by_label()`
  - `gsk_initialize()`
  - `gsk_secure_soc_init()`
  - `gsk_secure_soc_read()`
  - `gsk_secure_soc_write()`

* Changed CMS Status Codes:
  - 0335303F
  - 03353040

* Changed Function Return Codes:
  - 6
  - 7
  - 8
  - 102
  - 109
  - 202
  - 417
  - 428
  - 436

* Changed Deprecated Function Return Codes:
  - 1
Summary of Changes for SC24-5901-05
z/OS Version 1 Release 8


New information
• SSL started task message GSK01049A
• SSL started task message GSK01050I
• Added support for wildcarding by adding JOBSUFFIX to the Component Trace input command
• Added Display Certificate File.
• CMS Status Codes:
  – 0335304D
  – 0335304E
• Added Certificate Management Services (CMS) APIs:
  – gsk_client_auth_nocert
  – gsk_copy_attributes_signers()
  – gsk_free_attributes_signers()
  – gsk_make_enveloped_data_content_extended()
  – gsk_make_enveloped_data_msg_extended()
  – gsk_make_signed_data_content_extended()
  – gsk_make_signed_data_msg_extended()
  – gsk_read_enveloped_data_content_extended()
- `gsk_read_enveloped_data_msg_extended()`
- `gsk_read_signed_data_content_extended()`
- `gsk_read_signed_data_msg_extended()`

- Added support for AES 128-bit encryption through CPACF hardware.
- Function Return Codes updated for SAF keyring:
  - `440` on page 450
- Deprecated Function Return Code `-55` on page 458

**Changed information**
- SSL started task message GSK01047 on page 477
- CMS Status Codes:
  - `03353024` on page 465
  - `0335302B` on page 466
  - `0335302E` on page 466
  - `0335303D` on page 468
  - `03353031` on page 467
- GSK_HW_CRYPTO environment variable updated to provide a bit setting to enable AES 128-bit exploitation in hardware.
- `gsksrvr display crypto` command updated to indicate whether DES, TDES, AES, SHA-1 and SHA-256 support is available in the CPACF hardware.
- Function Return Codes updated for SAF key ring:
  - `7` on page 445
  - `402` on page 447
  - `428` on page 448
  - `434` on page 449
  - `435` on page 449
- Deprecated Function Return Codes:
  - `-1` on page 454
  - `-3` on page 454
  - `-19` on page 456
- Added note about issuing `gsk_secure_socket_shutdown` call before a `gsk_secure_socket_close` call in Chapter 5, Migrating to the New SSL Interfaces and in the following APIs:
  - `gsk_secure_socket_close`
  - `gsk_secure_socket_init`
  - `gsk_secure_socket_read`
  - `gsk_secure_socket_misc`
  - `gsk_secure_socket_open`
  - `gsk_secure_socket_write`
- Altered APIs supporting signature algorithm `x509_alg_sha256WithRsaEncryption`:
  - `gsk_sign_certificate`
  - `gsk_sign_crl`
  - `gsk_sign_data`
  - `gsk_verify_certificate_signature`
  - `gsk_verify_crl_signature`
  - `gsk_verify_data_signature`
- Clarified the revoked_certificates parameter of `gsk_create_signed_crl()`.
• Updated gsk_local_threads to indicate it is the maximum number of threads.
• Updated gsk_stderr and gsk_stdout environment variables saying messages are displayed from the externally documented messages.

Delete information
• API gsk_seed_random_bytes

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

Starting with z/OS V1R2, you may notice changes in the style and structure of some content in this document -- for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our documents.

Summary of Changes
for SC24-5901-04
z/OS Version 1 Release 6


New information
• A suite of SSL V3 cipher specifications has been added to support Diffie Hellman key exchange during the SSL V3 and TLS handshakes. These ciphers will support both RSA and DSS certificates.
• System SSL now provides 64-bit support for its SSL and CMS APIs.

Note: The gsksrvr, gskkyman, and deprecated SSL APIs will remain 31-bit only.
• New Certificate Management API (gsk_validate_server()) has been added to validate a server certificate by verifying the host name associated with the server. The server certificate must contain the specified host name as either the common name (CN) element of the subject name or as a DNS entry for the subject alternate name.
• The gskkyman interface and the gskkyman command has been enhanced to support the creation of:
  – certificates containing key sizes of 4096
  – certificates that contain subject alternate name extensions
  – renewal certificates (expand the identification information within the certificate and to have an existing certificate renewed without having to create a new certificate request)
  – PKCS #7 file containing a chain of certificates.
• Reference APIs
  – "gsk_get_ssl_vector()" on page 82
• Certificate Management Services APIs
  – "gsk_create_database_renewal_request()" on page 140
  – "gsk_create_signed_certificate_set()" on page 158
  – "gsk_decode_private_key()" on page 177
  – "gsk_decode_public_key()" on page 178
  – "gsk_encode_private_key()" on page 192
  – "gsk_encode_public_key()" on page 193
  – "gsk_generate_key_agreement_pair()" on page 223
  – "gsk_generate_key_pair()" on page 224
  – "gsk_generate_key_parameters()" on page 226
SSL function return codes
- 702 on page 451
- 703 on page 451
- 704 on page 451
- 705 on page 451
- 706 on page 451

CMS Status codes
- 014CE022 on page 462
- 03353048 on page 469
- 03353049 on page 469
- 0335304A on page 469
- 0335304B on page 469
- 0335304C on page 469

SSL started task message GSK01048W on page 477

Environment variables: GSK_EXC_ABEND_DUMP, GSK_SSL_HW_DETECT_MESSAGE, GSK_SSL_ICSF_ERROR_MESSAGE and GSK_T61_AS_LATIN1 in Table 8 on page 481

Changed information
- APIs
  - “gsk_environment_open()” on page 70 has been updated to include Diffie-Hellman SSL V3 cipher specifications
  - “gsk_secure_soc_init()” on page 354 has been updated to include Diffie-Hellman SSL V3 cipher specifications
  - Value GSKCMS_APL_LVL2 added to “gsk_get_cms_vector()” on page 231
- gskkyman
  - The -lc option has been added to “gskkyman” on page 428
  - The database menu has been updated to include a create key parameter file option (see 377
  - The manage keys and certificates menu has been updated to include two new options:
    - Create a signed certificate and key
    - Create a certificate renewal request
- Environment variable GSK_V3_CIPHER_SPECS has been updated to include additional SSL V3 cipher specifications in Table 8 on page 481
- SSL Started Task Messages
  - GSK01037E on page 476

Deleted information
- Java Classes and JNI support removed

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

Starting with z/OS V1R2, you may notice changes in the style and structure of some content in this book--for example, procedures that have a different look and format. These changes are ongoing improvements to the consistency and retrievability of information in our books.
Chapter 1. Introduction

Secure Sockets Layer (SSL) is a communications protocol that provides secure communications over an open communications network (for example, the Internet). The SSL protocol is a layered protocol that is intended to be used on top of a reliable transport, such as Transmission Control Protocol (TCP/IP). SSL provides data privacy and integrity as well as server and client authentication based on public key certificates. Once an SSL connection is established between a client and server, data communications between client and server are transparent to the encryption and integrity added by the SSL protocol.

System SSL supports the SSL V2.0, SSL V3.0 and TLS (Transport Layer Security) V1.0 and TLS V1.1 protocols. TLS V1.1 is the latest version of the secure sockets layer protocol.

Note: The phrase SSL is used throughout to describe both the SSL and TLS protocols.

z/OS provides a set of SSL C/C++ callable application programming interfaces that, when used with the z/OS Sockets APIs, provide the functions required for applications to establish secure sockets communications.

In addition to providing the API interfaces to exploit the Secure Sockets Layer and Transport Layer Security protocols, System SSL is also providing a suite of Certificate Management APIs. These APIs give the capability to create/manage your own certificate databases, utilize certificates stored in key databases, key rings or tokens for purposes other than SSL and to build/process PKCS #7 standard messages.

In addition to providing APIs for applications to use for both SSL and certificate management support, System SSL also provides a certificate management utility called gskkyman. gskkyman allows for the management of certificates stored in a key database file or z/OS PKCS #11 token.

System SSL has been designed to meet the Federal Information Processing Standard - FIPS 140-2 Level 1 criteria. See Chapter 4, “System SSL and FIPS 140-2,” on page 13 for more information.

Software Dependencies

- Cryptographic Services System SSL (Function Modification Identifier (FMID) HCPT3B0)
  - System SSL Version 1 Release 11 is part of the Cryptographic Services Base element of z/OS. (The System SSL Base members are installed in the PDSE pdasename.SIEALNKE and PDS pdasename.SGSKSAMP.)
- Cryptographic Services Security Level 3 (FMID JCPT3B1)
  - When you order the Cryptographic Services Security Level 3 support, GSKSUS31, GSKSUS64, GSKC31F, GSKC64F, GSKS31F and GSKS64F are installed as members of the pdasename.SIEALNKE PDSE. pdasename.SIEALNKE is the PDSE in which the System SSL Cryptographic Services Base members are installed.
- Japanese (FMID JCPT3BJ)
  - Contains Japanese message text files for gskkyman utility. The gskmsgs.cat file is installed in the /usr/lpp/gskssl/lib/nls/msg/Ja_JP.IBM-939 directory.

This table lists the encryption capabilities for each FMID:

<table>
<thead>
<tr>
<th>Encryption Types/Key Sizes</th>
<th>Base Security Level FMID HCPT3B0</th>
<th>Security Level 3 FMID JCPT3B1</th>
</tr>
</thead>
<tbody>
<tr>
<td>512 bit keys</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1024 bit keys</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2048 bit keys</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
### Table 1. FMID Encryption Capabilities (continued)

<table>
<thead>
<tr>
<th>Encryption Types/Key Sizes</th>
<th>Base Security Level</th>
<th>Security Level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FMID HCPT3B0</td>
<td>FMID JCPT3B1</td>
</tr>
<tr>
<td>4096 bit keys</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1 - SSL V2.0 RC4 US</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2 - SSL V2.0 RC4 Export</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3 - SSL V2.0 RC2 US</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 - SSL V2.0 RC2 Export</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>6 - SSL V2.0 DES 56-Bit</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>7 - SSL V2.0 Triple DES US</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>01 - SSL V3.0 NULL MD5</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>02 - SSL V3.0 NULL SHA-1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>03 - SSL V3.0 RC4 MD5 Export</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>04 - SSL V3.0 RC4 MD5 US</td>
<td></td>
<td></td>
</tr>
<tr>
<td>05 - SSL V3.0 RC4 SHA-1 US</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06 - SSL V3.0 RC2 MD5 Export</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>09 - SSL V3.0 DES SHA-1 Export</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>0A - SSL V3.0 Triple DES SHA-1 US</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>0C - SSL V3.0 DES SHA-1 (fixed Diffie-Hellman) DSS</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>0D - SSL V3.0 Triple DES SHA-1 (fixed Diffie-Hellman) DSS</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>0F - SSL V3.0 DES SHA-1 (fixed Diffie-Hellman) RSA</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>10 - SSL V3.0 Triple DES SHA-1 (fixed Diffie-Hellman)</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>12 - SSL V3.0 DES SHA-1 (ephemeral Diffie-Hellman) DSS</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>13 - SSL V3.0 Triple DES SHA-1 (ephemeral Diffie-Hellman) DSS</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>15 - SSL V3.0 DES SHA-1 (ephemeral Diffie-Hellman) RSA</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>16 - SSL V3.0 Triple DES SHA-1 (ephemeral Diffie-Hellman) RSA</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>2F - SSL V3.0 AES 128 Bit SHA-1 RSA</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>30 - SSL V3.0 AES SHA-1 (fixed Diffie Hellman) DSS</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>31 - SSL V3.0 AES SHA-1 (fixed Diffie-Hellman) RSA</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>32 - SSL V3.0 AES SHA-1 (ephemeral Diffie-Hellman) DSS</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>33 - SSL V3.0 AES SHA-1 (ephemeral Diffie-Hellman) RSA</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>35 - SSL V3.0 AES 256-Bit SHA-1 RSA</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>36 - SSL V3.0 AES SHA-1 (fixed Diffie-Hellman) DSS</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>37 - SSL V3.0 AES SHA-1 (fixed Diffie-Hellman) RSA</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>38 - SSL V3.0 AES SHA-1 (ephemeral Diffie-Hellman) DSS</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>39 - SSL V3.0 AES SHA-1 (ephemeral Diffie-Hellman) RSA</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

### Installation Information

System SSL is part of the System SSL Cryptographic Services Base element of z/OS. If you choose to install the z/OS Release 11 Server Pack, you will not need to install the System SSL Cryptographic Services Base element separately. If you choose the z/OS PDO, you can install the System SSL Cryptographic Services Base element using SMP/E. The [z/OS Program Directory](https://www.ibm.com/support/docview.wss?uid=swg21363315) contains the directions for installing the System SSL Cryptographic Services Base element using SMP/E.
System SSL Parts Shipped in HFS

- `/usr/lpp/gskssl/include`
  Contains the header files, `gskssl.h`, `gsktypes.h` and `gskcms.h`, which declare structures and constants that are used by the System SSL and Certificate Management interfaces.

- `/usr/lpp/gskssl/examples`
  Contains sample client/server files as well as a `display_certificate` sample program.

- `/usr/lpp/gskssl/lib`
  Contains GSKSSL.x for APIs exported by the GSKSSL DLL, GSKSSL64.x for APIs exported by the GSKSSL64 DLL, GSKCMS31.x for APIs exported by the GSKCMS31 DLL, and GSKCMS64.x for APIs exported by the GSKCMS64 DLL. You use GSKSSL.x and GSKCMS31.x when you linkedit a 31-bit program that uses System SSL and you use GSKSSL64.x and GSKCMS64.x when you linkedit a 64-bit program that uses System SSL.

- `/usr/lpp/gskssl/lib/nls/msg/En_US.IBM-1047`
  Contains the English gskmsgs.cat message catalog file.

- `/usr/lpp/gskssl/lib/nls/msg/Ja_JP.IBM-939`
  Contains the Kanji gskmsgs.cat message catalog file.

- `/usr/lpp/gskssl/bin`
  Contains the `gskkyman` and `gsktrace` utilities.

System SSL Parts Shipped in PDS and PDSE

- `pdsname.SIEALNKE` PDSE contains members GSKSSL, GSKCMS31, GSKSRBRD, GSKSRBW T, GSKKYMAN, GSKSCTSS, GSKSRVR, GSKCMS64, GSKS31, GSKS64, GSKC31, GSKC64 and GSKSSL64 when the base FMID HCPT3B0 is installed. When JCPT3B1 is installed, members GSKSUS31, GSKS31F, GSKS64F, GSKC31F, GSKC64F and GSKSUS64 are also in the PDSE.

- `pdsname.SIEAHDR` PDS contains header files GSKSSL, GSKCMS and GSKTYPES.

- `pdsname.SIEASID` PDS contains side files GSKSSL, GSKCMS31, GSKSSL64 and GSKCMS64 when the base FMID HCPT3B0 is installed.

- `pdsname.SGSKSAMP` PDS contains members GSKMSGXT, GSKRACF, GSKSRVR and GSKWTR.

- `pdsname.SIEAMIGE` PDS contains member GSKSCTFT.

`pdsname` and `pdsasename` are the names determined during installation. You will need to know the name of this PDS or PDSE when you identify the STEPLIB in the runtime steps. Refer to the z/OS Program Directory for information about installing the System SSL.

Notes:

1. The DLLs are shipped in PDSE form so the DLLs can be called from HFS-based as well as PDSE-based programs.
2. The DLLs are not placed in SYS1.LPALIB during installation. The DLLs cannot be added to an LPALSTxx member since PDSE datasets are not supported in LPALSTx. The DLLs can be added to the dynamic LPA by adding them to a PROGxx member.
3. The DLL cannot be added to the LPA if System SSL is to be utilized in FIPS mode.

System SSL has been designed to meet the National Institute of Standards and Technology (NIST) FIPS 140-2 Level 1 criteria. For more information on enabling applications and running System SSL FIPS enabled applications see Chapter 4, “System SSL and FIPS 140-2,” on page 13.
Chapter 2. How System SSL Works for Secure Socket Communication

System SSL supports both the TLS (Transport Layer Security) and SSL (Secure Sockets Layer) protocols. Before you start writing your application, let's look at how System SSL works.

Note: The phrase *SSL* is used throughout to describe both the SSL and TLS protocols.

The SSL protocol begins with a “handshake”. During the handshake, the client authenticates the server, the server optionally authenticates the client and the client and server agree on how to encrypt and decrypt information. In addition to the “handshake”, SSL also defines the format used to transmit encrypted data.

X.509 (V1, V2 or V3) certificates are used by both the client and server when securing communications using System SSL. The client must verify the server’s certificate based on the certificate of the Certificate Authority (CA) that signed the certificate or based on a self-signed certificate from the server. The server must verify the client’s certificate (if requested) using the certificate of the CA that signed the client’s certificate. The client and the server then use the negotiated session keys and begin encrypted communications.

The SSL protocol runs above the TCP/IP and below higher-level protocols such as HTTP. It uses TCP/IP on behalf of the higher-level protocols.

The capabilities of SSL address several fundamental concerns about communication over the Internet and other TCP/IP networks:

**SSL server authentication** allows a client application to confirm the identity of the server application. The client application through SSL uses standard public-key cryptography to verify that the server’s certificate and public key are valid and has been signed by a trusted certificate authority (CA) that is known to the client application.

**SSL client authentication** allows a server application to confirm the identity of the client application. The server application through SSL uses standard public-key cryptography to verify that the client’s certificate and public key are valid and has been signed by a trusted certificate authority (CA) that is known to the server application.

**An encrypted SSL connection** requires all information being sent between the client and server application to be encrypted. The sending application is responsible for encrypting the data and the receiving application is responsible for decrypting the data. In addition to encrypting the data, SSL provides message integrity. Message integrity provides a means to determine if the data has been tampered with since it was sent by the partner application.

Using System SSL on z/OS

System SSL provides programming interfaces to write both client and server applications. These programming interfaces provide functionality associated with either the SSL environment layer or secure socket connection layer. The SSL environment layer defines the general attributes of the environment, such as the key database file name, stash file name and session timeout. The secure socket connection layer defines the attributes associated with each secure connection being established, such as the file descriptor and certificate label. The SSL application program must first create the SSL environment layer. Once the environment is created, one or more instances of the secure socket connection layer can be associated with the SSL environment. Each of these secure socket connections can be established and closed independently of each other.
How System SSL Works

Each layer has four general function calls:
- open
- attribute_set
- initialize
- close

In addition, the secure socket connection layer has read and write function calls for reading and writing secure data between the two SSL enabled applications.

The open function calls return a handle (an environment handle or a secure socket connection handle) that must be passed back as a parameter on subsequent function calls. An instance of a secure socket connection handle is associated with an environment by passing the environment handle as a parameter on the `gsk_secure_socket_open()` call. The `gsk_secure_socket_open()` function is completely thread safe. Invocations to the `gsk_secure_socket_open()` function can be issued from different threads within an environment. Read and write functions are full-duplex, so asynchronous read and write function calls can be performed from different threads for a given secure socket connection. However, there can only be one read and one write call in progress at one time for any secure socket connection handle.

For every open, there must be a corresponding close.

In addition to these functions, various `gsk_attribute_set ...()` and `gsk_attribute_get...()` functions exist to define and retrieve attributes values associated with either the environment or secure socket connection layers. The syntax of these function calls is the same for both the environment and the secure socket connection layers. The target for the set/get function is determined by the handle specified on the function call.

System SSL Application Overview

Figure 1 on page 8 describes the basic structure of the elements needed in your System SSL source program.

Whether writing a server or client applications, the initial steps are the same. First, an SSL environment must be established with these function calls:

**gsk_environment_open()**
This is the first function call. It returns an environment handle that is used in all subsequent function calls. It also obtains storage and sets default values for all internal variables and picks up the values specified in system environment variables that override the built-in defaults.

**gsk_attribute_set...()**
One or more of these function calls are issued to set attribute values for the environment.

**gsk_environment_init()**
After you have set all variables, issue this function call to complete the initialization of the SSL environment. Once you have done this, you can open and close SSL connections.

At this point the client and server sides diverge. The server side sets up a listen environment. The listen environment is established by obtaining a socket descriptor through the `socket()` call and the activation of a connection through the `bind()`, `listen()` and `accept()` socket calls. Once the listen environment is established, the server waits for notification that a secure socket connection is requested and issues these System SSL API function calls:

**gsk_secure_socket_open()**
This function call reserves a handle in which to store information for initializing each secure socket. Default values for each SSL connection are set from the environment.
How System SSL Works

**gsk_attribute_set...()**
These function calls set attribute values for this particular SSL connection. These values could include the socket file descriptor, ciphers, protocol, and application-supplied callback routines.

**gsk_secure_socket_init()**
For each connection to be started, the application must issue this function call to complete the initialization of the SSL connection and to run the SSL handshake protocol. The SSL handshake is a function of the system SSL support.

**gsk_secure_socket_read()**
One or more read function calls is issued until the inbound data flow is complete. The number of calls is purely application-dependent.

**gsk_secure_socket_write()**
One or more write function calls is issued until all appropriate data is sent to the partner. Reads and writes may be alternated as defined by the application protocol until the data flow is complete.

**gsk_secure_socket_close()**
This function call frees all the resources used for the SSL connection.

All of the SSL API function calls are thread-safe. This is particularly useful on the server side, since each connection can be run on its own thread, simplifying application design. See the sample client/server program shipped with z/OS System SSL, for an illustration of multi-threaded application.

The client application then opens a connection to the server through the `socket()` and `connect()` calls and issues these System SSL API function calls:

**gsk_secure_socket_open()**
This function call reserves a handle in which to store information for initializing each secure socket.

**gsk_attribute_set...()**
These function calls set values for this particular SSL connection. These values could include the socket file descriptor, ciphers, protocol, and application-supplied callback routines.

**gsk_secure_socket_init()**
For each connection to be started, the application must issue this function call to complete the initialization of the SSL connection and to run the SSL handshake protocol. The SSL handshake is a function of the System SSL support.

**gsk_secure_socket_write()**
One or more write function calls are issued until the outbound data flow is complete. The number of calls is purely application-dependent.

**gsk_secure_socket_read()**
One or more read function calls are issued until all appropriate data is received from the partner. Writes and reads may be alternated as defined by the application protocol until the data flow is complete.

**gsk_secure_socket_close()**
This function call frees all the resources used for the SSL connection.

For both client and server applications, when the application is ready to end and all **gsk_secure_socket_close()** functions have completed, destroy the sockets through the `close()` call and issue the **gsk_environment_close()** function call to close the SSL environment and return resources to the operating system.

**Note:** The **skread** and **skwrite** routines are the routines responsible for sending and receiving data from the socket. They are invoked by the **gsk_secure_socket_init()**, **gsk_secure_socket_read()** and **gsk_secure_socket_write()** functions.
How System SSL Works

In addition to using the previous SSL programming interfaces in an application, an application is not complete until a key database is available for use by the SSL application. The key database contains certificate information and can be an HFS file built and managed using the `gskkyman` utility, a RACF® key ring or a z/OS PKCS #11 token. For more information about key databases, refer to Chapter 10, "Certificate/Key Management," on page 373.

![Figure 1. Sockets Programming Model Using System SSL](image-url)
Chapter 3. Using Hardware Cryptographic Features with System SSL

System SSL uses the Integrated Cryptographic Service Facility (ICSF) if it is available. ICSF provides hardware cryptographic support which will be used instead of the System SSL software algorithms. System SSL checks for the hardware support during its runtime initialization processing and will use the hardware support if available.

In order for System SSL to use the hardware support provided through ICSF, the ICSF started task must be running prior to the application and the application userid must be authorized to the appropriate resources in the RACF CSFSERV class (when the class is defined), either explicitly or through a generic resource profile. See Table 2 and Table 3 for supported crypto algorithms and required RACF classes for each hardware processor. In addition to the CSFSERV class, the application userid needs access to the RACF CSFKEYS class when SAF key rings are being used and the application's certificate keys are stored in ICSF. For more information on access to CSFKEYS, see the RACDCERT command in z/OS Security Server RACF Command Language Reference.

If a severe ICSF error occurs during a cryptographic operation, System SSL will stop using the hardware support and will revert to using the software algorithms when applicable. In this event, hardware failure notification will be available through the SSL Started Task or SSL trace output, if either facility is enabled. The SSL Started Task will output an error message to the console on the first occurrence of the hardware failure and to the system log on any subsequent events. A message showing the failing encryption algorithm will appear in the system log only. Any future cryptographic operations for the current SSL application that attempt to use this algorithm will be performed in software.

System SSL will also take advantage of the CP Assist for Cryptographic Function (CPACF) when available. CPACF is a set of cryptographic instructions available on all CPs of a z990, z890, z9 and z10 processors. The SHA-1 algorithm is always available. The SHA-224 and SHA-256 algorithms are available on the z9 and z10 processors. The SHA-384 and SHA-512 algorithms are available on z10 EC and z10 BC.

CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement, feature 3863, provides for clear key DES and TDES instructions. This feature also includes clear key AES for 128-bit keys on z9 and z10 processors, and clear key AES for 256-bit keys on z10 processors only. System SSL does not require ICSF to use the CPACF. System SSL calls the CPACF instructions directly.

Table 2 describes the cryptographic function exploited through ICSF and the required CSFSERV resource class accesses. For more information, refer to Controlling Who Can Use Cryptographic Keys and Services in z/OS Cryptographic Services ICSF Administrator’s Guide.

| Table 2. Hardware cryptographic functions exploited by System SSL |
|------------------|------------------|------------------|
| **Function**     | **z800, z900**   | **z890, z990, z9 EC, z9 BC, z10 EC and z10 BC** |
| DES              | CSFCKI, CSFDEC, CSFENC |                      |
| TDES             | CSFCKM, CSFDEC, CSFENC |                      |
| PKA (RSA) Decrypt| CSFPKD            | CSFPKD            |
| PKA (RSA) Encrypt| CSFPKE            | CSFPKE            |
| Digital Signature Generate | CSFPKI, CSFDSG | CSFPKI, CSFDSG |
| Digital Signature Verify | CSFDSV          | CSFDSV            |

Table 2 describes the hardware cryptographic functions that will be exploited by System SSL.
Using Hardware Cryptographic Features with System SSL

Note: Encrypt/decrypt using the CCF is dependent on the size of the data. For DES, when the data is less than or equal to 256 bytes, DES is performed in software. For TDES, when the data is less than or equal to 64 bytes, TDES is performed in software.

To exploit 4096-bit RSA keys in the hardware, you need z9 or higher with feature 0863 installed with the latest Crypto Express2 Coprocessor with microcode level MCL006-MCL009 or higher.

System SSL contains software implementations for algorithms in Table 3.

Table 3. Hardware cryptographic functions used by System SSL

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>z800, z900</th>
<th>z890, z990</th>
<th>z9 EC, z9 BC, z10 EC and z10 BC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DES</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>TDES</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>AES 128-bit</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>AES 256-bit</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>SHA-1</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>SHA-2 (SHA-224)</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>SHA-2 (SHA-256)</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>SHA-2 (SHA-384)</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHA-2 (SHA-512)</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PKA (RSA) Decrypt</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>PKA (RSA) Encrypt</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Digital Signature Generate</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Digital Signature Verify</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

System SSL handshake processing utilizes both the RSA and digital signature functions that are very expensive functions when performed in software. For installations that have high volumes of SSL handshake processing, utilizing the capabilities of the hardware will provide maximum performance and throughput. For example, on a z9 EC, z9 BC, z10 EC and z10 BC, having both CEX2C and CEX2A will result in the maximum clear key RSA and digital signature processing being done in hardware.

For installations that are more concerned with the transfer of encrypted data than SSL handshakes, moving the encrypt/decrypt processing to hardware will provide maximum performance. The encryption algorithm is determined by the SSL cipher value. To utilize hardware, the cipher's symmetric algorithm must be available in hardware. For example, on z9 EC, z9 BC, z10 EC and z10 BC, an application encrypting/decrypting data using the symmetric algorithm TDES would benefit from the processing being done in the hardware (CPACF).
For maximum performance and throughput, it is recommended that hardware be used for both the SSL handshake and data encrypt/decrypt.

For information on the types of hardware cryptographic features supported by ICSF, refer to \textit{z/OS Cryptographic Services ICSF Overview}. For information on configuring and using ICSF, refer to \textit{z/OS Cryptographic Services ICSF Administrator's Guide} and \textit{z/OS Cryptographic Services ICSF System Programmer's Guide}.

Several products use System SSL. Please check the specific product publications to see if there is information on System SSL and ICSF considerations.

Note that access to ICSF cryptographic services can be controlled by the z/OS Security Server (RACF). For further information, refer to the topic about controlling who can use cryptographic keys and services in \textit{z/OS Cryptographic Services ICSF Administrator's Guide}. 

\textbf{Using Hardware Cryptographic Features with System SSL}
Chapter 4. System SSL and FIPS 140-2

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 which specify that cryptographic-based security systems are to be used to provide protection for sensitive or valuable data.

The objective of System SSL is to provide the capability to execute securely in a mode that has been designed to meet the NIST FIPS 140-2 Level 1 criteria. To this end, System SSL can run in either 'FIPS mode' or 'non-FIPS mode'. System SSL by default runs in 'non-FIPS' mode.

To meet the FIPS 140-2 Level 1 criteria, System SSL, when executing in FIPS mode, is more restrictive with respect to cryptographic algorithms, protocols and key sizes that can be supported.

Algorithms and key sizes

When executing in FIPS mode, System SSL continues to take advantage of the CP Assist for Cryptographic Function (CPACF) when available. Cryptographic functions performed by ICSF-supported hardware when running in non-FIPS mode continue to be exploited when executing in FIPS mode apart from RSA signature generation which must be performed in software.

Table 4 summarizes the differences between FIPS mode and non-FIPS mode algorithm support. Hardware availability depends on the processor and CPACF feature installed. See Chapter 3, "Using Hardware Cryptographic Features with System SSL," on page 9 for more information about processors, CPACF algorithm availability and cryptographic card support.

Table 4. Algorithm support: FIPS and non-FIPS

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Non-FIPS</th>
<th>FIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC2</td>
<td>40 and 128</td>
<td></td>
</tr>
<tr>
<td>RC4</td>
<td>40 and 128</td>
<td></td>
</tr>
<tr>
<td>DES</td>
<td>56</td>
<td>X</td>
</tr>
<tr>
<td>TDES</td>
<td>168</td>
<td>X</td>
</tr>
<tr>
<td>AES</td>
<td>128 and 256</td>
<td>X</td>
</tr>
<tr>
<td>MD5</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>SHA-1</td>
<td>160</td>
<td>X</td>
</tr>
<tr>
<td>SHA-2</td>
<td>224, 256, 384 and 512</td>
<td>X</td>
</tr>
<tr>
<td>RSA</td>
<td>512–4096</td>
<td>X</td>
</tr>
<tr>
<td>DSA</td>
<td>512–1024</td>
<td></td>
</tr>
<tr>
<td>DH</td>
<td>512–2048</td>
<td>2048</td>
</tr>
</tbody>
</table>

When executing in FIPS mode, System SSL can only use certificates that use the algorithms and key sizes shown in Table 4. During X.509 certificate validation (including CA certificates from untrusted data sources, that is, certificates flowing during the SSL/TLS handshake), if an algorithm that is incompatible with FIPS mode is encountered, then the certificate cannot be used and is treated as not valid.
SSL/TLS Protocol

When executing in FIPS mode, applications are allowed to utilize the TLS V1.0 and TLS V1.1 protocols. SSL V2 and SSL V3 are not supported. The specification of SSL V2 and SSL V3 during setup of the SSL/TLS application is ignored. When executing in non-FIPS mode, the following cipher specifications string reflects the default order of suites supported:

050435363738392F303132330A1613100D0915120F0C0306020100

The algorithm restrictions (see Table 4 on page 13) result in the following default cipher specifications string in FIPS mode:

35363738392F303132330A1613100D

All cipher suites other than those in the preceding string are incompatible with the restrictions in Table 4 on page 13 and are therefore not supported while executing in FIPS mode.

If non-FIPS mode ciphers are specified, they are ignored during the TLS handshake processing.

For more information about ciphers and their two-character values, see “gsk_environment_open()” on page 70.

System SSL Module Verification Setup

System SSL requires Security Level 3 FMID (JCPT3B1) to be installed in order for enabled applications to execute in FIPS mode. Application enablement requires applications to invoke the gsk_fips_state_set API. For more information about the FIPS enablement API, see “gsk_fips_state_set()” on page 205.

The System SSL modules that form the FIPS 140-2 cryptographic boundary are signed using an IBM key during the build process. Once System SSL is installed, additional steps are required prior to the execution of a FIPS enabled System SSL application.

These steps involve:

- Defining specific RACF profiles to enable the verification of the System SSL module signature (added during the IBM module build process) when loaded by the z/OS loader.
- Defining specific RACF profiles and identifying which System SSL modules require signature verification.

Signature verification provides a method to ensure the System SSL modules remain unchanged from the time they were built, installed onto the system, and loaded into storage to be used by a FIPS enabled System SSL application.

The IBM key used to sign the System SSL modules is an RSA private key that belongs to an X.509 certificate signed by the STG Code Signing CA certificate. This certificate is shipped as a default CERTAUTH certificate in the RACF database under the label ‘STG Code Signing CA’.

Note: A sample clist, GSKRACF, is shipped in GSK.SGSKSAMP to assist you with the RACF commands needed to enable signature verification.

The following steps need to be followed by the system administrator to enable signature validation of the System SSL modules:

1. Mark the IBM root CA as TRUSTed if not already TRUSTed

   RACDCERT CERTAUTH LIST(LABEL('STG Code Signing CA'))
   RACDCERT CERTAUTH ALTER (LABEL('STG Code Signing CA')) TRUST
2. Create a key ring to hold the STG Code Signing CA certificate and connect the certificate to the key ring.

   The key ring needs to be owned by a valid RACF id and the key ring must be defined in uppercase.
   It is recommended that the id be an id of a security administrator. In our example the security administrator id is RACFADM.
   There can only be one designated signature verification key ring active at one time. If already active, add the CA certificate to the key ring. If not already active create the key ring. The recommended key ring name is CODE.SIGNATURE.VERIFICATION.KEYRING.
   - Determine if signature verification key ring already active:
     RLIST FACILITY IRR.PROGRAM.SIGNATURE.VERIFICATION
     The key ring will be present in the APPLICATION DATA field
   - Create key ring if needed and connect CA certificate:
     RACDCERT ID(RACFADM) ADDRING(CODE.SIGNATURE.VERIFICATION.KEYRING)
     RACDCERT ID(RACFADM) CONNECT(RING(CODE.SIGNATURE.VERIFICATION.KEYRING) CERTAUTH
     LABEL('STG Code Signing CA') USAGE(CERTAUTH))
   - If a key ring exists, verify the CA certificate is connected to the key ring. If not connected, connect the certificate:
     RACDCERT ID(RACFADM) LISTRING(CODE.SIGNATURE.VERIFICATION.KEYRING)
     RACDCERT ID(RACFADM) CONNECT(RING(CODE.SIGNATURE.VERIFICATION.KEYRING) CERTAUTH
     LABEL('STG Code Signing CA') USAGE(CERTAUTH))

3. Create the FACILITY class profile that tells RACF the key ring to use for module signature verification if it is not already defined.

   Note: Due to space constraints, the second command example appears on two lines. However, the command should be entered completely (on one line) on your system.
   RLIST FACILITY IRR.PROGRAM.SIGNATURE.VERIFICATION
   RDEFINE FACILITY IRR.PROGRAM.SIGNATURE.VERIFICATION APPLDATA('RACFADM/
   CODE.SIGNATURE.VERIFICATION.KEYRING')

4. Activate your profile changes in the FACILITY, DIGTCERT and/or DIGTRING classes if active and RACLISTed.

   SETROPTS RACLIST(FACILITY) REFRESH
   SETROPTS RACLIST(DIGTCERT, DIGTRING) REFRESH

5. Activate PROGRAM control, if not already active.

   SETROPTS WHEN(PROGRAM)

   Note: Installations that have not previously turned on program control, may encounter problems after issuing SETROPTS WHEN(PROGRAM). Program control is necessary for signature verification, hence installations must evaluate the impact of enabling program control for the first time.

6. Create the PROGRAM class profile that protects the program verification module IRRPVERS and specify its signature verification options.

   Note: Due to space constraints, the command appears on two lines. However, the command should be entered completely (on one line) on your system.
   RDEFINE PROGRAM IRRPVERS ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
   SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))

7. Refresh the PROGRAM class.

   SETROPTS WHEN(PROGRAM) REFRESH

8. Contact your system programmer to complete this step.
**System SSL and FIPS 140-2**

a. Notify your system programmer to initialize program signature verification by running the IRRVERLD program which loads and verifies the program verification module IRRPVERS. For programming information, refer to [z/OS Security Server RACF System Programmer's Guide](#). Check with your system programmer to ensure that IRRVERLD executed successfully. If it did not execute successfully, work with your system programmer to check error messages. Correct any setup errors and retry.

c. Do not define PROGRAM profiles for the System SSL modules until IRRVERLD executes successfully.

9. Create the PROGRAM class profiles to indicate the System SSL modules must be signed. The load should fail if the signature cannot be verified and auditing should occur for failure only. If your installation requires event logging for the signature verification, see the RALTER and RDEFINE commands in the [z/OS Security Server RACF Command Language Reference](#) for customizing the SIGAUDIT operand within the SIGVER segment.

**Note:** Due to space constraints, the command examples appear on two lines. However, the command should be entered completely (on one line) on your system.

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

RDEFINE PROGRAM GSKSSL64 ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
      SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))

RDEFINE PROGRAM GSKS31F ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
      SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))

RDEFINE PROGRAM GSKS64F ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
      SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))

RDEFINE PROGRAM GSKCMS31 ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
      SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))

RDEFINE PROGRAM GSKCMS64 ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
      SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))

RDEFINE PROGRAM GSKC31F ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
      SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))

RDEFINE PROGRAM GSKC64F ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
      SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))

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

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

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

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

10. Refresh the PROGRAM class.

```
SETROPTS WHEN(PROGRAM) REFRESH
```
Performance Guideline

RACF can use virtual lookaside facility (VLF) to cache signature verification data in order to improve the performance of signature verification of signed program objects. This in turn can improve the load time of the signed System SSL program objects. For more information on using VLF see “VLF considerations for program signature verification” in the documentation provided by z/OS Security Server RACF System Programmer's Guide.

Certificate Stores

To use FIPS mode, certificates can be stored in either a SAF key ring, PKCS #11 token, or a FIPS mode key database. All certificates in a certificate chain to be used by a FIPS enabled application must use algorithms and key sizes as specified in Table 4 on page 13.

SAF keyrings and PKCS #11 tokens

Provided a certificate and its signers chain use only valid algorithms and key sizes, then there are no changes required if using a SAF key ring or a PKCS #11 token. A SAF key ring or PKCS #11 token may contain certificates with keys sizes or algorithms that are not supported in FIPS mode as long as those certificates are never used while executing in FIPS mode. While executing in FIPS mode, if an attempt to use a certificate with unsupported key size or algorithms is made, then the process will fail. The corrective action is to either add/replace certificates with key sizes and algorithms that are valid in FIPS mode, or execute in non-FIPS mode.

gskkyman runs in non-FIPS mode when managing PKCS #11 tokens. It is therefore possible to add certificates/keys with algorithms or key sizes that are not supported if the PKCS #11 token is subsequently used while executing in FIPS mode.

Key database files

To use a key database in FIPS mode, it needs to be created as a FIPS mode database. Key databases created through gskkyman not explicitly specifying FIPS during creation, or created through an application not executing in FIPS mode, cannot be used by an application executing in FIPS mode. To create a FIPS mode key database using gskkyman, see “Creating, Opening and Deleting a Key Database File” on page 88. To create a FIPS mode key database using the Certificate Management Services API, the application must start in FIPS mode (see “gsk_fips_state_set()” on page 205).

The following are key points when using FIPS key databases:
- Only certificates that meet the requirements for FIPS (see Table 4 on page 13) can be added to a FIPS key database.
- A FIPS key database may only be modified if executing in FIPS mode. When opening an existing FIPS key database, gskkyman ensures that it is executing in FIPS mode. If an application modifies the key database via the Certificate Management Services (CMS) APIs, then it too must ensure it is executing in FIPS mode.
- A FIPS key database can be used in non-FIPS mode if it is opened for read only.
- A non-FIPS key database cannot be opened while executing in FIPS mode.

gskkyman automatically detects when a FIPS mode key database is opened, and executes in FIPS mode. This ensures that only certificates or certificate requests that meet the FIPS mode requirements in Table 4 on page 13 may be added to the key database.

Application Changes

In order to use System SSL in FIPS mode, application changes are required. By default, all applications that use System SSL execute in non-FIPS mode. The application needs to request System SSL execute in FIPS mode in the very early stages of interaction with the System SSL API. The application does this by invoking the function gsk_fips_state_set (see “gsk_fips_state_set()” on page 205). In order to set FIPS
mode, gsk_fips_state_set must be executed prior to all other System SSL functions with the exception of

gsk_get_cms_vector, gsk_get_ssl_vector and gsk_fips_state_query. It is possible to switch to non-FIPS

mode at a later time. It is not possible to switch from non-FIPS mode to FIPS mode at any time.

When executing in FIPS mode and a severe cryptographic problem is encountered, one of the following
return codes will be returned from the API executing at the time of failure. These return codes should be

treated as severe and the application should be terminated and restarted. If execution continues, all APIs
except for gsk_get_cms_vector, gsk_get_ssl_vector, gsk_fips_state_query and gsk_query_crypto_level will
fail:

- CMSERR_BAD_RNG_OUTPUT - Failure during random number generation
- GSK_ERR_RNG, GSK_ERROR_RNG - Failure during random number generation
- CMSERR_FIPS_KEY_PAIR_CONSISTENCY - Failure when generating either a RSA or DSA keypair
- CMSERR_KATPW_FAILED - Failure was encountered by the gsk_perform_kat API when performing
known answer tests against the System SSL cryptographic algorithms.
- CMSERR_SIGNVER_UNAVAIL - Failure was encountered when attempting a known answer test to
verify the R_PgmSignVer SAF callable service

The sample files (see Appendix B, “Sample C++ SSL Files,” on page 491) client.cpp and server.cpp
demonstrate the use of gsk_fips_state_set to set the application to run in FIPS mode. In both cases, the
gsk_fips_state_set function is invoked prior to any other System SSL function.

SSL Started Task

The System SSL started task (GSKSRVR) executes in non-FIPS mode as default. In order for the
GSKSRVR started task to execute in FIPS mode, environment variable GSK_FIPS_STATE must be
specified and set to GSK_FIPS_STATE_ON in the envar file in the GSKSRVR home directory. If the
GSKSRVR is unable to execute in FIPS mode (for example, the Level 3 FMID JCPT3B1 is not installed), it
will execute in non-FIPS mode after issuing message GSK01054E (see SSL Started Task Messages
(GSK01nnn)” on page 473).

Sysplex Session ID Cache

GSKSRVR must be running in FIPS mode in order to maintain Sysplex Session ID cache entries for SSL
server applications executing in FIPS mode. An SSL server application executing in FIPS mode will only
cache its session in the Sysplex Session cache provided GSKSRVR is also executing in FIPS mode. An
SSL server application executing in non-FIPS mode is able to cache its session in the Sysplex Session

cache if GSKSRVR is executing in either FIPS mode or non-FIPS mode.

An SSL server application executing in FIPS mode is only able to resume a Sysplex Session cached
session if it was for a session that executed in FIPS mode when the cache entry was created. Non-FIPS
SSL server applications can resume FIPS and non-FIPS sessions cached in the Sysplex Session cache.

SSL servers executing in non-FIPS mode on systems with a back-level GSKSRVR are able to resume
FIPS and non-FIPS sessions that are cached in the Sysplex Session cache by systems where the System
SSL started task is executing in FIPS mode.
Chapter 5. Writing and Building a z/OS System SSL Application

This topic describes how to write, build, and run a secure socket layer (SSL) application that uses the System SSL programming interfaces. You can write both client and server applications using the System SSL (TLS/SSL) programming interfaces.

In Release 2 of z/OS, a new set of functions were added that superseded some functions from previous System SSL releases. The functions that were superseded are referred to collectively as "the deprecated SSL interface". IBM recommends that new application programs do not use the deprecated SSL interface. For a complete list and descriptions of the recommended APIs, see Chapter 7, “API Reference,” on page 35. Information about the deprecated ones can be found in Chapter 9, “Deprecated Secure Sockets Layer APIs,” on page 345.

**Note:** When migrating from the deprecated SSL interface, the entire System SSL application must be migrated. The application must not contain a mixture of deprecated and superseding APIs.

In addition to writing the SSL applications, you must build a key database for the System SSL source program for a System SSL application to be complete. Refer to Chapter 10, “Certificate/Key Management,” on page 373 for details about creating and managing a key database.

Sample programs using the new APIs are shipped in /usr/lpp/gskssl/examples.

**Writing a System SSL Source Program**

The first step in creating a System SSL application is to write the source program using the new System SSL programming interfaces. Refer to Chapter 7, “API Reference,” on page 35 for a description of the format of the System SSL programming interfaces.

Prior to establishing a secure connection, SIGPIPE signals should be set to be ignored or a signal handler should be defined. TCP/IP functions can cause SIGPIPE signals. When the signal is ignored, TCP/IP reflects the signal as a EPIPE error for the TCP/IP functions.

**Create an SSL Environment**

For both the client and server System SSL programs, you must initialize the System SSL environment using the programming interfaces associated with the SSL environment layer.

```c
int gsk_environment_open();
```

Will define and obtain storage for the SSL environment and return an environment handle to be used on subsequent API invocations.

```c
int gsk_attribute_set...();
```

Sets environment attributes such as:

- The SSL protocol version to be used: SSL Version 2.0, SSL Version 3.0, TLS Version 1.0 and/or TLS Version 1.1.
- The key database to be used. (HFS key database file, SAF key ring or z/OS PKCS #11 token)
- The password for the key database. This can be specified directly by the application or through the use of a stashed password file. See Chapter 10, “Certificate/Key Management,” on page 373 for details about creating a stashed password file.

**Note:** When using SAF key rings or z/OS PKCS #11 tokens, the password and stash file must not be specified.
Writing and Building a z/OS System SSL Application

- The amount of time the SSL session identifier information is valid. By using already negotiated and agreed to SSL session identifier information, System SSL can reduce the amount of data exchanged during the SSL handshake that occurs during the `gsk_secure_socket_init()` call.

`gsk_environment_init()`

Initializes the SSL environment.

This example code illustrates how to call the environment layer programming interface from a client or server System SSL program. In this example, SSL Version 3.0 support is requested, /keyring/key.kdb is the key database that is used, the password for the key database is "password", and default values are taken for the remaining SSL environment variable attributes.

```c
    gsk_handle env_handle;
    int    rc;

    /* create the SSL environment */
    rc = gsk_environment_open(&env_handle);

    /* set environment attributes */
    rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_SSLV2, GSK_PROTOCOL_SSLV2_OFF);
        /* By default, SSL V2 protocol is set on */
    rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_SSLV3, GSK_PROTOCOL_SSLV3_ON);
    rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1, GSK_PROTOCOL_TLSV1_OFF);
        /* By Default, TLS V1.0 protocol is set on */
    rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1_1, GSK_PROTOCOL_TLSV1_1_OFF);
    rc = gsk_attribute_set_buffer(env_handle, GSK_KEYRING_FILE, "+/keyring/key.kdb",0);
    rc = gsk_attribute_set_buffer(env_handle, GSK_KEYRING_PW, "password",0);

    /* initialize environment */
    rc = gsk_environment_init(env_handle);
```

This example code illustrates how to create an SSL environment for a server System SSL program supporting SSL Version 2.0, SSL Version 3.0, TLS Version 1.0 and TLS Version 1.1.

```c
    gsk_handle env_handle;
    int    rc;

    /* create the SSL environment */
    rc = gsk_environment_open(&env_handle);

    /* set environment attributes */
    rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_SSLV2, GSK_PROTOCOL_SSLV2_ON);
    rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_SSLV3, GSK_PROTOCOL_SSLV3_ON);
    rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1, GSK_PROTOCOL_TLSV1_ON);
    rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1_1, GSK_PROTOCOL_TLSV1_1_ON);
    rc = gsk_attribute_set_buffer(env_handle, GSK_KEYRING_FILE, "+/keyring/key.kdb",0);
    rc = gsk_attribute_set_buffer(env_handle, GSK_KEYRING_PW, "password",0);

    /* initialize environment */
    rc = gsk_environment_init(env_handle);
```

**Note:** Once the environment is initialized, the environment attributes cannot be changed unless they are also attributes of the secure socket connection. In this case, they can be changed only for that connection. If changes are necessary to the environment, a new SSL environment can be created within the same process.

Once the System SSL program has successfully created the SSL environment, it must now perform the steps needed to allow the program to communicate with a peer program. The exact sockets and System SSL calls required to allow the program to communicate differ depending on whether the program is a client or a server.
System SSL Server Program

You can use these sockets and System SSL calls to enable a server program to communicate with a client program.

To create a stream socket to which client programs can connect, use this function call:

```c
int server_sock;
server_sock = socket(AF_INET, SOCK_STREAM, 0);
```

Now that the server program socket has been created, bind the socket to a port (for example, 1234) that is known to the client program using this function call:

```c
int rc;
int namelength;
struct sockaddr_in name;
nameLength = sizeof(name);
memset(&name, '\0', nameLength);
name.sin_family = AF_INET;
name.sin_port = 1234;
name.sin_addr.s_addr = INADDR_ANY;
rc = bind(server_sock, (struct sockaddr *)&name, nameLength);
```

To make the server program socket ready to listen for incoming connection request, use this function call:

```c
int rc;
rc = listen(server_sock, 5); /* allow max of 5 connections */
```

The server program is now ready to begin accepting connections from client programs. To accept connections, use these function calls:

```c
int client_sock;
int incomingNameLength;
struct sockaddr_in incomingName;

client_sock = accept(server_sock, (struct sockaddr *)&incomingName, &incomingNameLength);
```

After successfully accepting a connection from a client program, the server program must establish the secure socket connection which will result in the SSL handshake being performed. Once the handshake is completed, secure transfer of application data can be done. The secure socket connection will be established with these attribute values:

- The socket descriptor over which the communication is to occur.
- Certificate with label "ServerCertLabel"
- The type of handshake (for example, server) to be performed.
- The set of SSL protocol cipher specs to be allowed for the secure session. The cipher is selected by the System SSL Server program according to the System SSL Server's order of usage preference.
- The address of a routine to be called by System SSL to read data from the socket for the secure session.
- The address of a routine to be called by System SSL to write data on the socket for the secure session.

```c
gsk_handle soc_handle;
int rc;
gsk_iocallback local_io = {secureSocRecv, secureSocSend, NULL, NULL, NULL, NULL};

rc = gsk_secure_socket_open(env_handle, &soc_handle);
rc = gsk_attribute_set_numeric_value(soc_handle, GSK_FD, client_sock);
rc = gsk_attribute_set_buffer(soc_handle, GSK_KEYRING_LABEL, "ServerCertLabel", 0);
rc = gsk_attribute_set_buffer(soc_handle, GSK_SESSION_TYPE, GSK_SERVER_SESSION);
rc = gsk_attribute_set_buffer(soc_handle, GSK_V2_CIPHER_SPECS, "6321", 0);
```
Writing and Building a z/OS System SSL Application

The System SSL program should provide the function to send and receive data over the application socket. For more information, see "I/O Routine Replacement" on page 27. Use these function calls, send and recv, to send and receive the application data.

```c
int secureSocRecv(int fd, void *data, int len, char *user_data) {
    return( recv( fd, data, len,0 ));
}
int secureSocSend(int fd, void *data, int len, char *user_data) {
    return(send( fd, data, len,0 ));
}
```

After the server program has successfully called gsk_secure_socket_init(), it can now read and write data securely over the application socket. To read application data from the application socket, use this code:

```c
int rc;
int buffer_length;
int length_read;
char *data_buffer;
rc = gsk_secure_socket_read(soc_handle, data_buffer, buffer_length, &length_read);
```

To write application data over the application socket, use this code:

```c
int rc;
int buffer_length;
int length_written;
char *data_buffer;
rc = gsk_secure_socket_write(soc_handle, data_buffer, buffer_length, &length_written);
```

Once the server program is finished using the application socket to securely send and receive data, it must free all of the System SSL resources for the SSL session and close the socket. To free the System SSL resource for the SSL session, use the gsk_secure_socket_close() call:

```c
gsk_secure_socket_close(&soc_handle);
```

To free the resources used by the SSL environment, use the gsk_environment_close() call:

```c
gsk_environment_close(&env_handle);
```

Finally, to close the application socket, use this function call:

```c
int rc;
rc = close(client_sock);
```

System SSL Client Program

The sockets and System SSL calls a client program uses are very similar to the calls the server program uses. Rather than accepting connections like a server program, a client program connects to the server program.

To create a stream socket that the client program can use to connect to the server, use this function call:

```c
int sock;
sock = socket(AF_INET, SOCK_STREAM,0);
```

Now that the client program socket has been created, connect the socket to the server program port using this function call:
After successfully connecting to the server program, the client program must establish the secure socket connection. This connection will cause the SSL handshake to be performed. Once the handshake is complete, secure communication of the application data can be done. This example code establishes the connection using these attribute values:

- The socket descriptor over which the communication is to occur.
- Certificate with label "THELABEL"
- The type of handshake (client) to be performed.
- The set of SSL protocol cipher specs to be allowed for the secure session in client-preferred order.

**Note:** Although the client is allowed to specify a preference order, an SSL server may or may not honor the preference.

- The address of a routine to be called by System SSL to read data from the socket for the secure session.
- The address of a routine to be called by System SSL to write data on the socket for the secure session.

```c
int rc;

int namelength;
struct sockaddr_in name;
char *ServeHostName;

nameLength = sizeof(name);
memset(&name, '\0', nameLength);
name.sin_family = AF_INET;
name.sin_port = 1234;
name.sin_addr.s_addr = ServerHostName;
rc = connect(sock, (struct sockaddr *)&name, nameLength);
```

After successfully connecting to the server program, the client program must establish the secure socket connection. This connection will cause the SSL handshake to be performed. Once the handshake is complete, secure communication of the application data can be done. This example code establishes the connection using these attribute values:

- The socket descriptor over which the communication is to occur.
- Certificate with label "THELABEL"
- The type of handshake (client) to be performed.
- The set of SSL protocol cipher specs to be allowed for the secure session in client-preferred order.

**Note:** Although the client is allowed to specify a preference order, an SSL server may or may not honor the preference.

- The address of a routine to be called by System SSL to read data from the socket for the secure session.
- The address of a routine to be called by System SSL to write data on the socket for the secure session.

```c
int rc;

gsk_handle soc_handle;
gsk_iocallback local_io = {secureSocRecv, secureSocSend, NULL, NULL, NULL, NULL};

rc = gsk_secure_socket_open(env_handle, &soc_handle);
rc = gsk_attribute_set_numeric_value(soc_handle, GSK_FD, client_sock);
rc = gsk_attribute_set_buffer(soc_handle, GSK_KEYRING_LABEL, "THELABEL",0);
rc = gsk_attribute_set_buffer(soc_handle, GSK_V2_CIPHER_SPECS, "6321",0);
rc = gsk_attribute_set_buffer(soc_handle, GSK_V3_CIPHER_SPECS, "0906030201",0);
rc = gsk_attribute_set_buffer(soc_handle, GSK_IO_CALLBACK, &local_io);
rc = gsk_secure_socket_init(soc_handle);
```

The System SSL program should provide the function to send and receive data over the application socket. For more information, see "I/O Routine Replacement" on page 27. Use these function calls, `send` and `recv`, to send and receive the application data.

```c
int secureSocRecv(int fd, void *data, int len, char *user_data) {
    return( recv( fd, data, len,0 ));
}

int secureSocSend(int fd, void *data, int len, char *user_data) {
    return(send( fd, data, len,0 ));
}
```

After the client program has successfully called `gsk_secure_socket_init()`, it can now read and write data securely over the application socket. To read application data from the application socket, use this code:

```c
int rc;
int buffer_length;
int length_read;
char *data_buffer;

rc = gsk_secure_socket_read(soc_handle, data_buffer, buffer_length, &length_read);
```
Writing and Building a z/OS System SSL Application

To write application data over the application socket, use this code:

```c
int rc;
int buffer_length;
int length_written;
char *data_buffer;

c = gsk_secure_socket_write(soc_handle, data_buffer, buffer_length, &length_written);
```

Once the client program is finished using the application socket to securely send and receive data, it must free all of the System SSL resources for the SSL session and close the socket.

To free the System SSL resource for the SSL session, use the `gsk_secure_socket_close()` call:

```c
gsk_secure_socket_close(&soc_handle);
```

To free the resources used by the SSL environment, use the `gsk_environment_close()` call:

```c
gsk_environment_close(&env_handle);
```

Finally, to close the application socket, use this function call:

```c
int rc;
rc=close(sock);
```

Building a z/OS System SSL Application

1. Write the System SSL source program (see “Writing a System SSL Source Program” on page 19).
2. Compile your System SSL source program using the DLL compiler option.
3. Include the `/usr/lib/GSKSSL.x` sidedeck in the prelink or bind step input.
   
   If using the Certificate Management APIs, include either the `/usr/lib/GSKCMS31.x` or `/usr/lib/GSKCMS64.x` sidedeck in the prelink or bind step input.
4. Build a key database file or z/OS PKCS #11 token using the `gskkyman` utility or create a RACF key ring using the RACDCERT command. The name of the key database file or RACF key ring must match the name you specified as the GSK_KEYRING_FILE on the `gsk_attribute_set_buffer()` API. You need the name of the key database file or RACF key ring and for key database files either the password associated with the key file or the stash file name. The password must match the password specified on GSK_KEYRING_PW on the `gsk_attribute_set_buffer()` API or must be set to NULL if using a RACF key ring. Refer to Chapter 10, “Certificate/Key Management,” on page 373 for information on how to create a key database file, RACF key ring or z/OS PKCS #11 token.

Running a z/OS System SSL Application

After successfully writing and building the System SSL application and creating the key database, you can run the System SSL application. To run the application follow these steps:

1. Ensure that `pdsename.SIEALNKE`, the PDS that contains the System SSL DLLs, is in the MVS search order. If it's not in the linklist or LPA, you can use the STEPLIB DD statement in your JCL or the STEPLIB environment variable in the shell. For example, in the z/OS shell, issue this command:
   
   ```bash
   export STEPLIB=$STEPLIB:pdsename.SIEALNKE
   ```
2. Ensure that the key database file, SAF key ring or z/OS PKCS #11 token is accessible to the System SSL application.
3. Run the System SSL application.

In addition, please note:

- SSL applications must be run from within a POSIX environment.
- Once SSL applications have called `gsk_initialize` or `gsk_environment_open`, they cannot destroy the LE environment.
- SSL applications must call SSL APIs from a C program, as they are C APIs.
Non-Blocking I/O

Applications wishing to communicate securely to one another may establish a secure connection. Each application opens a socket and attempts to establish an SSL connection. After an SSL connection has been established, the applications may now use the socket to exchange data securely. The default (blocking) mode of a socket requires an application attempting to read or write to the socket to block until all expected data has been received. This blocking may not be desirable since no other processing may occur while the application is waiting for a read or write to complete. One solution to this problem is the use of non-blocking sockets.

When a socket is setup as non-blocking, reads and writes to the socket do not cause the application to block and wait. Instead the read or write function will read/write only the data currently available (if any). If the entire read/write is not completed, a status indicator is returned. The application may retry the read/write later.

Non-Blocking Socket Primer

When a server wishes to communicate with clients via a socket, these routines are used:

Table 5. Server Communicating with Clients Via a Socket

<table>
<thead>
<tr>
<th>Routine</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) socket()</td>
<td>Create a socket</td>
</tr>
<tr>
<td>2) bind()</td>
<td>Register the socket</td>
</tr>
<tr>
<td>3) listen()</td>
<td>Indicate willingness to accept connections</td>
</tr>
<tr>
<td>4) accept()</td>
<td>Accept a connection request</td>
</tr>
<tr>
<td>5) Read request</td>
<td></td>
</tr>
<tr>
<td>6) Write response</td>
<td></td>
</tr>
<tr>
<td>7) Return to step 4</td>
<td></td>
</tr>
</tbody>
</table>

Once the accept() routine is called, the server will block until data is available for the socket. Problems arise when the server wishes to monitor multiple sockets simultaneously or if the server wishes to perform other tasks until data is available on the socket. However, by configuring the socket as non-blocking, these problems may be avoided. For more information, see "Enable/Disable Non-Blocking Mode" on page 26.

When using non-blocking sockets, the select() routine is used to instruct the system to notify the server application when data is available on a particular socket.

Table 6. Using the select() Routine

<table>
<thead>
<tr>
<th>Routine</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) socket()</td>
<td>Create a socket</td>
</tr>
<tr>
<td>2) bind()</td>
<td>Register the socket</td>
</tr>
<tr>
<td>3) listen()</td>
<td>Indicate willingness to accept connections</td>
</tr>
<tr>
<td>4) Set socket as non-blocking</td>
<td>See &quot;Enable/Disable Non-Blocking Mode&quot; on page 26</td>
</tr>
<tr>
<td>5) select()</td>
<td>Monitor a number of sockets</td>
</tr>
<tr>
<td>6) accept()</td>
<td>Accept a connection request</td>
</tr>
<tr>
<td>7) Read request</td>
<td>If unable to read all data, return to step 5</td>
</tr>
<tr>
<td>8) Write response</td>
<td>If unable to write all data, return to step 5</td>
</tr>
<tr>
<td>9) Return to step 4</td>
<td></td>
</tr>
</tbody>
</table>
Affected SSL Functions
These functions are affected by the use of non-blocking sockets with SSL.

gsk_secure_socket_init()
During the SSL handshake, the io_setsocketoptions() routine is called by the
gsk_secure_socket_init() routine before initiating the SSL handshake
(GSK_SET_SOCKET_STATE_FOR_HANDSHAKE) and again upon completion of the SSL
handshake (GSK_SET_SOCKET_STATE_FOR_READ_WRITE). The default
io_setsocketoptions() routine puts the socket into blocking mode for
GSK_SET_SOCKET_STATE_FOR_HANDSHAKE and restores the original mode for
GSK_SET_SOCKET_STATE_FOR_READ_WRITE. In order to perform a non-blocking SSL
handshake, an application supplied io_setsocketoptions() callback must be provided to control
the state of the socket. When the socket is in non-blocking mode, gsk_secure_socket_init() may
return GSK_WOULD_BLOCK_READ or GSK_WOULD_BLOCK_WRITE. This error indicates that
System SSL was unable to read or write the entire message. When this occurs, the application
should call select() and then call gsk_secure_socket_init() again.

gsk_secure_socket_read()
Once the socket has been configured as non-blocking, any calls to gsk_secure_socket_read()
can potentially return GSK_WOULD_BLOCK. When this occurs, the application should call
select() and then call gsk_secure_socket_read() again.

gsk_secure_socket_write()
Once the socket has been configured as non-blocking, any calls to gsk_secure_socket_write()
can potentially return GSK_WOULD_BLOCK. When this occurs, the application should call
select() and then call gsk_secure_socket_write() again.

Enable/Disable Non-Blocking Mode:  Once a socket has been created using the socket() call, it may be
set to non-blocking as follows:
#include "sys/ioctl.h"
int on =1;
int off =0;

//Enable non-blocking
ioctl (mySocket, FIONBIO, &(on));
//Disable non-blocking
ioctl (mySocket, FIONBIO, (char *) &(off));

Differences in SSL and Unsecured Non-Blocking Mode:

Partial Data
An unsecured socket in non-blocking mode will return the partial data received or written. Since
System SSL processes encrypted data, it is not possible to decrypt a message until the entire
message has been received, making it impossible to return partial data.

Error Indicator
When non-blocking mode is used on a non-secure socket, the status indicator is generally found
by checking the errno variable, which is normally EWOULDBLOCK. System SSL does not set the
errno variable. Instead the value returned from gsk_secure_socket_read() or
gsk_secure_socket_write() is set to GSK_WOULD_BLOCK. gsk_secure_socket_init() will
return either GSK_WOULD_BLOCK_READ or GSK_WOULD_BLOCK_WRITE.

Client Authentication Certificate Selection
SSL enables the application to prompt the client user to select a certificate from a list during the client
authentication process in the SSL handshake.

This is accomplished with a registered callback routine that is invoked from inside the
gsk_secure_socket_init() function call. This topic provides an overview of that code.
The client application code must provide these functions:

- Register a standard C linkage callback routine using the `gsk_attribute_set_callback()` function call.
- Implement the callback routine that performs these functions:
  - Get the list of available certificates using the `gsk_attribute_get_data()` function call with the GSK_DATA_ID_SUPPORTED_KEYS option. This returns a list of labels from the key data base file, SAF key ring or z/OS PKCS #11 token.
  - Display the list of labels to the user.
  - Prompt the user to select the label from the list.
  - Set the label to be used with a `gsk_attribute_set_buffer()` function call with the GSK_KEYRING_LABEL option.
  - Return to SSL with the return value set to indicate use client authentication.
  - If the user elects to not use any of the certificates in the list, return with the value set to skip client authentication. A certificate will not be sent to the partner, but the SSL handshake will complete. The server will decide whether to continue or close the connection.
  - Optionally, the application can display certificate information using the `gsk_get_cert_by_label()` function call.
  - Optionally, the application can use the `gsk_attribute_get_data()` function call with the GSK_DATA_ID_SERVER_ISSUERS option to display a list of server signer certificates.

### I/O Routine Replacement

#### Callback Routine for IO

SSL allows applications to specify how I/O is to take place. This is done by specifying callback routines for receiving and sending data. The contents of this routine can be very unique per application. SSL has an internally defined default routine which is used if `gsk_attribute_set_callback()` is not used to override I/O routines. The default assumes that TCP/IP is being used. For reading it will execute a `recv()` and for write a `send()`. If not using TCP/IP, applications should also consider the specification of the `getpeername` and `setsockopt` callback routine. It also depends on TCP/IP as being the transport layer protocol.

**Note:** Application provided I/O routines must use standard C linkage conventions.

#### Use of User Data

Some complex applications require application-specific data to be available in the SSL callbacks. SSL enables this with the `gsk_attribute_set_buffer()` and `gsk_attribute_get_buffer()` function calls. In addition, the I/O callbacks pass a pointer to the user data.

These are the steps that need to be taken to effectively use the user data functions:

- Issue the `gsk_secure_socket_open()` function. This will return a soc_handle.
- To set the user data for a connection issue:
  - `gsk_attribute_set_buffer(soc_handle, GSK_USER_DATA, user_data, sizeof(user_data));`
  - This function call copies the `user_data` into an area of storage owned by SSL.
- The address of the SSL copy of the user data is passed as a parameter to the user-specified read, write, getpeername, and set_socket_options callbacks.
- Other callbacks pass the `soc_handle` as a parameter to the callback. To find the address of the copy of user data associated with a particular connection, issue:
  - `gsk_attribute_get_buffer(soc_handle, GSK_USER_DATA, &user_data_ptr,&user_data_size);`
  - You can modify the contents of the SSL copy of the user data, but you may not free or re-allocate the SSL user data. The SSL user data will be freed when the connection is closed with the `gsk_secure_socket_close()` function call.
You can point to other application data from the SSL user data area. However, it is up to the application to free this other application data before the connection is closed.

**Session ID (SID) Cache Replacement**

The SSL protocol has a mechanism built in to allow for faster secure connections between a client/server pair. There is a concept of a SSL Session that allows this to happen. The first time a client and server connect, cryptographic characteristics of that connection are saved into a Session Cache buffer. A Session is identified by a Session ID (SID). The cached cryptographic components (SID cache entry) allows for new bulk encryption keys to be generated with subsequent SSL handshakes between the same client/server pair. The subsequent handshakes would be abbreviated since much of the data used to generate keys is in the SID cache entry. This abbreviated handshake does not require public key encryption to take place.

Public key encryption is very time consuming thus avoiding it is a great performance boost for servers using SSL. A SID Cache entry exists for a limited time. Care should be used when specifying how long a SSL session is allowed to live. Setting the SID cache timeout or number of SID cache buffer entries to ZERO will turn off SID caching causing a full handshake to be completed for every connection.

Applications need to be sensitive to both security and performance issues. Security conscious applications should keep the session timeout values very low to ensure keys are generated frequently to avoid security breaches. Applications that are more performance conscious than security conscious should have longer session timeouts and a larger cache size.

**Session ID (SID)**

Modifying SSL session caching parameters can be used to help tune the security performance characteristics of SSL enabled servers. SSL internally does session caching and is controlled only by setting the length of a SSL session and the number of entries in the buffer. The internal SSL SID cache is fixed to a configurable number of entries. There is no way to remove or to re-use entries for other connections except for repeated connections between the same client/server pair. The list of options for extending SID caching functionality can become quite long so an external SID cache buffer API was created for those who are more discriminating about managing SID cache data. There are several callbacks used for external SID cache buffer access.

It should be noted that there are probably few applications where using an external SID cache makes sense. Some recommended environments where it might be considered is in a server configuration where multiple instances of a server exist for work load balancing purposes. It might be desirable to have a single SID cache buffer to be used by all of the processes which each server is running in. Usually this can be avoided by writing applications which are multi threaded. All threads would use the single internal SID cache buffer.

Format:

```c
typedef gsk_data_buffer * (*ptgsk_getcache) (const unsigned char * session_id, unsigned int session_id_length, int ssl_version);

typedef gsk_data_buffer * (*ptgsk_putcache) (gsk_data_buffer * ssl_session_data, const unsigned char * session_id, unsigned int session_id_length, int ssl_version);

typedef void (*ptgsk_deletecache) (const unsigned char * session_id, unsigned int session_id_length, int ssl_version);

typedef void (*ptgsk_freecache) (const unsigned char * session_id, unsigned int session_id_length, int ssl_version);
```
Writing and Building a z/OS System SSL Application

```c
typedef struct _gsk_sidcache_callback {
    ptgsk_getcache Get;
    ptgsk_putcache Put;
    ptgsk_deletecache Delete;
    ptgsk_freecache FreeDataBuffer;
} gsk_sidcache_callback;
```

**Callbacks:**

**Get**

Specifies the routine System SSL calls to search the session ID cache for the entry that matches the passed values in `sessionId`, `sessionIdLen`, and `SSLVersion`. The value returned by this routine is a pointer to a malloc'ed `gsk_data_buffer` structure for the `sslSessionData` that contains the session id cache entry.

**Put**

Specifies the routine System SSL calls to add an entry to the session ID cache. The passed in values `sessionId`, `sessionIdLen`, `SSLVersion` and `sslSessionData` are used to define the entry. This routine is responsible for getting storage to hold the entry. The value returned by this routine is either `NULL` if unable to allocate storage or a pointer to a `gsk_data_buffer` structure containing the `sslSessionData` that was passed into the routine.

**Delete**

Specifies the routine System SSL calls to delete an entry from the session ID cache. `sessionId`, `sessionIdLen` and `SSLVersion` are used to determine which entry is deleted.

**FreeDataBuffer**

Specifies the routine that System SSL calls to free memory that was returned by the `Get` session id cache callback routine.

**Parameters:**

`sessionId`

The buffer containing the Session data

`sessionIdLen`

The length of the entry for the SID cache buffer entry.

`SSLVersion`

The version of the SSL Protocol.

`data`

This is the buffer that is created by the external SID cache process to transfer the SID cache entry to SSL.

**Session Renegotiation Notification**

SSL provides a mechanism to renegotiate the communications session to establish a new session key or have the session cipher reset. This can be initiated by either the SSL server or SSL client through the `gsk_secure_socket_misc` API. System SSL allows applications to specify callback routines for receiving notifications when SSL is commencing and completing a session renegotiation. System SSL will call the specified routines and supply the connection handle for session identification, indicating that new session keys are being negotiated. This allows the user application to take specific actions during a session renegotiation, such as suspending application communications until the negotiation is complete.

**TLS Extensions**

System SSL allows applications to specify TLS extensions that add functionality to the Transport Layer Security (TLS) protocol. TLS extensions may be set by both TLS clients and servers. The use of TLS extensions is backwards compatible: communication is possible between TLS clients that support TLS extensions and TLS servers that do not support TLS extensions, and vice versa.
To use TLS extensions in a TLS client/server session, the `gsk_attribute_set_tls_extension()` SSL API must be used to define the extensions that the TLS client or server will support. TLS extensions may be defined:

- After `gsk_environment_open()` has been performed but prior to the `gsk_environment_init()` call
- After `gsk_secure_socket_open()` has been performed but prior to the `gsk_secure_socket_init()` call

TLS extensions that are defined for an SSL environment will apply to all connections within the environment. Each connection can define additional TLS extensions to be used for that connection, or may override TLS extension settings that have been defined for the environment. System SSL currently provides support for the following TLS extensions:

**Truncated HMAC**
Truncates the HMAC used to authenticate record layer communications to 80 bits

**Maximum Fragment Length**
Allows the client to use a fragment length smaller than the TLS default of 16,384 bytes when transmitting messages

**Server Name Indication**
Allows the client to tell the server the name of the server it wants to connect to

**Setting Server Side Extensions**
The following example illustrates how to define each of the supported System SSL TLS extensions for a TLS server. The extensions are defined at the environment level and are optional. Optional allows the TLS server to communicate with TLS clients that support the extensions as well as TLS clients that do not support the extensions.

```c
int rc;
gsk_handle envHandle;

gsk_tls_extension tls_extn[3];
char server1[] = "server1.ibm.com";
char server2[] = "server2.ibm.com";
char server3[] = "server3.ibm.com";
char label1[] = "Server1 Certificate";
char label2[] = "Server2 Certificate";
char label3[] = "Server3 Certificate";
gsk_server_key_label serverLabelPairs[] = {{server1, label1},
                                           {server2, label2},
                                           {server3, label3}};

/*
 * Open the SSL environment
 */
rc = gsk_environment_open(&envHandle);

/*
 * Set truncated HMAC extension
 */
memset(&tls_extn[0], 0, sizeof(gsk_tls_extension));
tls_extn[0].extId = GSK_TLS_EXTID_TRUNCATED_HMAC;
tls_extn[0].required = FALSE; /* optional extension */
tls_extn[0].u.truncateHmac = TRUE; /* enable extension */
rc = gsk_attribute_set_tls_extension(envHandle, &tls_extn[0]);
*/
```
Writing and Building a z/OS System SSL Application

```c
/* Set maximum fragment length extension */
memset(&tls_extn[1], 0, sizeof(gsk_tls_extension));
tls_extn[1].extId = GSK_TLS_EXTID_SERVER_MFL;
tls_extn[1].required = FALSE; /* optional extension */
tls_extn[1].u.maxFragmentLength = GSK_TLS_MFL_ON;
/* enable extension */
rc = gsk_attribute_set_tls_extension(envHandle, &tls_extn[1]);

/* Set server name indication extension */
memset(&tls_extn[2], 0, sizeof(gsk_tls_extension));
tls_extn[2].extId = GSK_TLS_EXTID_SNI_SERVER_LABELS;
tls_extn[2].required = FALSE; /* optional extension */
tls_extn[2].u.serverLabels.setSni = TRUE;
/* enable extension */
tls_extn[2].u.serverLabels.unrecognized_name_fatal = TRUE;
/* unrecognized name is fatal */
tls_extn[2].u.serverLabels.count = 3;
tls_extn[2].u.serverLabels.serverKeyLabel = serverLabelPairs;
rc = gsk_attribute_set_tls_extension(envHandle, &tls_extn[2]);

/* Initialize the SSL environment */
r = gsk_environment_init(envHandle);
```

**Setting Client Side Extensions**

The following example illustrates how to define each of the supported System SSL TLS extensions for a TLS client. The HMAC and maximum fragment extensions are defined at the environment level. The server name indication extension is defined, while the HMAC extension is modified for a particular connection. The environment level extensions are being defined as required and connection level extensions as optional. Required extensions require that the partner TLS server support the specified TLS extensions. If it does not support the extensions, the TLS handshake will fail.

```c
int gsk_handle

gsk_handle

gsk_tls_extension tls_extn_env[2];
gsk_tls_extension tls_extn_con[2]
char server1[] = "server1.ibm.com";
char server2[] = "server2.ibm.com";
char * serverNames[] = {server1, server2};

/* Open the SSL environment */
r = gsk_environment_open(&envHandle);
/*
    * Set truncated HMAC extension
    */
memset(&tls_extn_env[0], 0, sizeof(gsk_tls_extension));
tls_extn_env[0].extId = GSK_TLS_EXTID_TRUNCATED_HMAC;
```
Writing and Building a z/OS System SSL Application

tls_extn_env[0].required = TRUE; /* required extension */
tls_extn_env[0].u.truncateHmac = TRUE; /* enable extension */
rc = gsk_attribute_set_tls_extension(envHandle,&tls_extn_env[0]);

/*
 * Set maximum fragment length extension
 */
memset(&tls_extn_env[1], 0, sizeof(gsk_tls_extension);
tls_extn_env[1].extId = GSK_TLS_EXTID_CLIENT_MFL;
tls_extn_env[1].required = TRUE; /* required extension */
tls_extn_env[1].u.maxFragmentLength = GSK_TLS_MFL_4096;
/* set 4096 bit fragment length */
rc = gsk_attribute_set_tls_extension(envHandle,&tls_extn_env[1]);

/*
 * Initialize the SSL environment
 */
rc = gsk_environment_init(envHandle);

/*
 * Open the SSL connection
 */
rc = gsk_secure_socket_open(envHandle, &conHandle);

/*
 * Set server name indication extension
 */
memset(&tls_extn_con[0], 0, sizeof(gsk_tls_extension);
tls_extn_con[0].extId = GSK_TLS_EXTID_SNI_CLIENT_SNAMES;
tls_extn_con[0].required = FALSE; /* optional extension */
tls_extn_con[0].u.clientSnameList.setSni = TRUE;
/* enable extension */
tls_extn_con[0].u.clientSnameList.unrecognized_name_fatal = TRUE;
/* unrecognized name is fatal */
tls_extn_con[0].u.clientSnameList.count = 2;
tls_extn_con[0].u.clientSnameList.serverNames = serverNames;
rc = gsk_attribute_set_tls_extension(envHandle,&tls_extn_con[0]);

/*
 * Modify truncated HMAC extension
 */
memset(&tls_extn_con[0], 0, sizeof(gsk_tls_extension);
tls_extn_con[0].extId = GSK_TLS_EXTID_TRUNCATED_HMAC;
tls_extn_con[1].required = FALSE; /* optional extension */
tls_extn_con[0].u.truncateHmac = TRUE; /* enable extension */
rc = gsk_attribute_set_tls_extension(envHandle,&tls_extn_con[1]);

/*
 * Initialize the SSL connection
 */
rc = gsk_secure_socket_init(conHandle)
Chapter 6. Migrating from Deprecated SSL Interfaces

In Release 2 of z/OS, a new set of functions were added that superseded some functions from previous System SSL releases. The functions that were superseded are referred to collectively as “the deprecated SSL interface”. IBM recommends that new application programs do not use the deprecated SSL interface. For application programs that currently make use of the deprecated SSL interface, this topic describes how to migrate to the most recent interface.

**Note:** When migrating from the deprecated SSL interface, the entire System SSL application must be migrated. The application must not contain a mixture of deprecated and superseding APIs.

- Replace manually initializing the `gsk_init_data` structure with `gsk_environment_open()`, plus a number of `gsk_attribute_set_buffer()`, `gsk_attribute_set_enum()` and `gsk_attribute_set_numeric_value()` functions (as needed) to set attributes.

- Replace `gsk_get_cipher_info()` with a call to `gsk_attribute_get_buffer()` to get the list of available ciphers. This call must be done after a successful `gsk_environment_open()` call. The ciphers returned always represent the high security ciphers.

- Replace `gsk_initialize()` with `gsk_environment_init()`.

- Replace manually initializing the `gsk_soc_init_data` structure with `gsk_secure_socket_open()`, plus a number of `gsk_attribute_set_buffer()`, `gsk_attribute_set_enum()` and `gsk_attribute_set_numeric_value()` functions (as needed) to set attributes.

- Replace manually initializing the `gsk_soc_init_data` structure with the addresses of your I/O callback routines with `gsk_attribute_set_callback()`.
  You specify the address of a `gsk_iocallback` structure that contains the addresses of the callback routines. The `gsk_iocallback` structure is defined in `gskssl.h`. Note that an additional parameter must be added to the function declarator for your existing callback routines.

- Replace `gsk_user_set()` with `gsk_attribute_set_callback()` for defining the address of your get peer ID callback routine. You specify the address of an `gsk_iocallback` structure that contains the address of the callback routine. The `gsk_iocallback` structure is defined in `gskssl.h`. Note that an additional parameter must be added to the function declarator for your existing callback routine.

- Replace `gsk_user_set()` with `gsk_attribute_set_callback()` for defining the address of your session ID cache callback routines. You specify the address of a `gsk_sidcache_callback` structure that contains the address of the callback routines. The `gsk_sidcache_callback` structure is defined in `gskssl.h`.

- Replace `gsk_get_dn_by_label()` with `gsk_get_cert_by_label()`.

- Replace `gsk_secure_soc_init()` with `gsk_secure_socket_init()`.

- Replace `gsk_secure_soc_read()` with `gsk_secure_socket_read()`. Note that `gsk_secure_socket_read()` has an extra parameter to return the length of the data read.

- Replace `gsk_secure_soc_write()` with `gsk_secure_socket_write()`. Note that `gsk_secure_socket_write()` has an extra parameter to return the length of the data written.

- To notify your partner application that you are done sending data on the secure connection, a call to `gsk_secure_socket_shutdown` should be issued before the `gsk_secure_socket_close` call.

- Replace `gsk_secure_soc_close()` with `gsk_secure_socket_close()`.

- Be sure that every `gsk_secure_socket_open()` is matched with a `gsk_secure_socket_close()` even if there is an error on `gsk_environment_init()`. Normal sequence is open, init, close. So, if `init` gets an error return code, you still must do the `close`.

- Be sure that every `gsk_environment_open()` is matched with a `gsk_environment_close()` even if there is an error on `gsk_secure_socket_init()`. Normal sequence is open, init, close. So, if `init` gets an error return code, you still must do the `close`.

- A method is provided to display certificates after `gsk_secure_socket_init()` has been issued. You may use `gsk_attribute_get_cert_info()` if desired.
• Be aware that all of the error return values have been renamed and renumbered. Program logic will have to be changed accordingly.

• There is a `gsk_strerror()` debug routine that returns a text string (in English only) when an error number is passed to it.
Chapter 7. API Reference

This topic describes the set of application programming interfaces (APIs) that z/OS System SSL supports for performing secure sockets layer (SSL/TLS) communication.

These APIs were introduced in z/OS Version 1 Release 2 and beyond and supersede the APIs from prior releases. Only the APIs in this topic should be used for writing new application programs. Existing application programs should be recoded if possible to use the new APIs. See Chapter 6, “Migrating from Deprecated SSL Interfaces,” on page 33 for more information about updating your application programs.

The deprecated APIs included in Chapter 9, “Deprecated Secure Sockets Layer APIs,” on page 345 are for reference only. When creating new application programs, you must not include any of the deprecated APIs; you should use only the APIs in this topic.

These provide more information on X.509 certificates and the Secure Sockets Layer protocol:
- ANSI X9.31 - RSA Key Generation
- FIPS 186 - Digital Signature Standard (DSS)
- PKCS #1 - RSA Encryption Standard
- PKCS #3 - Diffie-Hellman Key Agreement Standard
- PKCS #5 - Password-based Encryption
- PKCS #7 - Cryptographic Message Syntax
- PKCS #8 - Private Key Information Syntax
- PKCS #10 - Certification Request
- PKCS #12 - Personal Information Exchange
- RFC 2246 - Transport Layer Security (TLS) Version 1
- RFC 2253 - String Representation of Distinguished Names
- RFC 2279 - UTF-8 (UCS Transformation Format 8)
- RFC 2459 - X.509 certificate, certificate revocation list, and certificate extensions
- RFC 2587 - PKIX LDAP Version 2 Schema
- RFC 2631 - Diffie-Hellman Key Agreement Method
- RFC 3268 - Advanced Encryption Standard (AES) for Transport Layer Security (TLS)
- RFC 3280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
- RFC 4346 - Transport Layer Security (TLS) Version 1.1
- RFC 4366 - Transport Layer Security (TLS) Extensions

This is a list of APIs. Use these APIs when creating new application programs. If possible, recode your existing application programs to use these APIs as well:
- gsk_attribute_get_buffer() (see page 37)
- gsk_attribute_get_cert_info() (see page 40)
- gsk_attribute_get_data() (see page 45)
- gsk_attribute_get_enum() (see page 47)
- gsk_attribute_get_numeric_value() (see page 50)
- gsk_attribute_set_buffer() (see page 52)
- gsk_attribute_set_callback() (see page 55)
- gsk_attribute_set_enum() (see page 59)
- gsk_attribute_set_numeric_value() (see page 62)
• gsk_attribute_set_tls_extensions() (see page 64)
• gsk_environment_close() (see page 67)
• gsk_environment_init() (see page 68)
• gsk_environment_open() (see page 70)
• gsk_free_cert_data() (see page 75)
• gsk_get_cert_by_label() (see page 76)
• gsk_get_cipher_suites() (see page 81)
• gsk_get_ssl_vector() (see page 82)
• gsk_get_update() (see page 83)
• gsk_list_free() (see page 84)
• gsk_secure_socket_close() (see page 85)
• gsk_secure_socket_init() (see page 86)
• gsk_secure_socket_misc() (see page 90)
• gsk_secure_socket_open() (see page 92)
• gsk_secure_socket_read() (see page 93)
• gsk_secure_socket_shutdown() (see page 95)
• gsk_secure_socket_write() (see page 97)
• gsk_strerror() (see page 99)
gsk_attribute_get_buffer() 

Gets the value of an attribute buffer.

Format

```c
#include <gskssl.h>
gsk_status gsk_attribute_get_buffer ( 
  gsk_handle ssl_handle, 
  GSK_BUF_ID buffer_id, 
  const char ** buffer_value, 
  int * buffer_length)
```

Parameters

- **ssl_handle**
  Specifies an SSL environment handle returned by `gsk_environment_open()` or an SSL connection handle returned by `gsk_secure_socket_open()`.

- **buffer_id**
  Specifies the buffer identifier.

- **buffer_value**
  Returns the address of the buffer value. The buffer is in storage owned by the SSL runtime and must not be modified or released by the application. The buffer returned for the GSK_USER_DATA identifier may be modified by the application but must not be released.

- **buffer_length**
  Returns the length of the buffer value.

Results

The function return value will be 0 (GSK_OK) if no error is detected. Otherwise, it will be one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

- **[GSK_ATTRIBUTE_INVALID_ID]**
  The buffer identifier is not valid or cannot be used with the specified handle.

- **[GSK_INVALID_HANDLE]**
  The handle is not valid.

- **[GSK_INVALID_STATE]**
  The handle is closed.

Usage

The `gsk_attribute_get_buffer()` routine will return a buffer value for an SSL environment or an SSL connection. The buffer is in storage owned by the SSL runtime and must not be released by the application. The address will remain valid until the SSL environment or connection is closed or until the application calls the `gsk_attribute_set_buffer()` routine to set a new buffer value.

These buffer identifiers are supported:

- **GSK_CONNECT_CIPHER_SPEC**
  Returns the cipher specification selected for an initialized connection. Refer to the description of the `gsk_environment_open()` routine for a list of valid cipher specifications. GSK_CONNECT_CIPHER_SPEC may be specified only for an SSL connection.

- **GSK_CONNECT_SEC_TYPE**
  Returns the security protocol for an initialized connection. The value will be "SSLV2", "SSLV3", .
"TLSV1", or "TLSV1.1" depending upon the protocol selected during the SSL handshake. GSK_CONNECT_SEC_TYPE may be specified only for an SSL connection.

GSK_KEYRING_FILE
Returns the name of the key database HFS file, SAF key ring or z/OS PKCS #11 token. A key database is used if a database password or stash file is defined using either an environment variable or the gsk_attribute_set_buffer() routine.

GSK_KEYRING_LABEL
Returns the label associated with the certificate being used by the SSL environment or connection. This will be the value set by the application if the environment or connection is not initialized. GSK_KEYRING_LABEL may be specified for an SSL environment or an SSL connection.

GSK_KEYRING_PW
Returns the password for the key database. A NULL address will be returned after the environment is initialized. GSK_KEYRING_PW may be specified only for an SSL environment.

GSK_KEYRING_STASH_FILE
Returns the name of the key database password stash file. GSK_KEYRING_STASH_FILE may be specified only for an SSL environment.

GSK_LDAP_SERVER
Returns the DNS name or IP address of the LDAP server. GSK_LDAP_SERVER may be specified only for an SSL environment.

GSK_LDAP_USER
Returns the distinguished name to use when connecting to the LDAP server. GSK_LDAP_USER may be specified only for an SSL environment.

GSK_LDAP_USER_PW
Returns the password to use when connecting to the LDAP server. GSK_LDAP_USER_PW may be specified only for an SSL environment.

GSK_SID_VALUE
Returns the session identifier for an initialized connection. This is the Base64-encoded version of the session identifier and consists of displayable characters. GSK_SID_VALUE may be specified only for an SSL connection.

GSK_SNI_LIST
Returns the address of a list of server names passed to the server by the client for use during server name indication callback routine. Server name indication is an extension to TLS V1.0 and TLS V1.1 protocols which allows the client to pass server names to the server. The server can use the list of server names as an aid in selection of the certificate to be used by the server. GSK_SNI_LIST may be specified only for an SSL connection and only on the server side of the connection. When returned, the buffer will contain a list of server names with each server name preceded by a 1-byte name type and a 2-byte field (in big endian format) containing the length of the server name. The name type will always contain X'00' to indicate it is a hostname; however, new name types may be introduced in the future. The server name content will be in UTF-8 format.

GSK_USER_DATA
Returns the address of the user data to be passed to SSL exit routines. The application may alter the user data but may not free it. GSK_USER_DATA may be specified only for an SSL connection.

GSK_V2_CIPHER_SPECS
Returns the SSL V2 cipher specifications as a string consisting of 1-character values. GSK_V2_CIPHER_SPECS may be specified for an SSL environment or an SSL connection. Refer to the description of gsk_environment_open() for a list of valid cipher specifications.

GSK_V3_CIPHER_SPECS
Returns the SSL V3 cipher specifications as a string consisting of 2-character values. GSK_V3_CIPHER_SPECS may be specified for an SSL environment or an SSL connection.
SSL V3 cipher specifications are used for the SSL V3, TLS V1.0 and TLS V1.1 protocols. Refer to the description of `gsk_environment_open()` for a list of valid cipher specifications.

**Related Topics**

- `gsk_attribute_set_buffer()`
- `gsk_environment_open()`
- `gsk_secure_socket_open()`
gsk_attribute_get_cert_info()

Returns certificate information following an SSL handshake.

Format

```c
#include <gskssl.h>

gsk_status gsk_attribute_get_cert_info (  
    gsk_handle soc_handle,  
    GSK_CERT_ID cert_id,  
    gsk_cert_data_elem **cert_data,  
    int *elem_count)
```

Parameters

- **soc_handle**
  Specifies the connection handle returned by the `gsk_secure_socket_open()` routine.

- **cert_id**
  Specifies the certificate identifier.

- **cert_data**
  Returns the certificate data array. The `gsk_free_cert_data()` routine should be called to release the array when the certificate information is no longer needed. A NULL address will be returned if no certificate information is available.

- **elem_count**
  Returns the number of elements in the array of gsk_cert_data_elem structures.

Results

The function return value will be 0 (GSK_OK) if no error is detected. Otherwise, it will be one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

- **[GSK_ATTRIBUTE_INVALID_ID]**
  The certificate identifier is not valid.

- **[GSK_ERR_ASN]**
  Unable to decode certificate.

- **[GSK_INSUFFICIENT_STORAGE]**
  Insufficient storage is available.

- **[GSK_INVALID_HANDLE]**
  The connection handle is not valid.

- **[GSK_INVALID_STATE]**
  The connection is not initialized.

Usage

The `gsk_attribute_get_cert_info()` routine returns information about certificates used in an SSL handshake. The connection must be in the initialized state. The certificate data address will be NULL if there is no certificate information available.

These certificate identifiers are supported:

- **GSK_LOCAL_CERT_INFO**
  Returns information about the local certificate.

- **GSK_PARTNER_CERT_INFO**
  Returns information about the partner certificate.
Each element of the certificate data array has an element identifier. The element identifiers used for a particular certificate will depend upon the contents of the certificate. These element identifiers are currently provided:

**CERT_BODY_BASE64**
Certificate body in Base64-encoded format

**CERT_BODY_DER**
Certificate body in binary ASN.1 DER-encoded format

**CERT_COMMON_NAME**
Subject common name (CN)

**CERT_COUNTRY**
Subject country (C)

**CERT_DN_DER**
Subject distinguished name in binary ASN.1 DER-encoded format

**CERT_DN_PRINTABLE**
Subject distinguished name as a printable character string

These DN attribute names are recognized by the System SSL runtime.

- C - Country
- CN - Common name
- DC - Domain component
- DNQUALIFIER - Distinguished name qualifier
- EMAIL - E-mail address
- GENERATIONQUALIFIER - Generation qualifier
- GIVENNAME - Given name
- INITIALS - Initials
- L - Locality
- MAIL - RFC822 style address
- NAME - Name
- O - Organization name
- OU - Organizational unit name
- PC - Postal code
- SERIALNUMBER - Serial number
- SN - Surname
- ST - State or province
- STREET - Street
- T - Title

**CERT_DNQUALIFIER**
Subject distinguished name qualifier (DNQUALIFIER)

**CERT_DOMAIN_COMPONENT**
Subject domain component (DC)

**CERT_EMAIL**
Subject e-mail address (EMAIL)

**CERT_GENERATIONQUALIFIER**
Subject generation qualifier (GENERATIONQUALIFIER)

**CERT_GIVENNAME**
Subject given name (GIVENNAME)
gsk_attribute_get_cert_info()

- CERT_INITIALS
  - Subject initials (INITIALS)

- CERT_ISSUER_COMMON_NAME
  - Issuer common name (CN)

- CERT_ISSUER_COUNTRY
  - Issuer country (C)

- CERT_ISSUER_DN_DER
  - Issuer distinguished name in binary ASN.1 DER-encoded format

- CERT_ISSUER_DN_PRINTABLE
  - Issuer distinguished name as a printable character string
    - These DN attribute names are recognized by the System SSL runtime.
      - C - Country
      - CN - Common name
      - DC - Domain component
    - DNQUALIFIER - Distinguished name qualifier
      - EMAIL - E-mail address
    - GENERATIONQUALIFIER - Generation qualifier
    - GIVENNAME - Given name
    - INITIALS - Initials
    - L - Locality
    - MAIL - RFC822 style address
    - NAME - Name
      - O - Organization name
      - OU - Organizational unit name
    - PC - Postal code
    - SERIALNUMBER - Serial number
      - SN - Surname
      - ST - State or province
      - STREET - Street
      - T - Title

- CERT_ISSUER_DNQUALIFIER
  - Issuer distinguished name qualifier (DNQUALIFIER)

- CERT_ISSUER_DOMAIN_COMPONENT
  - Issuer domain component (DC)

- CERT_ISSUER_EMAIL
  - Issuer e-mail address (EMAIL)

- CERT_ISSUER_GENERATIONQUALIFIER
  - Issuer generation qualifier (GENERATIONQUALIFIER)

- CERT_ISSUER_GIVENNAME
  - Issuer given name (GIVENNAME)

- CERT_ISSUER_INITIALS
  - Issuer initials (INITIALS)

- CERT_ISSUER_LOCALITY
  - Issuer locality (L)
gsk_attribute_get_cert_info()

| CERT_ISSUER_MAIL       | Issuer RFC822 style address (MAIL)       |
| CERT_ISSUER_NAME       | Issuer name (NAME)                       |
| CERT_ISSUER_ORG        | Issuer organization (O)                  |
| CERT_ISSUER_ORG_UNIT   | Issuer organizational unit (OU)          |
| CERT_ISSUER_POSTAL_CODE | Issuer postal code (PC)              |
| CERT_ISSUER_SERIALNUMBER | Issuer serial number (SERIALNUMBER) |
| CERT_ISSUER_STATE_OR_PROVINCE | Issuer state or province (ST)  |
| CERT_ISSUER_STREET     | Issuer street (STREET)                   |
| CERT_ISSUER_SURNAME    | Issuer surname (SN)                      |
| CERT_ISSUER_TITLE      | Issuer title (T)                         |
| CERT_LOCALITY          | Subject locality (L)                      |
| CERT_MAIL              | Subject RFC822 style address (MAIL)      |
| CERT_NAME              | Subject name (NAME)                      |
| CERT_ORG               | Subject organization (O)                 |
| CERT_ORG_UNIT          | Subject organizational unit (OU)         |
| CERT_POSTAL_CODE       | Subject postal code (PC)                  |
| CERT_SERIAL_NUMBER     | Certificate serial number                |
| CERT_SERIALNUMBER      | Subject serial number (SERIALNUMBER)     |
| CERT_STATE_OR_PROVINCE | Subject state or province (ST)           |
| CERT_STREET            | Subject street (STREET)                   |
| CERT_SURNAME           | Subject surname (SN)                      |
| CERT_TITLE             | Subject title (T)                         |
gsk_attribute_get_cert_info()

The CERT_BODY_DER, CERT_DN_DER, and CERT_ISSUER_DN_DER elements are not null-terminated and the 'cert_data' field must be used to get the element length. All of the other elements are null-terminated character strings and the 'cert_data' field is the length of the string excluding the end-of-string delimiter.

Related Topics

- gsk_secure_socket_init()
- gsk_free_cert_data()
gsk_attribute_get_data()

Returns information related to a certificate request.

Format

```c
#include <gskssl.h>
gsk_status gsk_attribute_get_data (  
gsk_handle  soc_handle,  
GSK_DATA_ID  data_id,  
void **  data_ptr)
```

Parameters

- `soc_handle` Specifies the connection handle returned by the `gsk_secure_socket_open()` routine.
- `data_id` Specifies the data identifier.
- `data_ptr` Returns the address of the requested data. The address will be NULL if the requested data is not available.

Results

The function return value will be 0 (GSK_OK) if no error is detected. Otherwise, it will be one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

- **[GSK_ATTRIBUTE_INVALID_ID]**
  The data identifier is not valid.
- **[GSK_ERR_ASN]**
  Unable to decode certification authority name.
- **[GSK_INSUFFICIENT_STORAGE]**
  Insufficient storage is available.
- **[GSK_INVALID_HANDLE]**
  The connection handle is not valid.
- **[GSK_INVALID_STATE]**
  The connection is not initialized.

Usage

The `gsk_attribute_get_data()` routine returns information related to a certificate request. The server sends a certificate request to the client as part of the client authentication portion of the SSL handshake. The connection must be in the initialized state.

These data identifiers are supported:

**GSK_DATA_ID_SUPPORTED_KEYS**
Returns a list of labels in the key database for certificates signed by a certification authority that is in the list provided by the server. A database entry will be included in the list only if it has both a certificate and a private key. If executing in FIPS mode, the list will only include labels that can be used in FIPS mode. The `gsk_list_free()` routine should be called to release the list when it is no longer needed.
**Related Topics**
gsk_list_free()
**gsk_attribute_get_enum()**

Gets an enumerated value.

**Format**

```c
#include <gskssl.h>

gboolean gsk_attribute_get_enum (const GskHandle ssl_handle, GskEnumId id, GskEnumValue *value);
```

**Parameters**

- `ssl_handle`: Specifies an SSL environment handle returned by `gsk_environment_open()` or an SSL connection handle returned by `gsk_secure_socket_open()`.
- `id`: Specifies the enumeration identifier.
- `value`: Returns the enumeration value.

**Results**

The function return value will be 0 (`GSK_OK`) if no error is detected. Otherwise, it will be one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

- **[GSK_ATTRIBUTE_INVALID_ID]**
  The enumeration identifier is not valid or cannot be used with the specified handle.

- **[GSK_INVALID_HANDLE]**
  The handle is not valid.

- **[GSK_INVALID_STATE]**
  The environment has been closed or the SSL connection has been established.

**Usage**

The `gsk_attribute_get_enum()` routine will return an enumerated value for an SSL environment or an SSL connection.

These enumeration identifiers are supported:

- **GSK_CERT_VALIDATION_MODE**
  Returns the certificate validation mode setting. Returns `GSK_CERT_VALIDATION_MODE_2459` if certificate validation will be based on the RFC2459 method and `GSK_CERT_VALIDATION_MODE_3280` if certificate validation will be based on the RFC3280 method. Returns `GSK_CERT_VALIDATION_MODE_ANY` if certificate validation can use any supported X.509 certificate validation method. `GSK_CERT_VALIDATION_MODE` can only be specified for an SSL environment.

- **GSK_CLIENT_AUTH_ALERT**
  Returns `GSK_CLIENT_AUTH_NOCERT_ALERT_OFF` if the SSL server application has been configured to allow client connections where client authentication has been requested and the client has failed to supply an X.509 certificate. Returns `GSK_CLIENT_AUTH_NOCERT_ALERT_ON` if the SSL server application has been configured to terminate client connections where client authentication has been requested and the client has failed to supply an X.509 certificate. `GSK_CLIENT_AUTH_ALERT` can be specified only for an SSL environment.
gsk_attribute_get_enum()

GSK_CLIENT_AUTH_TYPE
Returns GSK_CLIENT_AUTH_FULL_TYPE if received certificates are validated by the System SSL runtime and GSK_CLIENT_AUTH_PASSTHRU_TYPE otherwise. GSK_CLIENT_AUTH_TYPE can be specified only for an SSL environment.

GSK_CRL_SECURITY_LEVEL
Returns the level of security set for the SSL environment when contact is attempted between the application and an LDAP server that may contain a Certificate Revocation List (CRL). GSK_CRL_SECURITY_LEVEL can be specified only for an SSL environment.

One of the three possible settings for GSK_CRL_SECURITY_LEVEL will be returned:
- GSK_CRL_SECURITY_LEVEL_LOW - Certificate validation will not fail if the LDAP server cannot be contacted.
- GSK_CRL_SECURITY_LEVEL_MEDIUM - Certificate validation requires the LDAP server to be contactable, but does not require a CRL to be defined. This is the default setting.
- GSK_CRL_SECURITY_LEVEL_HIGH - Certificate validation requires the LDAP server to be contactable, and a CRL to be defined.

GSK_PROTOCOL_SSLV2
Returns GSK_PROTOCOL_SSLV2_ON if the SSL Version 2 protocol is enabled and GSK_PROTOCOL_SSLV2_OFF if the SSL Version 2 protocol is not enabled. GSK_PROTOCOL_SSLV2 can be specified for an SSL environment or an SSL connection.

GSK_PROTOCOL_SSLV3
Returns GSK_PROTOCOL_SSLV3_ON if the SSL Version 3 protocol is enabled and GSK_PROTOCOL_SSLV3_OFF if the SSL Version 3 protocol is not enabled. GSK_PROTOCOL_SSLV3 can be specified for an SSL environment or an SSL connection.

GSK_PROTOCOL_TLSV1
Returns GSK_PROTOCOL_TLSV1_ON if the TLS Version 1 protocol is enabled and GSK_PROTOCOL_TLSV1_OFF if the TLS Version 1 protocol is not enabled. GSK_PROTOCOL_TLSV1 can be specified for an SSL environment or an SSL connection.

GSK_PROTOCOL_TLSV1_1
Returns GSK_PROTOCOL_TLSV1_1_ON if the TLS Version 1.1 protocol is enabled and GSK_PROTOCOL_TLSV1_1_OFF if the TLS Version 1.1 protocol is not enabled. GSK_PROTOCOL_TLSV1_1 can be specified for an SSL environment or an SSL connection.

GSK_PROTOCOL_USED
Returns GSK_PROTOCOL_USED_SSLV2 if the SSL Version 2 protocol was used to establish the connection, GSK_PROTOCOL_USED_SSLV3 if the SSL Version 3 protocol was used to establish the connection, GSK_PROTOCOL_USED_TLSV1 if the TLS Version 1.0 protocol was used to establish the connection, or GSK_PROTOCOL_USED_TLSV1_1 if the TLS Version 1.1 protocol was used to establish the connection. GSK_NULL will be returned if a connection has not been established. GSK_PROTOCOL_USED can be specified only for an SSL connection.

GSK_SESSION_TYPE
Returns GSK_CLIENT_SESSION if the SSL handshake is to be performed as a client, GSK_SERVER_SESSION if the SSL handshake is to be performed as a server, or GSK_SERVER_SESSION_WITH_CL_AUTH if the SSL handshake is to be performed as a server requiring client authentication. GSK_SESSION_TYPE can be specified for an SSL environment or an SSL connection.

GSK_SID_FIRST
Returns GSK_SID_IS_FIRST if a full SSL handshake was performed to establish the connection or GSK_SID_NOT_FIRST if an existing session was used to establish the connection. GSK_NULL will be returned if a connection has not been established. GSK_SID_FIRST can be specified only for an SSL connection.
gsk_attribute_get_enum()

GSK_SYSPLEX_SIDCACHE
Returns GSK_SYSPLEX_SIDCACHE_ON if sysplex session caching is enabled for this application
or GSK_SYSPLEX_SIDCACHE_OFF if sysplex session caching is not enabled.
GSK_SYSPLEX_SIDCACHE can be specified only for an SSL environment.

GSK_TLSEXT_MFL
Returns GSK_TLSEXT_MFL_OFF if the "Maximum Fragment Length" type TLS extension has not
been negotiated, and the SSL connection is therefore using the default fragment length (16384
bytes). Returns GSK_TLSEXT_MFL_512, GSK_TLSEXT_MFL_1024, GSK_TLSEXT_MFL_2048 or
GSK_TLSEXT_MFL_4096 if the "Maximum Fragment Length" type TLS extension has been
negotiated, where the returned value will reflect the negotiated maximum fragment length.
GSK_TLSEXT_MFL can be specified only for an SSL connection.

GSK_TLSEXT_THMAC
Returns GSK_TLSEXT_THMAC_ON if the "Truncated HMAC" type TLS extension has been
negotiated and is in use. Returns GSK_TLSEXT_THMAC_OFF if the "Truncated HMAC" type TLS
extension has not been negotiated. GSK_TLSEXT_MFL can be specified only for an SSL
connection.

GSK_TLSEXT_SNI
Returns GSK_TLSEXT_SNI_ON if the “Server Name Indication” type TLS extension has been
negotiated and is in use. Returns GSK_TLSEXT_SNI_OFF if the "Server Name Indication" type
TLS extension has not been negotiated. GSK_TLSEXT_SNI can be specified only for an SSL
connection.

GSK_T61_AS_LATIN1
Returns GSK_T61_AS_LATIN1_ON if the ISO8859-1 character set will be used when converting a
string tagged as TELETEXSTRING or GSK_T61_AS_LATIN1_OFF if the T.61 character set will be
used. GSK_T61_AS_LATIN1 can be specified only for an SSL environment. The
GSK_T61_AS_LATIN1 setting is global and applies to all SSL environments.

Related Topics

- gsk_attribute_set_enum()
- gsk_environment_open()
- gsk_secure_socket_open()
gsk_attribute_get_numeric_value()

getattribute_get_numeric_value()

Gets a numeric value.

Format

```c
#include <gskssl.h>
gsk_status gsk_attribute_get_numeric_value (  
gsk_handle ssl_handle,  
GSK_NUM_ID num_id,  
int * num_value)
```

Parameters

- **ssl_handle**
  Specifies an SSL environment handle returned by `gsk_environment_open()` or an SSL connection handle returned by `gsk_secure_socket_open()`.

- **num_id**
  Specifies the numeric identifier.

- **num_value**
  Returns the numeric value.

Results

The function return value will be 0 (`GSK_OK`) if no error is detected. Otherwise, it will be one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

- **[GSK_ATTRIBUTE_INVALID_ID]**
  The numeric identifier is not valid or cannot be used with the specified handle.

- **[GSK_INVALID_HANDLE]**
  The handle is not valid.

- **[GSK_INVALID_STATE]**
  The environment is closed.

Usage

The `gsk_attribute_get_numeric_value()` routine will return a numeric value for an SSL environment or an SSL connection.

These numeric identifiers are supported:

- **GSK_CRL_CACHE_TIMEOUT**
  Returns the CRL cache timeout. `GSK_CRL_CACHE_TIMEOUT` can be specified only for an SSL environment.

- **GSK_FD**
  Returns the socket descriptor used for network operations. `GSK_FD` can be specified only for an SSL connection.

- **GSK_LDAP_SERVER_PORT**
  Returns the LDAP server port. `GSK_LDAP_SERVER_PORT` can be specified only for an SSL environment.

- **GSK_V2_SESSION_TIMEOUT**
  Returns the SSL Version 2 session timeout. `GSK_V2_SESSION_TIMEOUT` can be specified only for an SSL environment.
gsk_attribute_get_numeric_value()

GSK_V2_SIDCACHE_SIZE
Returns the size of the SSL Version 2 session identifier cache. GSK_V2_SIDCACHE_SIZE can be specified only for an SSL environment.

GSK_V3_SESSION_TIMEOUT
Returns the SSL Version 3 session timeout. GSK_V3_SESSION_TIMEOUT can be specified only for an SSL environment.

GSK_V3_SIDCACHE_SIZE
Returns the size of the SSL Version 3 session identifier cache. GSK_V3_SIDCACHE_SIZE can be specified only for an SSL environment.

Related Topics
- gsk_attribute_set_numeric_value()
- gsk_environment_open()
- gsk_secure_socket_open()
gsk_attribute_set_buffer()

Sets the value of an attribute buffer.

Format

```
#include <gskssl.h>

int gsk_attribute_set_buffer (gsk_handle ssl_handle, GSK_BUF_ID buffer_id, const char * buffer_value, int buffer_length);
```

Parameters

- **ssl_handle**: Specifies an SSL environment handle returned by `gsk_environment_open()` or an SSL connection handle returned by `gsk_secure_socket_open()`.
- **buffer_id**: Specifies the buffer identifier.
- **buffer_value**: Specifies the buffer value.
- **buffer_length**: Specifies the buffer length. Specify 0 for this parameter if the buffer value is a null-delimited character string.

Results

The function return value will be 0 (`GSK_OK`) if no error is detected. Otherwise, it will be one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

- **[GSK_ATTRIBUTE_INVALID_ID]**: The buffer identifier is not valid or cannot be used with the specified handle.
- **[GSK_ATTRIBUTE_INVALID_LENGTH]**: The buffer length is not valid.
- **[GSK_INSUFFICIENT_STORAGE]**: Insufficient storage is available.
- **[GSK_INVALID_HANDLE]**: The handle is not valid.
- **[GSK_INVALID_STATE]**: The environment or connection is not in the open state.

Usage

The `gsk_attribute_set_buffer()` routine will set a buffer value in an SSL environment or an SSL connection. The environment or connection must be in the open state and not in the initialized state (that is, `gsk_environment_init()` or `gsk_secure_socket_init()` has not been called).

These buffer identifiers are supported:

- **GSK_KEYRING_FILE**: Specifies the name of the key database HFS file, SAF key ring or z/OS PKCS #11 token. A key database is used if a database password or stash file is defined using either an environment
The SAF key ring name is specified as "userid/keyring". The current userid is used if the userid is omitted. The user must have READ access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by the user. The user must have UPDATE access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by another user. Note: Certificate private keys are not available when using a SAF key ring owned by another user. The z/OS PKCS #11 token name is specified as *TOKEN*/token-name. *TOKEN* indicates that the specified key ring is actually a token name. The application userid must have READ access to resource USER.token-name in the CRYPTOZ class in order for the certificates and their private keys, if present, to be read.

**GSK_KEYRING_LABEL**
- Specifies the label of the key used to authenticate the application. The default key will be used if a key label is not specified. GSK_KEYRING_LABEL may be specified for an SSL environment or an SSL connection. If either the GSK_CLIENT_CERT_CALLBACK function or the GSK_SNI_CALLBACK function is registered, the key label can be set or reset by the callback function after a call to gsk_secure_socket_init.

**GSK_KEYRING_PW**
- Specifies the password for the key database. GSK_KEYRING_PW may be specified only for an SSL environment.

**GSK_KEYRING_STASH_FILE**
- Specifies the name of the key database password stash file. The stash file name always has an extension of "sth" and the supplied name will be changed if it does not have the correct extension. The GSK_KEYRING_PW value will be used instead of the GSK_KEYRING_STASH_FILE value if it is also specified. GSK_KEYRING_STASH_FILE may be specified only for an SSL environment.

**GSK_LDAP_SERVER**
- Specifies one or more blank-separated LDAP server host names. Each host name can contain an optional port number separated from the host name by a colon. GSK_LDAP_SERVER may be specified only for an SSL environment. The LDAP server is used to obtain CA certificates when validating a certificate and the local database does not contain the required certificate. The local database must contain the required certificates if no LDAP server is specified. Even when an LDAP server is used, root CA certificates must be found in the local database since the LDAP server is not a trusted data source. The LDAP server is also used to obtain certificate revocation lists.

**GSK_LDAP_USER**
- Specifies the distinguished name to use when connecting to the LDAP server. GSK_LDAP_USER may be specified only for an SSL environment.

**GSK_LDAP_USER_PW**
- Specifies the password to use when connecting to the LDAP server. GSK_LDAP_USER_PW may be specified only for an SSL environment.

**GSK_USER_DATA**
- Specifies the user data to be passed to SSL exit routines. The user data is copied to storage owned by the SSL runtime and the address of this storage is passed to the SSL exit routines. The application may alter this copy of the user data but may not free it. GSK_USER_DATA may be specified only for an SSL connection.

**GSK_V2_CIPHER_SPECS**
- Specifies the SSL V2 cipher specifications as a string consisting of 1 or more 1-character values. GSK_V2_CIPHER_SPECS may be specified for an SSL environment or an SSL connection. Refer to the description of gsk_environment_open() for a list of valid cipher specifications.
GSK_V3_CIPHER_SPECS
Specifies the SSL V3 cipher specifications as a string consisting of 1 or more 2-character values. GSK_V3_CIPHER_SPECS may be specified for an SSL environment or an SSL connection. The SSL V3 cipher specifications are used for the SSL V3, TLS V1.0 and TLS V1.1 protocols. Refer to the description of **gsk_environment_open()** for a list of valid cipher specifications.

Related Topics
- [gsk_attribute_get_buffer()](#)
- [gsk_environment_open()](#)
- [gsk_environment_init()](#)
- [gsk_secure_socket_open()](#)
- [gsk_secure_socket_init()](#)
gsk_attribute_set_callback()

Sets an SSL callback.

Format

```c
#include <gskssl.h>
gsk_status gsk_attribute_set_callback (  
gsk_handle ssl_handle,  
GSK_CALLBACK_ID callback_id,  
void * callback)
```

Parameters

- **ssl_handle**
  Specifies an SSL environment handle returned by `gsk_environment_open()` or an SSL connection handle returned by `gsk_secure_socket_open()`.

- **callback_id**
  Specifies the callback identifier.

- **callback**
  Specifies the address of the callback parameter.

Results

The function return value will be 0 (`GSK_OK`) if no error is detected. Otherwise, it will be one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

- **[GSK_ATTRIBUTE_INVALID_ID]**
  The callback identifier is not valid or cannot be used with the specified handle.

- **[GSK_ATTRIBUTE_INVALID_PARAMETER]**
  The attribute parameter value is not valid.

- **[GSK_INVALID_HANDLE]**
  The handle is not valid.

- **[GSK_INVALID_STATE]**
  The environment or connection is not in the open state.

Usage

The `gsk_attribute_set_callback()` routine establishes a callback to an application routine by the SSL runtime. A callback allows the application to replace the default routine used by the SSL runtime. The SSL environment or SSL connection must be in the open state and not in the initialized state (that is, `gsk_environment_init()` or `gsk_secure_socket_init()` has not been called). The callback routine must use standard C linkage and not C++ linkage.

These callback identifiers are supported:

**GSK_CLIENT_CERT_CALLBACK**

Indicates the application is providing the routine to prompt a client user to select a certificate from a list during the client authentication process. The `callback` parameter is the address of this routine. The exit routine can obtain the user data address by calling the `gsk_attribute_get_buffer()` routine. The `gsk_attribute_set_buffer()` routine should be called to set the selected key label before returning from the callback routine. The function return value should be 0 if a key label has been set or GSK_ERR_NO_CERTIFICATE if no client certificate is to be used. `GSK_CLIENT_CERT_CALLBACK` can be specified only for an SSL environment.
gsk_attribute_set_callback()

This is the prototype for the callback routine provided by the application. It shows the parameters passed to the application callback and the value returned by the callback.

```c
int client_cert_callback (  
gsk_handle  soc_handle)
```

GSK_IO_CALLBACK

Indicates the application is providing the routines to perform read, write, and control functions. The `callback` parameter is the address of a `gsk_iocallback` structure. Each entry in the structure will override the corresponding SSL runtime routine. A NULL entry will cause the current callback routine to be used or the SSL runtime routine will be used if there is no callback routine.

GSK_IO_CALLBACK can be specified for an SSL environment or an SSL connection.

The routine specified by the `io_read` entry is used to read data from the network. The `fd` parameter is the socket descriptor, the `buffer` parameter is the address of the data buffer, the `count` parameter is the buffer size, and the `user_data` parameter is the user data address. The function return value should be 0 if the connection has been closed by the remote partner, -1 if an error is detected, or the number of bytes read from the network. The `errno` runtime variable. The default routine uses the `recv()` library routine to read data from the network.

```c
int io_read (  
    int  fd,  
    void * buffer,  
    int  count,  
    char * user_data)
```

The routine specified by the `io_write` entry is used to write data to the network. The `fd` parameter is the socket descriptor, the `buffer` parameter is the address of the data buffer, the `count` parameter is the data length, and the `user_data` parameter is the user data address. The function return value should be -1 if an error is detected or the number of bytes written to the network. The `errno` runtime variable. The default routine uses the `send()` library routine to write data to the network.

```c
int io_write (  
    int  fd,  
    void * buffer,  
    int  count,  
    char * user_data)
```

The routine specified by the `io_getpeerid` entry is used to get the 32-bit network identifier for the remote partner. The `fd` parameter is the socket descriptor and the `user_data` parameter is the user data address. However, the `io_getpeerid` entry is deprecated and should not be used since it does not support IPv6 networks which use a 16-byte network identifier. Instead, the `io_getpeername` entry should be used for both IPv4 and IPv6 networks. The `io_getpeerid` entry will not be used if the `io_getpeername` entry is not NULL.

```c
unsigned long io_getpeerid (  
    int  fd,  
    char * user_data)
```

The routine specified by the `io_setsockoptoptions` entry is used to set socket options. The `fd` parameter is the socket descriptor, the `cmd` parameter is the function to be performed, and the `user_data` parameter is the user data address. The return value should be -1 if an error is detected and 0 otherwise. The `errno` code is returned in the `errno` runtime variable. The `io_setsockoptoptions()` routine is called by the `gsk_secure_socket_init()` routine before initiating the SSL handshake (GSK_SET_SOCKET_STATE_FOR_HANDSHAKE) and again upon completion of the SSL handshake (GSK_SET_SOCKET_STATE_FOR_READ_WRITE). The default `io_setsockoptoptions()` routine puts the socket into blocking mode for GSK_SET_SOCKET_STATE_FOR_HANDSHAKE and restores the original mode for GSK_SET_SOCKET_STATE_FOR_READ_WRITE.

```c
int io_setsockoptoptions (  
    int  fd,  
    int  cmd,  
    char * user_data)
```
The routine specified by the `io_getpeername` entry is used to get the network identifier for the remote partner. The `fd` parameter is the socket descriptor, the `buffer` parameter is the address of the return buffer, the length parameter is the size of the return buffer, and the `user_data` parameter is the user data address. Upon return, the `length` parameter should contain the actual length of the network identifier. The function return value should be -1 if an error is detected and 0 otherwise. The error code is returned in the `errno` runtime variable. The default routine uses the `getpeername()` library routine and returns the IP address of the remote partner (4 bytes for IPv4 and 16 bytes for IPv6) followed by the 2-byte port number.

```c
int io_getpeername(
    int fd,
    void * buffer,
    int * length,
    char * user_data)
```

**GSK_SESSION_RESET_CALLBACK**

Indicates the application is providing the routines to be called when a session renegotiation has been initiated or completed in order to establish a new session key or have the session cipher reset. The callback parameter is the address of a `gsk_reset_callback` structure.

**GSK_SESSION_RESET_CALLBACK** can be specified for an SSL environment or an SSL connection. The callback is only invoked when using SSL V3, TLS V1.0 and TLS V1.1 protocols.

The routine specified by the `Reset_Init` entry is called when a session renegotiation has been initiated, and the SSL client has commenced the renegotiation process. The `con_handle` parameter is the handle for the SSL connection.

```c
void (Reset_Init) (gsk_handle con_handle)
```

The `Reset_Complete` routine is called when a session renegotiation has been completed. The `con_handle` parameter is the handle for the SSL connection.

```c
void (Reset_Complete) (gsk_handle con_handle)
```

**GSK_SID_CACHE_CALLBACK**

Indicates the application is providing the routines to maintain the session identifier cache. The callback parameter is the address of a `gsk_sidcache_callback` structure.

**GSK_SID_CACHE_CALLBACK** can be specified only for an SSL environment and will be used only for SSL servers (the internal cache is always used for SSL clients).

The routine specified by the `Get` entry is called to retrieve an entry from the session identifier cache. The `session_id` parameter is the session identifier, the `session_id_length` parameter is the length of the session identifier, and the `ssl_version` parameter is the SSL protocol version number (GSK_SSLVERSION_V2 or GSK_SSLVERSION_V3). The function return value is the address of the session data buffer or NULL if an error is detected. The `FreeDataBuffer` routine will be called to release the session data buffer when it is no longer needed by the SSL runtime.

```c
gsk_data_buffer * Get (const unsigned char * session_id,
    unsigned int session_id_length,
    gsk_sslversion ssl_version)
```

The routine specified by the `Put` entry is called to store an entry in the session identifier cache. The `ssl_session_data` parameter is the session data, the `session_id` parameter is the session identifier, the `session_id_length` parameter is the length of the session identifier, and the `ssl_version` parameter is the SSL protocol version number (GSK_SSLVERSION_V2 or GSK_SSLVERSION_V3). The function return value is ignored and can be a NULL address. The callback routine must make its own copy of the session data since the SSL structure will be released when the connection is closed.
**gsk_attribute_set_callback()**

```c
void gsk_attribute_set_callback (gsk_attribute_set_callback callback)
```

The callback routine does not have the capability to enforce the required use of the server name indication extension. The failure to select a key label will cause a fatal UNRECOGNIZED_NAME alert. To enforce such actions in conjunction with the callback routine the user must set the GSK_TLS_EXTID_SNI_SERVER_LABELS extension by calling the `attribute_set_tls_extension()` routine. The required and unrecognized_name_fatal fields of the extension must be set appropriately to achieve the desired outcome, although the serverKeyLabel list may be empty.

The function return value should be 0 if a key label has been set or GSK_ERR_UNRECOGNIZED_NAME if no server certificate is selected. Enforcement of the required and unrecognized_name_fatal settings occur on return from the callback routine.

GSK_SNI_CALLBACK can be specified only for an SSL environment.

This is the prototype for the callback routine provided by the application. It shows the parameters passed to the application callback and the value returned by the callback.

```c
int sni_callback (gsk_attribute_set_callback callback)
```
gsk_attribute_set_enum()

Sets an enumerated value.

Format

```c
#include <gskssl.h>

gsk_status gsk_attribute_set_enum (gsk_handle ssl_handle, GSK_ENUM_ID enum_id, GSK_ENUM_VALUE enum_value)
```

Parameters

- **ssl_handle**
  Specifies an SSL environment handle returned by `gsk_environment_open()` or an SSL connection handle returned by `gsk_secure_socket_open()`.

- **enum_id**
  Specifies the enumeration identifier.

- **enum_value**
  Specifies the enumeration value.

Results

The function return value will be 0 (`GSK_OK`) if no error is detected. Otherwise, it will be one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

- **[GSK_ATTRIBUTE_INVALID_ID]**
  The enumeration identifier is not valid or cannot be used with the specified handle.

- **[GSK_INVALID_HANDLE]**
  The handle is not valid.

- **[GSK_INVALID_STATE]**
  The environment or connection is not in the open state.

Usage

The `gsk_attribute_set_enum()` routine will set an enumerated value for an SSL environment or an SSL connection. The environment or connection must be in the open state and not in the initialized state (that is, `gsk_environment_init()` or `gsk_secure_socket_init()` has not been called).

These enumeration identifiers are supported:

- **GSK_CERT_VALIDATION_MODE**
  Specifies the method of certificate validation. RFC2459 and RFC3280 describe differing methods of certificate validation. Specify `GSK_CERT_VALIDATION_MODE_2459` if certificate validation according to the RFC2459 method is required or `GSK_CERT_VALIDATION_MODE_3280` if certificate validation according to the RFC3280 method is required.

- **GSK_CERT_VALIDATION_MODE_ANY**
  Specify `GSK_CERT_VALIDATION_MODE_ANY` if certificate validation can use any supported X.509 certificate validation method.

- **GSK_CRL_SECURITY_LEVEL**
  Specify the level of security to be used when contacting an LDAP server. In order to check for revoked certificates in a Certificate Revocation List (CRL), CRLs located will be cached according to the `GSK_CRL_CACHE_TIMEOUT` setting of the SSL environment. To enforce contact with the
LDAP server for each CRL check, CRL caching must be disabled. If a CRL is not defined an empty CRL will be placed in the CRL cache to prevent repeated calls to the LDAP server. This entry will not be cleared until the CRL cache timeout is reached. See "gsk_attribute_set_numeric_value()" on page 62 and Appendix A, "Environment Variables," on page 481 for additional information on the GSK_CRL_CACHE_TIMEOUT setting.

GSK_CRL_SECURITY_LEVEL can only be specified at the environment level.

Three levels of security are available:

- GSK_CRL_SECURITY_LEVEL_LOW - Certificate validation will not fail if the LDAP server cannot be contacted.
- GSK_CRL_SECURITY_LEVEL_MEDIUM - Certificate validation requires the LDAP server to be contactable, but does not require a CRL to be defined. This is the default.
- GSK_CRL_SECURITY_LEVEL_HIGH - Certificate validation requires the LDAP server to be contactable, and a CRL to be defined.

GSK_CLIENT_AUTH_ALERT

Specify GSK_CLIENT_AUTH_NOCERT_ALERT_OFF if the SSL server application is to allow client connections where client authentication has been requested and the client fails to supply an X.509 certificate. Specify GSK_CLIENT_AUTH_NOCERT_ALERT_ON if the SSL server application is to terminate client connections where client authentication has been requested and the client fails to supply an X.509 certificate.

GSK_CLIENT_AUTH_ALERT can be specified only for an SSL environment and is only applicable for server sessions with client authentication active.

GSK_CLIENT_AUTH_TYPE

Specifies GSK_CLIENT_AUTH_FULL_TYPE to validate client certificates. If a certificate is not valid, the connection is not started and an error code is returned by the gsk_secure_socket_init() routine. If an LDAP server is specified, the LDAP server is queried for CA certificates and certificate revocation lists. If the LDAP server is not available, only local validation will be performed. If no client certificate is received and either GSK_CLIENT_AUTH_ALERT is not specified or is set to GSK_CLIENT_AUTH_NOCERT_ALERT_OFF, the connection will be successful. The application can check for this case by calling the gsk_attribute_get_cert_info() routine and checking for a NULL return address.

Specify GSK_CLIENT_AUTH_PASSTHRU_TYPE to bypass client certificate validation. The application can retrieve the certificate by calling the gsk_attribute_get_cert_info() routine.

GSK_CLIENT_AUTH_TYPE can be specified only for an SSL environment and is only applicable for server sessions with client authentication active.

GSK_PROTOCOL_SSLV2

Specifies GSK_PROTOCOL_SSLV2_ON to enable the SSL Version 2 protocol or GSK_PROTOCOL_SSLV2_OFF to disable the SSL Version 2 protocol. The SSL V2 protocol should be disabled whenever possible since the SSL V3 protocol provides significant security enhancements.

GSK_PROTOCOL_SSLV2 can be specified for an SSL environment or an SSL connection.

When operating in FIPS mode, the SSL Version 2 protocol will not be used. Enabling this protocol will have no effect.

When TLS extensions are defined for the client and any of the TLS protocols are enabled for the connection, the SSL Version 2 protocol will not be used. Enabling this protocol will have no effect.

GSK_PROTOCOL_SSLV3

Specifies GSK_PROTOCOL_SSLV3_ON to enable the SSL Version 3 protocol or GSK_PROTOCOL_SSLV3_OFF to disable the SSL Version 3 protocol.

GSK_PROTOCOL_SSLV3 can be specified for an SSL environment or an SSL connection.
When operating in FIPS mode, the SSL Version 3 protocol will not be used. Enabling this protocol will have no effect.

**GSK_PROTOCOL_TLSV1**
- Specifies `GSK_PROTOCOL_TLSV1_ON` to enable the TLS Version 1.0 protocol or `GSK_PROTOCOL_TLSV1_OFF` to disable the TLS Version 1.0 protocol.
- `GSK_PROTOCOL_TLSV1` can be specified for an SSL environment or an SSL connection.

**GSK_PROTOCOL_TLSV1_1**
- Specifies `GSK_PROTOCOL_TLSV1_1_ON` to enable the TLS Version 1.1 protocol or `GSK_PROTOCOL_TLSV1_1_OFF` to disable the TLS Version 1.1 protocol.
- `GSK_PROTOCOL_TLSV1_1` can be specified for an SSL environment or an SSL connection.

**GSK_SESSION_TYPE**
- Specifies `GSK_CLIENT_SESSION` to perform the SSL handshake as a client,
- `GSK_SERVER_SESSION` to perform the SSL handshake as a server,
- or `GSK_SERVER_SESSION_WITH_CL_AUTH` to perform the SSL handshake as a server requiring client authentication.
- `GSK_SESSION_TYPE` can be specified for an SSL environment or an SSL connection.

**GSK_SYSPLEX_SIDCACHE**
- Returns `GSK_SYSPLEX_SIDCACHE_ON` if sysplex session caching is enabled for this application or `GSK_SYSPLEX_SIDCACHE_OFF` if sysplex session caching is not enabled.
- `GSK_SYSPLEX_SIDCACHE` can be specified only for an SSL environment.

**GSK_T61_AS_LATIN1**
- Specify `GSK_T61_AS_LATIN1_ON` to use the ISO8859-1 character set when processing a TELETEX string. Specify `GSK_T61_AS_LATIN1_OFF` to use the T.61 character set. The default is to use the ISO8859-1 character set. Note that selecting the incorrect character set can cause strings to be converted incorrectly. `GSK_T61_AS_LATIN1` can be specified only for an SSL environment. This setting is global and affects all string conversions for all SSL environments.

**Related Topics**
- `gsk_attribute_get_enum()`
- `gsk_environment_open()`
- `gsk_environment_init()`
- `gsk_secure_socket_open()`
- `gsk_secure_socket_init()`
gsk_attribute_set_numeric_value()

Sets a numeric value.

**Format**

```c
#include <gskssl.h>

gsk_status gsk_attribute_set_numeric_value (  
gsk_handle ssl_handle,  
GSK_NUM_ID num_id,  
int num_value)
```

**Parameters**

- **ssl_handle**
  Specifies an SSL environment handle returned by `gsk_environment_open()` or an SSL connection handle returned by `gsk_secure_socket_open()`.

- **num_id**
  Specifies the numeric identifier.

- **num_value**
  Specifies the numeric value.

**Results**

The function return value will be 0 (`GSK_OK`) if no error is detected. Otherwise, it will be one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

- **[GSK_ATTRIBUTE_INVALID_ID]**
  The numeric identifier is not valid or cannot be used with the specified handle.

- **[GSK_ATTRIBUTE_INVALID_NUMERIC_VALUE]**
  The numeric value is not within the valid range.

- **[GSK_INVALID_HANDLE]**
  The handle is not valid.

- **[GSK_INVALID_STATE]**
  The environment or connection is not in the open state.

**Usage**

The `gsk_attribute_set_numeric_value()` routine will set a numeric value for an SSL environment or an SSL connection. The environment or connection must be in the open state and not in the initialized state (that is, `gsk_environment_init()` or `gsk_secure_socket_init()` has not been called).

These numeric identifiers are supported:

- **GSK_CRL_CACHE_TIMEOUT**
  Sets the CRL cache timeout. This is the number of hours that a cached CRL will remain valid. The range is 0-720 and defaults to 24. A value of 0 will disable CRL caching.
  `GSK_CRL_CACHE_TIMEOUT` can be specified only for an SSL environment.

- **GSK_LDAP_SERVER_PORT**
  Sets the LDAP server port. The port must be between 1 and 65535. Port 389 will be used if no

```
gsk_attribute_set_numeric_value()

LDAP server port is set. GSK_LDAP_SERVER_PORT can be specified only for an SSL environment. GSK_LDAP_SERVER_PORT can be specified only for an SSL environment.

GSK_V2_SESSION_TIMEOUT
Sets the SSL Version 2 session timeout. This is the number of seconds until an SSL V2 session identifier expires. The range is 0-100 and defaults to 100. System SSL will remember SSL V2 session identifiers for this amount of time. This reduces the amount of data exchanged during the SSL handshake when a complete initial handshake has already been performed. Session identifiers will not be remembered if a value of 0 is specified. GSK_V2_SESSION_TIMEOUT can be specified only for an SSL environment.

GSK_V2_SIDCACHE_SIZE
Sets the size of the SSL Version 2 session identifier cache. The oldest entry will be removed when the cache is full in order to add a new entry. The range is 0-32000 and defaults to 256. Session identifiers will not be remembered if a value of 0 is specified. The session identifier cache will be allocated using the requested size rounded up to a power of 2 with a minimum size of 16. GSK_V2_SIDCACHE_SIZE can be specified only for an SSL environment.

GSK_V3_SESSION_TIMEOUT
Sets the session timeout for the SSL V3, TLS V1.0 and TLS V1.1 protocols. This is the number of seconds until an SSL V3 session identifier expires. The range is 0-86400 and defaults to 86400. System SSL will remember session identifiers for this amount of time. This reduces the amount of data exchanged during the SSL handshake when a complete initial handshake has already been performed. Session identifiers will not be remembered if a value of 0 is specified. GSK_V3_SESSION_TIMEOUT can be specified only for an SSL environment.

GSK_V3_SIDCACHE_SIZE
Sets the size of the SSL Version 3 session identifier cache. The oldest entry will be removed when the cache is full in order to add a new entry. The range is 0-64000 and defaults to 512. Session identifiers will not be remembered if a value of 0 is specified. The SSL V3 session cache is used for the SSL V3, TLS V1.0 and TLS V1.1 protocols. The session identifier cache will be allocated using the requested size rounded up to a power of 2 with a minimum size of 16. GSK_V3_SIDCACHE_SIZE can be specified only for an SSL environment.

Related Topics
gsk_attribute_get_numeric_value()
gsk_environment_open()
gsk_environment_init()
gsk_secure_socket_init()
gsk_secure_socket_open()
gsk_attribute_set_tls_extension()

Defines a TLS extension to the SSL environment or connection.

Format

```c
#include <gskssl.h>
gsk_attribute_set_tls_extension(
    gsk_handle ssl_handle,
    gsk_tls_extension * tls_extension)
```

Parameters

- `ssl_handle` Specifies an SSL environment handle returned by `gsk_environment_open()` or an SSL connection handle returned by `gsk_secure_socket_open()`.
- `tls_extension` Specifies the TLS extension structure containing extension data.

Results

The function return value will be 0 (`GSK_OK`) if no error is detected. Otherwise, it will be one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

- `[GSK_ATTRIBUTE_INVALID_TLS_EXTENSION]` The TLS extension type identifier is not valid or cannot be used with the specified handle.
- `[GSK_ATTRIBUTE_INVALID_TLS_EXT_DATA]` TLS extension data has been incorrectly defined.
- `[GSK_INVALID_HANDLE]` The handle is not valid.
- `[GSK_INVALID_STATE]` The handle is closed.

Usage

The `gsk_attribute_set_tls_extension()` routine will define a TLS extension for an SSL environment or an SSL connection. The environment or connection must be in the open state and not in the initialized state (that is, `gsk_environment_init()` or `gsk_secure_socket_init()` has not been called). TLS Extensions defined for an SSL environment will apply to all connections made as part of that environment unless explicitly deactivated or replaced using a call to `gsk_attribute_set_tls_extension()` for the connection. TLS extensions are applied to TLS V1.0 and TLS V1.1 connections only.

The application must prime the TLS extension structure with the appropriate TLS extension data prior to calling the routine, including the TLS extension type identifier and the specific data required for the TLS extension type. The TLS extension may be designated as required or optional in the `gsk_tls_extension` structure. A required setting will enforce support requirements of the specific extension type on the communicating partner. If the partner indicates it does not support the extension, the connection will be rejected. An optional setting will allow the connection to continue without support for that particular extension type if the communicating partner indicates it does not support the TLS extension type.

Notes:

1. Setting an extension as required for a server will mean that all clients connecting to the server must have the extension enabled. Failure for a client to do so will result in the server rejecting the connection request from the client. It is recommended that for maximum interoperability, that the required field not be enabled on the server side.
2. The gsk_tls_extension structure contains a 32-byte field, rsvd, which is reserved for future use. This field must contain binary zeroes; any non-zero data will result in gsk_attribute_set_tls_extension() returning a GSK_ATTRIBUTE_INVALID_TLS_EXT_DATA error.

3. Definition of TLS extensions for the client when any of the TLS protocols are enabled will prevent the SSL V2 protocol from being used.

These TLS extension type identifiers are supported:

**GSK_TLS_EXTID_SNI_SERVER_LABELS**
Specifies the pairings of server name to certificate key label to be used when the TLS server receives a 'Server Name Indication' type TLS extension from the TLS client. The server name/key label pairs are used in conjunction with the server name details received from the client to determine which certificate from the key database, key ring or token will be sent to the client as the server's certificate.

Set the setSni setting of the gsk_sni_server_labels extension data to TRUE to register the extension data with the SSL environment or connection. A setSni setting of FALSE will deactivate a previously registered GSK_TLS_EXTID_SNI_SERVER_LABELS type TLS extension setting.

If the TLS server does not recognize any server names in the client's server name list the server will send an 'unrecognized_name' alert to the client, which, by default, is a warning. Set the unrecognized_name_fatal flag in the gsk_sni_server_labels extension data to TRUE to treat the 'unrecognized_name' alert as fatal and close the connection.

GSK_TLS_EXTID_SNI_SERVER_LABELS can be defined on both the server and client sides. Its settings however will only have effect when running as a server; it will be ignored for clients.

Notes:
1. It is recommended that the gsk_sni_server_labels structure to be included in the gsk_tls_extension data be initialized with binary zeroes prior to setting the required server label data. This will ensure future application compatibility when additional bits within the gsk_sni_server_labels structure are utilized.
2. System SSL only supports server names that contain US-ASCII charaters.

**GSK_TLS_EXTID_SNI_CLIENT_SNAMES**
Specifies the server name (or list of server names) that the client will send to the server in a 'Server Name Indication' type TLS extension to indicate with which server the client wishes to communicate. The list of servernames is defined using a pointer to an array of pointers to strings containing the server names.

Set the setSni setting of the gsk_sni_client_names extension data to TRUE to register the extension data with the SSL environment or connection. A setSni setting of FALSE will deactivate a previously registered GSK_TLS_EXTID_SNI_CLIENT_SNAMES type TLS extension setting.

If the TLS server does not recognize any server names in the client's server name list the server will send an 'unrecognized_name' alert to the client, which, by default, is a warning. Set the unrecognized_name_fatal flag in the gsk_sni_client_names extension data to TRUE to treat the 'unrecognized_name' alert as fatal and close the connection.

GSK_TLS_EXTID_SNI_CLIENT_SNAMES can be defined on both the server and client sides. Its settings however will only have effect when running as a client; it will be ignored for servers.

Notes:
1. It is recommended that the gsk_sni_client_names structure to be included in the gsk_tls_extension data be initialized with binary zeroes prior to setting the required server label data. This will ensure future application compatibility when additional bits within the gsk_sni_client_names structure are utilized.
2. System SSL only supports server names that contain US-ASCII charaters.
gsk_attribute_set_tls_extension()

- **GSK_TLS_EXTID_SERVER_MFL**
  - Specifies the 'Maximum Fragment Length' type TLS extension requirements for the TLS server.
  - Specify to the TLS server whether to support the 'Maximum Fragment Length' TLS extension using
    the GSK_TLS_MFL_ON setting. The GSK_TLS_MFL_OFF setting will deactivate a previously
    registered GSK_TLS_EXTID_SERVER_MFL type TLS extension setting.

- **GSK_TLS_EXTID_CLIENT_MFL**
  - Specifies the 'Maximum Fragment Length' type TLS extension requirements for the TLS client.
  - Specify the size of the maximum fragment length to be used using settings GSK_TLS_MFL_512
    (2^9 bytes), GSK_TLS_MFL_1024 (2^10), GSK_TLS_MFL_2048 (2^11) or GSK_TLS_MFL_4096 (2^12).
  - The GSK_TLS_MFL_OFF setting will deactivate a previously registered
    GSK_TLS_EXTID_CLIENT_MFL type TLS extension setting.

- **GSK_TLS_EXTID_TRUNCATED_HMAC**
  - Specifies whether the TLS server or client will support the 'Truncated HMAC' type TLS extension.
  - Set truncateHmac to TRUE to enable the extension. A truncateHmac setting of FALSE will
    deactivate a previously registered GSK_TLS_EXTID_TRUNCATED_HMAC type TLS extension
    setting.
gsk_environment_close()

Closes a SSL environment.

Format

```c
#include <gskssl.h>

gsk_status gsk_environment_close ( gsk_handle * env_handle)
```

Parameters

`env_handle`

Specifies the SSL environment handle returned by the `gsk_environment_open()` routine. The environment handle will be set to NULL upon completion.

Results

The function return value will be 0 (`GSK_OK`) if no error is detected. Otherwise, it will be one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

- `[GSK_INVALID_HANDLE]`
  - The environment handle is not valid.
- `[GSK_INVALID_STATE]`
  - The environment is already closed.

Usage

The `gsk_environment_close()` routine will close an environment created by the `gsk_environment_open()` routine. The storage allocated for the environment will not be released until all connections created using the environment have been closed. The SSL environment cannot be used to create new connections upon completion of the close.

Related Topics

- `gsk_environment_open()`
- `gsk_environment_init()`
- `gsk_secure_socket_init()`
- `gsk_secure_socket_close()`
gsk_environment_init()

Initializes a SSL environment.

Format

```c
#include <gskssl.h>

gsk_status gsk_environment_init (gsk_handle env_handle)
```

Parameters

`env_handle` 
Specifies the SSL environment handle returned by the `gsk_environment_open()` routine.

Results

The function return value will be 0 (GSK_OK) if no error is detected. Otherwise, it will be one of the return codes listed in the gskssl.h include file. These are some possible errors:

- **[GSK_CERTIFICATE_NOTAVAILABLE]**
  The key database, key ring or token does not contain any certificates.

- **[GSK_ERR_BAD_KEYFILE_PASSWORD]**
  The key database password is not correct.

- **[GSK_ERR_LDAP]**
  Unable to initialize the LDAP client.

- **[GSK_ERR_LDAP_NOTAVAILABLE]**
  The LDAP server is not available.

- **[GSK_ERR_PERMISSION_DENIED]**
  Not authorized to access key database, SAF key ring or z/OS PKCS #11 token.

- **[GSK_INSUFFICIENT_STORAGE]**
  Insufficient storage is available.

- **[GSK_INVALIDHANDLE]**
  The environment handle is not valid.

- **[GSK_INVALID_STATE]**
  The environment is not in the open state.

- **[GSK_KEYFILEINVALID_FORMAT]**
  The database is not a key database.

- **[GSK_KEYFILE_IO_ERR]**
  An input/output error occurred while reading the key database, key ring or token.

- **[GSK_KEYFILE_PASSWORD_EXPIRED]**
  The key database password is expired.

- **[GSK_KEYRING_OPEN_ERROR]**
  Unable to open the key database, key ring or token.

- **[GSK_NOKEYFILE_PASSWORD]**
  The key database password is not available.

Usage

The `gsk_environment_init()` routine initializes an SSL environment created by the `gsk_environment_open()` routine. After the SSL environment has been initialized, it can be used to create
one or more SSL connections by calling the `gsk_secure_socket_open()` routine. The `gsk_environment_close()` routine should be called to close the environment when it is no longer needed. The `gsk_environment_close()` routine should also be called if an error is returned by the `gsk_environment_init()` routine.

**Related Topics**
- `gsk_environment_open()`
- `gsk_environment_close()`
- `gsk_secure_socket_open()`
gsk_environment_open()

Creates a SSL environment.

Format

```
#include <gskssl.h>

gsk_status gsk_environment_open (
    gsk_handle * env_handle)
```

Parameters

`env_handle`

Returns the handle for the environment. The application should call the `gsk_environment_close()` routine to release the environment when it is no longer needed.

Results

The function return value will be 0 (GSK_OK) if no error is detected. Otherwise, it will be one of the return codes listed in the gskssl.h include file. These are some possible errors:

[GSK_ATTRIBUTE_INVALID_ENUMERATION]

The value of an environment variable is not valid.

[GSK_ATTRIBUTE_INVALID_LENGTH]

The length of an environment variable value is not valid.

[GSK_ATTRIBUTE_INVALID_NUMERIC_VALUE]

The value of an environment variable is not valid.

[GSK_INSUFFICIENT_STORAGE]

Insufficient storage is available.

Usage

The `gsk_environment_open()` routine creates an SSL environment. The environment will be initialized with default values and then any SSL environment variables will be processed. These values can be changed by the application using the appropriate `gsk_attribute_set_*()` routines. The `gsk_environment_init()` routine should then be called to initialize the SSL environment. This environment can then be used to establish one or more SSL connections.

- When not executing in FIPS mode, the following default values are set:
  - SSL V2, SSL V3, and TLS V1.0 are enabled (TLS V1.1 is disabled by default)
  - The connection type is set to CLIENT
  - The SSL V2 connection timeout is set to 100 seconds
  - The SSL V3 connection timeout is set to 86400 seconds
  - The SSL V2 cache size is set to 256
  - The SSL V3 cache size is set to 512
  - The sysplex session cache is disabled
  - The default key will be used
  - No revoked certificate checking is performed
  - The default callback routines will be used
  - The SSL V2 cipher specification is set to “713642” if domestic encryption is enabled and “642” otherwise
gsk_environment_open()

- The SSL V3 cipher specification is set to "050435363738392F303132330A1613100D0915120F0C0306020100" if domestic encryption is enabled and "0915120F0C0306020100" otherwise
- No TLS extensions are initialized

When executing in FIPS mode, the following default values are set:
- TLS V1.0 is enabled (TLS V1.1 is disabled by default)
- The connection type is set to CLIENT
- The connection timeout is set to 86400 seconds
- The cache size is set to 512
- The sysplex session cache is disabled
- The default key will be used
- No revoked certificate checking performed
- The default callback routines will be used
- The cipher specification is set to "35363738392F303132330A1613100D"

These SSL V2 cipher specifications are supported:
- "1" = 128-bit RC4 encryption with MD5 message authentication (128-bit secret key)
- "2" = 128-bit RC4 export encryption with MD5 message authentication (40-bit secret key)
- "3" = 128-bit RC2 encryption with MD5 message authentication (128-bit secret key)
- "4" = 128-bit RC2 export encryption with MD5 message authentication (40-bit secret key)
- "6" = 56-bit DES encryption with MD5 message authentication (56-bit secret key)
- "7" = 168-bit Triple DES encryption with MD5 message authentication (168-bit secret key)

These SSL V3 cipher specifications are supported:
- "00" = No encryption or message authentication and RSA key exchange
- "01" = No encryption with MD5 message authentication and RSA key exchange
- "02" = No encryption with SHA-1 message authentication and RSA key exchange
- "03" = 40-bit RC4 encryption with MD5 message authentication and RSA key exchange
- "04" = 128-bit RC4 encryption with MD5 message authentication and RSA key exchange
- "05" = 128-bit RC4 encryption with SHA-1 message authentication and RSA key exchange
- "06" = 40-bit RC2 encryption with MD5 message authentication and RSA key exchange
- "09" = 56-bit DES encryption with SHA-1 message authentication and RSA key exchange
- "0A" = 168-bit Triple DES encryption with SHA-1 message authentication and RSA key exchange
- "0C" = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate
- "0D" = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate
- "0F" = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "10" = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "12" = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate
- "13" = 168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate
- "15" = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
gsk_environment_open()

- "16" = 168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
- "2F" = 128-bit AES encryption with SHA-1 message authentication and RSA key exchange
- "30" = 128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate
- "31" = 128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "32" = 128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate
- "33" = 128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
- "35" = 256-bit AES encryption with SHA-1 message authentication and RSA key exchange
- "36" = 256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate
- "37" = 256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "38" = 256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate
- "39" = 256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate

If executing in FIPS mode, only the following cipher specifications are supported:

- 0A 0D 10 13 16 2F 30 31 32 33 35 36 37 38 39

If using the TLS V1.1 protocol, export ciphers are not supported. The 40 bit ciphers (cipher specifications "03" and "06") will be ignored if specified.

These environment variables are processed:

**GSK_CRL_SECURITY_LEVEL**
Specifies the level of security SSL applications will use when contacting LDAP servers to check CRLs for revoked certificates during certificate validation.

**GSK_KEY_LABEL**
Specifies the label of the key used to authenticate the application. The default key will be used if a key label is not specified.

**GSK_KEYRING_FILE**
Specifies the name of the key database HFS file, SAF key ring or z/OS PKCS #11 token. A key database is used if a database password or stash file is defined using either an environment variable or the gsk_attribute_set_buffer() routine. Otherwise a SAF key ring or z/OS PKCS #11 token is used. GSK_KEYRING_FILE may be specified only for an SSL environment.

The SAF key ring name is specified as "userid/keyring". The current userid is used if the userid is omitted. The user must have READ access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by the user. The user must have UPDATE access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by another user. Note: Certificate private keys are not available when using a SAF key ring owned by another user. The z/OS PKCS #11 token name is specified as *TOKEN*/token-name. *TOKEN* indicates that the specified key ring is actually a token name. The application userid must have READ access to resource USER.token-name in the CRYPTOZ class in order for the certificates and their private keys, if present, to be read.

**GSK_KEYRING_PW**
Specifies the password for the key database.
GSK_KEYRING_STASH
Specifies the name of the key database password stash file. The stash file name always has an extension of "*.sth" and the supplied name will be changed if it does not have the correct extension. The GSK_KEYRING_PW environment variable will be used instead of the GSK_KEYRING_STASH environment variable if it is also specified.

GSK_LDAP_SERVER
Specifies one or more blank-separated LDAP server host names. Each host name can contain an optional port number separated from the host name by a colon. The LDAP server is used to obtain CA certificates when validating a certificate and the local database does not contain the required certificate. The local database must contain the required certificates if no LDAP server is specified. Even when an LDAP server is used, root CA certificates must be found in the local database since the LDAP server is not a trusted data source. The LDAP server is also used to obtain certificate revocation lists.

GSK_LDAP_PASSWORD
Specifies the password to use when connecting to the LDAP server.

GSK_LDAP_PORT
Specifies the LDAP server port. Port 389 will be used if no LDAP server port is specified.

GSK_LDAP_USER
Specifies the distinguished name to use when connecting to the LDAP server.

GSK_PROTOCOL_SSLV2
Specifies whether the SSL V2 protocol is supported. A value of 0, OFF or DISABLED will disable the SSL V2 protocol while a value of 1, ON or ENABLED will enable the SSL V2 protocol. The SSL V2 protocol should be disabled whenever possible since the SSL V3 protocol provides significant security enhancements.

When operating in FIPS mode, SSL Version 2 protocol will not be used. Enabling this protocol will have no effect.

When TLS extensions are defined for the client and any of the TLS protocols are also enabled, the SSL Version 2 protocol will not be used. Enabling this protocol will have no effect.

GSK_PROTOCOL_SSLV3
Specifies whether the SSL V3 protocol is supported. A value of 0, OFF or DISABLED will disable the SSL V3 protocol while a value of 1, ON or ENABLED will enable the SSL V3 protocol.

When operating in FIPS mode, SSL Version 3 protocol will not be used. Enabling this protocol will have no effect.

GSK_PROTOCOL_TLSV1
Specifies whether the TLS V1.0 protocol is supported. A value of 0, OFF or DISABLED will disable the TLS V1.0 protocol while a value of 1, ON or ENABLED will enable the TLS V1.0 protocol. The TLS V1.0 protocol uses the same session cache and cipher specifications as the SSL V3 protocol.

GSK_PROTOCOL_TLSV1_1
Specifies whether the TLS V1.1 protocol is supported. A value of 0, OFF or DISABLED will disable the TLS V1.1 protocol while a value of 1, ON or ENABLED will enable the TLS V1.1 protocol. The TLS V1.1 protocol uses the same session cache and cipher specifications as the SSL V3 protocol. The TLS V1.1 protocol will not use export (40-bit) ciphers. They will be ignored if TLS V1.1 is negotiated as the communications protocol.

GSK_SYSPLEX_SIDCACHE
Specifies whether sysplex session caching is supported for this application. A value of 0, OFF or DISABLED will disable sysplex session caching while a value of 1, ON or ENABLED will enable sysplex session caching.

GSK_V2_CIPHER_SPECS
Specifies the SSL V2 cipher specifications in order of preference as a null-terminated string

Chapter 7. API Reference  73
**gsk_environment_open()**

consisting of 1 or more 1-character values. Valid cipher specifications that are not supported due to the installed cryptographic level will be skipped when the connection is initialized.

**GSK_V2_SESSION_TIMEOUT**
Specifies the session timeout value in seconds for the SSL V2 protocol. The valid timeout values are 0 through 100 and defaults to 100.

**GSK_V2_SIDCACHE_SIZE**
Specifies the number of session identifiers that can be contained in the SSL V2 cache. The valid cache sizes are 0 through 32000 and defaults to 256. The SSL V2 cache will be disabled if 0 is specified.

**GSK_V3_CIPHER_SPECS**
Specifies the SSL V3 cipher specifications in order of preference as a null-terminated string consisting of 1 or more 2-character values. The SSL V3 cipher specifications are used for the SSL V3, TLS V1.0 and TLS V1.1 protocols. Valid cipher specifications that are not supported due to the installed cryptographic level will be skipped when the connection is initialized. The TLS V1.1 protocol will not use export (40-bit) ciphers. They will be ignored if TLS V1.1 is negotiated as the communications protocol.

**GSK_V3_SESSION_TIMEOUT**
Specifies the session timeout value in seconds for the SSL V3, TLS V1.0 and TLS V1.1 protocols. The valid timeout values are 0 through 86400 and defaults to 86400.

**GSK_V3_SIDCACHE_SIZE**
Specifies the number of session identifiers that can be contained in the SSL V3 cache. The valid cache sizes are 0 through 64000 and defaults to 512. The SSL V3 cache will be disabled if 0 is specified. The SSL V3 cache is used for the SSL V3, TLS V1.0 and TLS V1.1 protocols.

**Related Topics**

- gsk_environment_init()
- gsk_environment_close()
gsk_free_cert_data()

Releases the storage allocated for a certificate data array.

Format

```c
#include <gskssl.h>

void gsk_free_cert_data ( gsk_cert_data_elem * cert_data, int elem_count )
```

Parameters

cert_data
   Specifies the certificate data array to be released.

elem_count
   Specifies the number of elements in the certificate data array.

Usage

The `gsk_free_cert_data()` routine will release the storage allocated for an array of certificate data elements.

Related Topics

gsk_attribute_get_cert_info()
gsk_get_cert_by_label()
gsk_get_cert_by_label()

**gsk_get_cert_by_label()**

Gets certificate information for a record label.

**Format**

```c
#include <gskssl.h>

# gsk_status gsk_get_cert_by_label ( 
  gsk_handle ssl_handle, 
  const char * record_label, 
  gsk_cert_data_elem ** cert_data, 
  int * elem_count)
```

**Parameters**

- **ssl_handle**
  Specifies an SSL environment handle returned by `gsk_environment_open()` or an SSL connection handle returned by `gsk_secure_socket_open()`.

- **record_label**
  Specifies the record label for the certificate.

- **cert_data**
  Returns the certificate data array. The `gsk_free_cert_data()` routine should be called to release the array when the certificate information is no longer needed.

- **elem_count**
  Returns the number of elements in the array of gsk_cert_data_elem structures.

**Results**

The function return value will be 0 (`GSK_OK`) if no error is detected. Otherwise, it will be one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

- **[GSK_ERR ASN]**
  Unable to decode certificate.

- **[GSK_ERR_MULTIPLE_LABEL]**
  Multiple certificates exist for label.

- **[GSK_INSUFFICIENT_STORAGE]**
  Insufficient storage is available.

- **[GSK_INVALID_HANDLE]**
  The handle is not valid.

- **[GSK_KEY_LABEL_NOT_FOUND]**
  The key record is not found.

**Usage**

The `gsk_get_cert_by_label()` routine returns certificate information for a record label. The supplied handle can be for an SSL environment or an SSL connection.

Each element of the certificate data array has an element identifier. The element identifiers used for a particular certificate will depend upon the contents of the certificate. These element identifiers are currently provided:

- **CERT BODY BASE64**
  Certificate body in Base64-encoded format
gsk_get_cert_by_label()

**CERT_BODY_DER**
Certificate body in binary ASN.1 DER-encoded format

**CERT_COMMON_NAME**
Subject common name (CN)

**CERT_COUNTRY**
Subject country (C)

**CERT_DN_DER**
Subject distinguished name in binary ASN.1 DER-encoded format

**CERT_DN_PRINTABLE**
Subject distinguished name as a printable character string
These DN attribute names are recognized by the System SSL runtime.
- C - Country
- CN - Common name
- DC - Domain component
- DNQUALIFIER - Distinguished name qualifier
- EMAIL - E-mail address
- GENERATIONQUALIFIER - Generation qualifier
- GIVENNAME - Given name
- INITIALS - Initials
- L - Locality
- MAIL - RFC822 style address
- NAME - Name
- O - Organization name
- OU - Organizational unit name
- PC - Postal code
- SERIALNUMBER - Serial number
- SN - Surname
- ST - State or province
- STREET - Street
- T - Title

**CERT_DNQUALIFIER**
Subject distinguished name qualifier (DNQUALIFIER)

**CERT_DOMAIN_COMPONENT**
Subject domain component (DC)

**CERT_EMAIL**
Subject e-mail address (EMAIL)

**CERT_GENERATIONQUALIFIER**
Subject generation qualifier (GENERATIONQUALIFIER)

**CERT_GIVENNAME**
Subject given name (GIVENNAME)

**CERT_INITIALS**
Subject initials (INITIALS)

**CERT_ISSUER_COMMON_NAME**
Issuer common name (CN)
CERT_ISSUER_COUNTRY
Issuer country (C)

CERT_ISSUER_DN_DER
Issuer distinguished name in binary ASN.1 DER-encoded format

CERT_ISSUER_DN_PRINTABLE
Issuer distinguished name as a printable character string

These DN attribute names are recognized by the System SSL runtime.
- C - Country
- CN - Common name
- DC - Domain component
- DNQUALIFIER - Distinguished name qualifier
- EMAIL - E-mail address
- GENERATIONQUALIFIER - Generation qualifier
- GIVENNAME - Given name
- INITIALS - Initials
- L - Locality
- MAIL - RFC822 style address
- NAME - Name
- O - Organization name
- OU - Organizational unit name
- PC - Postal code
- SERIALNUMBER - Serial number
- SN - Surname
- ST - State or province
- STREET - Street
- T - Title

CERT_ISSUER_DNQUALIFIER
Issuer distinguished name qualifier (DNQUALIFIER)

CERT_ISSUER_DOMAIN_COMPONENT
Issuer domain component (DC)

CERT_ISSUER_EMAIL
Issuer e-mail address (EMAIL)

CERT_ISSUER_GENERATIONQUALIFIER
Issuer generation qualifier (GENERATIONQUALIFIER)

CERT_ISSUER_GIVENNAME
Issuer given name (GIVENNAME)

CERT_ISSUER_INITIALS
Issuer initials (INITIALS)

CERT_ISSUER_LOCALITY
Issuer locality (L)

CERT_ISSUER_MAIL
Issuer RFC822 style address (MAIL)

CERT_ISSUER_NAME
Issuer name (NAME)
gsk_get_cert_by_label()

CERT_ISSUER_ORG
Issuer organization (O)

CERT_ISSUER_ORG_UNIT
Issuer organizational unit (OU)

CERT_ISSUER_POSTAL_CODE
Issuer postal code (PC)

CERT_ISSUER_SERIALNUMBER
Issuer serial number (SERIALNUMBER)

CERT_ISSUER_STATE_OR_PROVINCE
Issuer state or province (ST)

CERT_ISSUER_STREET
Issuer street (STREET)

CERT_ISSUER_SURNAME
Issuer surname (SN)

CERT_ISSUER_TITLE
Issuer title (T)

CERT_LOCALITY
Subject locality (L)

CERT_MAIL
Subject RFC822 style address (MAIL)

CERT_NAME
Subject name (NAME)

CERT_ORG
Subject organization (O)

CERT_ORG_UNIT
Subject organizational unit (OU)

CERT_POSTAL_CODE
Subject postal code (PC)

CERT_SERIAL_NUMBER
Certificate serial number

CERT_SERIALNUMBER
Subject serial number (SERIALNUMBER)

CERT_STATE_OR_PROVINCE
Subject state or province (ST)

CERT_STREET
Subject street (STREET)

CERT_SURNAME
Subject surname (SN)

CERT_TITLE
Subject title (T)

The CERT_BODY_DER, CERT_BODY_BASE64, CERT_DN_DER, and CERT_ISSUER_DN_DER elements are not null-terminated and the ‘cert_data_l’ field must be used to get the element length. All of the other elements are null-terminated character strings and the ‘cert_data_l’ field is the length of the string excluding the string delimiter.
gsk_get_cert_by_label()

Related Topics

gsk_environment_init()

gsk_secure_socket_init()
gsk_get_cipherSuites()

Returns the available SSL cipher suites.

Format

```c
#include <gskssl.h>

void gsk_get_cipherSuites ( gsk_cipherSuites * cipherSuites )
```

Parameters

cipherSuites

Returns the runtime version, release, security level, and cipher suites.

Usage

The `gsk_get_cipherSuites()` routine returns the System SSL runtime version, release, security level, and available cipher suites. The current System SSL runtime is Version 3 Release 21. The cipher suites are static null-terminated character strings which must not be modified or freed by the application.

If executing in FIPS mode, the cipher suites are those that meet FIPS 140-2 criteria. For more information about the FIPS cipher suites, see `gsk_environment_open()` on page 70.
gsk_get_ssl_vector()

Obtain the address of the Secure Socket Layer function vector.

Format

```c
#include <gskssl.h>

void gsk_get_ssl_vector (    gsk_uint32 * function_mask,
    gsk_ssl_vector ** function_vector)
```

Parameters

- `function_mask`
  Returns a bit mask indicating the Secure Socket Layer level.

- `function_vector`
  Returns the address of the Secure Socket Layer function vector.

Usage

The Secure Socket Layer (SSL) functions can be called using either static binding or runtime binding. Static binding is performed when the application is compiled while runtime binding is performed when the application is run.

In order to use static binding, the SSL sidefile is specified as input to the binder. This causes all SSL functions to be resolved at bind time and will cause the SSL DLL to be implicitly loaded when the application is run.

In order to use runtime binding, the SSL DLL must be explicitly loaded by the application and the SSL functions must be called using indirect addresses. The `gsk_get_ssl_vector()` routine allows an application to obtain the address of the SSL function vector containing an entry for each SSL API routine. This eliminates the need for the application to build the function vector through repeated calls to the dllqueryfn() routine.

The function mask indicates the capabilities of the SSL DLL. These values have been defined:

- **GSKSSL_API_LVL1**
  SSL functions provided as part of z/OS Version 1 Release 6 and later are available.

- **GSKSSL_API_LVL2**
  SSL functions provided as part of z/OS Version 1 Release 11 are available.


**gsk_get_update()**

Checks for a key database, SAF key ring or z/OS PKCS #11 token update.

**Format**

```c
#include <gskssl.h>

gsk_status gsk_get_update (gsk_handle env_handle, long *update_flag)
```

**Parameters**

- **env_handle**
  Specifies the SSL environment handle returned by the `gsk_environment_open()` routine.

- **update_flag**
  Returns 1 if the key database, SAF key ring or z/OS PKCS #11 token has been updated or 0 if it has not been updated.

**Results**

The function return value will be 0 (`GSK_OK`) if no error is detected. Otherwise, it will be one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

- **[GSK_INVALID_HANDLE]**
  The environment handle is not valid.

- **[GSK_INVALID_STATE]**
  The environment is not in the initialized state.

- **[GSK_KEYRING_OPEN_ERROR]**
  The key ring or token cannot be accessed.

**Usage**

The `gsk_get_update()` routine tests if the key database, SAF key ring or z/OS PKCS #11 token associated with the SSL environment has been updated since the last time that `gsk_get_update()` was called or since the environment was initialized if `gsk_get_update()` has not been called yet. If an update has occurred, the application can close the current environment and then create a new environment to pick up the updates.

**Related Topics**

- `gsk_environment_open()`
gsk_list_free()

Releases storage allocated for a list.

Format

```
#include <gskssl.h>

void gsk_list_free (gsk_list * list)
```

Parameters

`list` Specifies the list to be released.

Usage

The `gsk_list_free()` routine releases storage allocated for a list. This includes the gsk_list structure itself and all gsk_list structures anchored by the structure passed on the function call.

Related Topics

`gsk_attribute_get_data()`
gsk_secure_socket_close()

Closes a secure socket connection.

Format

```c
#include <gskssl.h>

gsk_status gsk_secure_socket_close (gsk_handle * soc_handle)
```

Parameters

`soc_handle`

Specifies the connection handle returned by the `gsk_secure_socket_open()` routine. The connection handle will be set to NULL upon completion.

Results

The function return value will be 0 (`GSK_OK`) if no error is detected. Otherwise, it will be one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

- `[GSK_CONNECTION_ACTIVE]`
  The connection has an active read or write request.
- `[GSK_INVALID_HANDLE]`
  The connection handle is not valid.
- `[GSK_WOULD_BLOCK_WRITE]`
  An attempt to write pending data failed with EWOULDBLOCK.

Usage

The `gsk_secure_socket_close()` routine will close a secure socket connection created by the `gsk_secure_socket_open()` routine. The socket itself is not closed (the application is responsible for closing the socket). The connection can no longer be used for secure communications after calling the `gsk_secure_socket_close()` routine.

The `gsk_secure_socket_close()` routine can return `GSK_WOULD_BLOCK_WRITE` if the socket is in non-blocking mode and there is pending write data. The connection is not closed in this case and the application should call `gsk_secure_socket_close()` again when the socket is ready to accept a write request.

Be sure `gsk_secure_socket_shutdown` call is issued before a `gsk_secure_socket_close` call.

Related Topics

- `gsk_secure_socket_open()`
- `gsk_secure_socket_init()`
- `gsk_secure_socket_read()`
- `gsk_secure_socket_write()`
gsk_secure_socket_init()

Initializes a secure socket connection.

Format

```c
#include <gskssl.h>

gsk_status gsk_secure_socket_init(
    gsk_handle    soc_handle)
```

Parameters

```
soc_handle

  Specifies the connection handle returned by the gsk_secure_socket_open() routine.
```

Results

The function return value will be 0 (GSK_OK) if no error is detected. Otherwise, it will be one of the return codes listed in the gskssl.h include file. These are some possible errors:

- **[GSK_ERR_BAD_CERT]**
  Certificate is not valid.
- **[GSK_ERR_BAD_DATE]**
  Certificate is not valid yet or is expired.
- **[GSK_ERR_BAD_KEYFILE_LABEL]**
  The specified key is not found in the key database or the key is not trusted.
- **[GSK_ERR_BAD_MAC]**
  Message verification failed.
- **[GSK_ERR_BAD_MESSAGE]**
  Incorrectly-formatted message received from peer application.
- **[GSK_ERR_BAD_MSG_LEN]**
  Incorrectly-formatted TLS extension data contained within message received from peer application.
- **[GSK_ERR_BAD_PEER]**
  Peer application has violated the SSL protocol.
- **[GSK_ERR_CERT_VALIDATION]**
  Certificate validation error.
- **[GSK_ERR_CERTIFICATE_REVOKED]**
  Peer certificate is revoked.
- **[GSK_ERR_CRYPTO]**
  Cryptographic error detected.
- **[GSK_ERR_INCOMPATIABLE_KEY]**
  Certificate key is not compatible with cipher suite.
- **[GSK_ERR_INVALID_FRAGMENT_LENGTH]**
  An unsupported fragment length was received.
- **[GSK_ERR_IO]**
  I/O error communicating with peer application.
- **[GSK_ERR_LDAP]**
  An LDAP error is detected.
The LDAP server is not available.

Multiple keys are marked as the default.

Multiple certificates exist for label.

No cipher specifications.

Certificate does not contain a private key or the private key is unusable.

A self-signed certificate cannot be validated.

Socket connection closed by peer application.

Error encountered when generating random bytes.

A certification authority certificate is missing.

The requested server name is not recognized.

The certificate type is not supported by System SSL.

A required TLS extension has been rejected.

An unrequested TLS Extension has been encountered.

Insufficient storage is available.

The connection handle is not valid.

The connection is not in the open state or a previous initialization request has failed.

Unable to generate temporary RSA public/private key pair.

An attempt to read a handshake message failed with EWOULDBLOCK.

An attempt to write a handshake message failed with EWOULDBLOCK.

The `gsk_secure_socket_init()` routine initializes a secure socket connection created by the `gsk_secure_socket_open()` routine. After the connection has been initialized, it can be used for secure data transmission using the `gsk_secure_socket_read()` and `gsk_secure_socket_write()` routines. The `gsk_secure_socket_close()` routine should be called to close the connection when it is no longer needed. The `gsk_secure_socket_close()` routine should also be called if an error is returned by the `gsk_secure_socket_init()` routine.
gsk_secure_socket_init()

Before calling the `gsk_secure_socket_init()` routine, the application must create a connected socket and store the socket descriptor in the SSL connection by calling the `gsk_attribute_set_numeric_value()` routine. For a client, this means calling the `socket()` and `connect()` routines. For a server, this means calling the `socket()`, `listen()`, and `accept()` routines. However, SSL does not require the use of TCP/IP for the communications layer. The socket descriptor can be any integer value which is meaningful to the application. The application must provide its own socket routines if it is not using TCP/IP by calling the `gsk_attribute_set_callback()` routine.

An SSL handshake is performed as part of the processing of the `gsk_secure_socket_init()` routine. This establishes the server identity and optionally the client identity. It also negotiates the cryptographic parameters to be used for the connection. The client and server will attempt to use the highest available protocol version as determined by the intersection of the enabled protocol versions for the client and the server and the compatible ciphers. Thus:
- TLS V1.1 will be used if it is enabled on both the client and the server
- If TLS V1.1 cannot be used and TLS V1.0 is enabled, negotiations will drop back to TLS V1.0
- If TLS V1.0 cannot be used and SSL V3 is enabled, negotiations will drop back to SSL V3
- If SSL V3 cannot be used and SSL V2 is enabled, negotiations will drop back to SSL V2

Notes:
1. SSL V2 is not as secure as SSL V3 or TLS and should be disabled whenever possible to avoid attacks which force the client and server to drop back to SSL V2 even though they are capable of using SSL V3, TLS V1.0 or TLS V1.1.
2. When TLS extensions are defined for a client and any of the TLS protocols are enabled for the connection, SSL V2 will not be negotiated even if it is enabled.

The client sends a list of ciphers it supports during the SSL handshake. The server application uses this list, and the defined ciphers supported by the server, to determine the cipher to be used during the SSL handshake. If the client is operating in FIPS mode, then the list provided only contains FIPS ciphers. A server executing in FIPS mode will only use FIPS ciphers. The cipher selection is done by looking through the server’s cipher list for a match in the client’s list. The first matching cipher is used. The TLS V1.1 protocol will not use export (40-bit) ciphers. If TLS V1.1 is the intended protocol and only export ciphers are available, the connection will fail with GSK_ERR_NO_CIPHERS.

The server certificate can use either RSA or DSA as the public/private key algorithm. In FIPS mode, the RSA or DSA key size must be at least 1024 bits. An RSA certificate can be used with an RSA, fixed Diffie-Hellman or ephemeral Diffie-Hellman key exchange. A DSA certificate can be used with either a fixed or ephemeral Diffie-Hellman key exchange. In FIPS mode, Diffie-Hellman key size must be 2048 bits. If the server’s certificate contains a key usage extension during the SSL handshake, it must allow key usage as follows:
- RSA certificates using export restricted ciphers (40-bit RC4 encryption and 40-bit RC2 encryption) with a public key size greater than 512 bits must allow digital signature. If operating in FIPS mode, export restricted ciphers cannot be selected.
- RSA or DSA certificates using fixed Diffie-Hellman key exchange must allow key agreement.
- Other RSA certificates must allow key encipherment.
- DSA certificates using ephemeral Diffie-Hellman key exchange must allow digital signature.

System SSL does not honor Verisign Global Server ID certificates. When specified, System SSL will utilize these certificates as any other certificate when determining the encryption cipher to be used for the SSL session.

The client certificate must support digital signatures. This means the certificate key usage extension (if any) must allow digital signature. The key algorithm can be either the RSA encryption algorithm or the Digital Signature Standard algorithm (DSA).
The SSL server always provides its certificate to the SSL client as part of the handshake. The client always performs server authentication using the certificate provided by the server. Depending upon the server handshake type, the server may ask the client to provide its certificate. The key label stored in the connection is used to retrieve the certificate from the key database, key ring or token. The default key will be used if no label is set. The key record must contain both an X.509 certificate and a private key. Refer to the gsk_validate_certificate_mode() routine for a description of the steps which are performed during certificate validation.

The gsk_secure_socket_init() routine can return GSK_WOULD_BLOCK_READ or GSK_WOULD_BLOCK_WRITE if the socket is in non-blocking mode. The connection is not initialized in this case and the application must call gsk_secure_socket_init() again when the socket is ready to accept a read request (GSK_WOULD_BLOCK_READ) or a write request (GSK_WOULD_BLOCK_WRITE). The application must provide its own callback routine for io_setsocketoptions() in order to have the SSL handshake processed in non-blocking mode (the default io_setsocketoptions() routine will place the socket into blocking mode during the handshake processing).

Be sure gsk_secure_socket_shutdown call is issued before a gsk_secure_socket_close call.

Related Topics

- gsk_environment_init()
- gsk_secure_socket_write()
- gsk_secure_socket_read()
- gsk_secure_socket_misc()
- gsk_secure_socket_close()
gsk_secure_socket_misc()

Performs miscellaneous secure connection functions.

Format

```c
#include <gskssl.h>

gsk_status gsk_secure_socket_misc (gsk_handle soc_handle, GSK_MISC_ID misc_id)
```

Parameters

- `soc_handle`
  Specifies the connection handle returned by the `gsk_secure_socket_open()` routine.

- `misc_id`
  Miscellaneous function identifier.

Results

The function return value will be 0 (GSK_OK) if no error is detected. Otherwise, it will be one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

- [GSK_ERR_CONNECTION_CLOSED]
  A close notification alert has been sent for the connection.

- [GSK_ERR_IO]
  I/O error communicating with peer application.

- [GSK_ERR_NOT_SSLV3]
  The session is not using the SSL V3, TLS V1.0 or TLS V1.1 protocol.

- [GSK_ERR_SOCKET_CLOSED]
  Socket connection closed by peer application.

- [GSK_INVALID_HANDLE]
  The connection handle is not valid.

- [GSK_INVALID_STATE]
  The connection is not in the initialized state.

- [GSK_MISC_INVALID_ID]
  The miscellaneous identifier is not valid.

Usage

The `gsk_secure_socket_misc()` routine performs miscellaneous function for an initialized secure connection.

These miscellaneous functions are provided:

- **GSK_RESET_CIPHER**
  This function generates new session keys for the connection. A full SSL handshake will be performed if the session has expired or has been reset by the GSK_RESET_SESSION function. Otherwise a short SSL handshake will be performed. The GSK_RESET_CIPHER function can be performed only for a session using the SSL V3, TLS V1.0 or TLS V1.1 protocol. The GSK_RESET_CIPHER function initiates the SSL handshake but does not wait for it to complete. Any pending handshake messages will be processed when the `gsk_secure_socket_read()` routine is called to process incoming data.
GSK_RESET_SESSION
This function resets the session associated with the connection. A full SSL handshake will be performed for the next connection using the session. The current connection is not affected unless the GSK_RESET_CIPHER function is performed after the GSK_RESET_SESSION function has completed.

Related Topics
- gsk_secure_socket_open()
- gsk_secure_socket_read()
- gsk_secure_socket_write()
gsk_secure_socket_open()

Creates a secure socket connection.

Format

```c
#include <gskssl.h>

gsk_status gsk_secure_socket_open ( gsk_handle env_handle,
            gsk_handle * soc_handle)
```

Parameters

- `env_handle` specifies the SSL environment handle returned by the `gsk_environment_open()` routine.
- `soc_handle` returns the handle for the secure connection. The application should call the `gsk_secure_socket_close()` routine to release the connection when it is no longer needed.

Results

The function return value will be 0 (GSK_OK) if no error is detected. Otherwise, it will be one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

- [GSK_INSUFFICIENT_STORAGE] Insufficient storage is available.
- [GSK_INVALID_HANDLE] The environment handle is not valid.
- [GSK_INVALID_STATE] The environment is not in the initialized state.

Usage

The `gsk_secure_socket_open()` routine creates a secure socket connection. The connection will be initialized with values obtained from the SSL environment. These values can be changed by the application using the appropriate `gsk_attribute_set_*()` routines. The `gsk_secure_socket_init()` routine should then be called to initialize the connection. This connection can then be used to send and receive data with the remote partner.

Related Topics

- `gsk_secure_socket_close()
- `gsk_secure_socket_init()`
gsk_secure_socket_read()

Reads data using a secure socket connection.

Format

```c
#include <gskssl.h>
gsk_status gsk_secure_socket_read (gsk_handle soc_handle, char * buffer, int size, int * length)
```

Parameters

- **soc_handle**: Specifies the connection handle returned by the `gsk_secure_socket_open()` routine.
- **buffer**: Specifies the buffer to receive the data read from the secure socket connection. The maximum amount of data returned by `gsk_secure_socket_read()` is 16384 (16K) bytes. If the SSL V2 protocol is used, then the maximum length is 16384 minus the length of the SSL protocol headers.
- **size**: Specifies the size of the supplied buffer.
- **length**: Returns the length of the data read into the supplied buffer.

Results

The function return value will be 0 (GSK_OK) if no error is detected. Otherwise, it will be one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

**[GSK_CONNECTION_ACTIVE]**
A read request is already active for the connection.

**[GSK_ERR_BAD_MAC]**
Message verification failed.

**[GSK_ERR_BAD_MESSAGE]**
Incorrectly-formatted message received from peer application.

**[GSK_ERR_BAD_PEER]**
Peer application has violated the SSL protocol.

**[GSK_ERR_CONNECTION_CLOSED]**
Close notification received from peer application.

**[GSK_ERR_CRYPTO]**
Cryptographic error detected.

**[GSK_ERR_IO]**
I/O error communicating with peer application.

**[GSK_ERR_SOCKET_CLOSED]**
Socket connection closed by peer application.

**[GSK_INSUFFICIENT_STORAGE]**
Insufficient storage is available.

**[GSK_INVALID_BUFFER_SIZE]**
The buffer address or buffer size is not valid.

**[GSK_INVALID_HANDLE]**
The connection handle is not valid.
**Usage**

The `gsk_secure_socket_read()` routine reads data from a secure socket connection and returns it in the application buffer. SSL is a record-based protocol and a single call will never return more than a single SSL record. The maximum amount of data returned by `gsk_secure_socket_read()` is 16384 (16K) bytes. If the SSL V2 protocol is used, then the maximum length is 16384 minus the length of the SSL protocol headers. The application can read an entire SSL record in a single call by supplying a buffer large enough to contain the record. Otherwise, multiple calls will be required to retrieve the entire SSL record.

SSL supports multiple threads but only one thread at a time can call the `gsk_secure_socket_read()` routine for a given connection handle. Multiple concurrent threads can call `gsk_secure_socket_read()` as long as each thread has its own connection handle.

SSL supports sockets in blocking mode and in non-blocking mode. When a socket is in non-blocking mode and a complete SSL record is not available, `gsk_secure_socket_read()` will return with GSK_WOULD_BLOCK. No data will be returned in the application buffer when GSK_WOULD_BLOCK is returned. The application should call `gsk_secure_socket_read()` again when there is data available to be read from the socket.

The peer application can initiate an SSL handshake sequence after the connection is established. If this is done and the socket is in non-blocking mode, it is possible for `gsk_secure_socket_read()` to return with GSK_WOULD_BLOCK_WRITE. This indicates that an SSL handshake is in progress and the application should call `gsk_secure_socket_read()` again when data can be written to the socket. No data will be returned in the application buffer when GSK_WOULD_BLOCK_WRITE is returned.

The application should not read data directly from the socket since this can cause SSL protocol errors if the application inadvertently reads part of an SSL record. If the application must read data from the socket, it is responsible for synchronizing this activity with the peer application so that no SSL records are sent while the application is performing its own read operations.

**Related Topics**

- `gsk_secure_socket_write()
- `gsk_secure_socket_init()`
gsk_secure_socket_shutdowm()

Shuts down a secure socket connection.

Format

```c
#include <gskssl.h>

gsk_status gsk_secure_socket_shutdown (gsk_handle soc_handle);
```

Parameters

`soc_handle`

Specifies the connection handle returned by the `gsk_secure_socket_open()` routine.

Results

The function return value will be 0 (GSK_OK) if no error is detected. Otherwise, it will be one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

- **GSK_CONNECTION_ACTIVE**
  
The connection has an active write request.

- **GSK_ERR_CONNECTION_CLOSED**
  
The close notification alert has already been sent.

- **GSK_ERR_IO**
  
I/O error communicating with peer application.

- **GSK_ERR_NOT_SSLV3**
  
The session is not using the SSL V3, TLS V1.0 or TLS V1.1 protocol.

- **GSK_INVALID_HANDLE**
  
The connection handle is not valid.

- **GSK_INVALID_STATE**
  
The connection is not in the initialized state.

- **GSK_WOULD_BLOCK_WRITE**
  
An attempt to write pending data failed with EWOULDBLOCK.

Usage

The `gsk_secure_socket_shutdown()` routine will send a close notification alert to the peer application. Any subsequent calls to the `gsk_secure_socket_write()` routine will return GSK_ERR_CONNECTION_CLOSED. The `gsk_secure_socket_shutdown()` routine cannot be used with the SSL V2 protocol.

The application should call `gsk_secure_socket_shutdown()` before calling `gsk_secure_socket_close()` in order to comply with the SSL V3, TLS V1.0 or TLS V1.1 specifications, which require that a close notification alert be sent before closing the transport connection.

For a 1-step shutdown, the application should call the `gsk_secure_socket_shutdown()` routine and then call the `gsk_secure_socket_close()` routine. This sends the close notification alert and then closes the secure socket connection. The application does not wait for acknowledgement from the peer application to the close notification.
For a 2-step shutdown, the application should call the `gsk_secure_socket_shutdown()` routine to send the close notification alert and then call the `gsk_secure_socket_read()` routine to process any pending data sent by the peer application. The SSL runtime on the peer system will send a close notification alert when it receives the close notification alert from the local system. The `gsk_secure_socket_read()` routine will return GSK_ERR_CONNECTION_CLOSED when it receives this close notification. The application should then call the `gsk_secure_socket_close()` routine to close the secure socket connection.

Related Topics

- `gsk_secure_socket_close()
- `gsk_secure_socket_open()
- `gsk_secure_socket_read()
- `gsk_secure_socket_write()`
gsk_secure_socket_write()

Writes data using a secure socket connection.

Format

```c
#include <gskssl.h>

gsk_status gsk_secure_socket_write ( gsk_handle soc_handle, char * buffer, int size, int * length )
```

Parameters

- `soc_handle` Specifies the connection handle returned by the `gsk_secure_socket_open()` routine.
- `buffer` Specifies the buffer containing the data to write to the secure socket connection.
- `size` Specifies the amount to write.
- `length` Returns the length of the data written.

Results

The function return value will be 0 (GSK_OK) if no error is detected. Otherwise, it will be one of the return codes listed in the gskssl.h include file. These are some possible errors:

- **[GSK_CONNECTION_ACTIVE]**
  A write request is already active for the connection.

- **[GSK_ERR_CONNECTION_CLOSED]**
  A close notification alert has been sent for the connection.

- **[GSK_ERR_CRYPTO]**
  Cryptographic error detected.

- **[GSK_ERR_IO]**
  I/O error communicating with peer application.

- **[GSK_ERR_SOCKET_CLOSED]**
  Socket connection closed by peer application.

- **[GSK_INSUFFICIENT_STORAGE]**
  Insufficient storage is available.

- **[GSK_INVALID_BUFFER_SIZE]**
  The buffer address or buffer size is not valid.

- **[GSK_INVALID_HANDLE]**
  The connection handle is not valid.

- **[GSK_INVALID_STATE]**
  The connection is not in the initialized state.

- **[GSK_WOULD_BLOCK]**
  The SSL record cannot be written to the socket due to an EWOULDBLOCK condition.
gsk_secure_socket_write()

Usage
The gsk_secure_socket_write() routine writes data to a secure socket connection. SSL is a record-based protocol with a maximum record length of 16384 bytes. If the SSL V2 protocol is used, then the maximum length is 16384 minus the length of the SSL protocol headers. Application data larger than the size of an SSL record will be sent using multiple records.

SSL supports multiple threads but only one thread at a time can call the gsk_secure_socket_write() routine for a given connection handle. Multiple concurrent threads can call gsk_secure_socket_write() as long as each thread has its own connection handle.

SSL supports sockets in blocking mode and in non-blocking mode. When a socket is in non-blocking mode and the SSL record cannot be written to the socket, gsk_secure_socket_write() will return with GSK_WOULD_BLOCK. The application must call gsk_secure_socket_write() again when the socket is ready to accept more data, specifying the same buffer address and buffer size as the original request. A new write request must not be initiated until the pending write request has been completed as indicated by a return value of 0.

The application should not write data directly to the socket since this can cause SSL protocol errors if the application inadvertently intermixes its data with SSL protocol data. If the application must write data to the socket, it is responsible for synchronizing this activity with the peer application so that application data is not intermixed with SSL data.

To notify your partner application that you are done sending data on the secure connection, a call to gsk_secure_socket_shutdown should be issued before the gsk_secure_socket_close call.

Related Topics
- gsk_secure_socket_read()
- gsk_secure_socket_init()
gsk_strerror()

Return a text string for an SSL error code

Format

```c
#include <gskssl.h>

const char * gsk_strerror (  
gsk_status       error_code
);
```

Parameters

`error_code`
Specifies an error code returned by a Secure Sockets layer (SSL) routine or by a Certificate Management Services (CMS) routine.

Results

The function return value is the address of the text string. The return value will always be a valid text string address even when the error code is not recognized (the return value will be the string "N/A" in this case).

Usage

The `gsk_strerror()` routine returns a text string describing an error code returned by an SSL (Secure Sockets layer) or CMS (Certificate Management Services) routine. The `gsk_strerror()` routine cannot be used to return a text string for an error code returned by one of the deprecated SSL routines. The text string must not be modified or released by the application program.
gsk_strerror()
Chapter 8. Certificate Management Services (CMS) API Reference

This topic describes the Certificate Management Services (CMS) APIs. These APIs can be used to create/manage your own key database files in a similar function to the SSL gskkyman utility, use certificates stored in the key database file or key ring for purposes other than SSL, and basic PKCS #7 message support.

- System SSL supports X.509 certificates (V1, V2 or V3) and X.509 V2 Certificate Revocation Lists as described in RFC3280 (Internet X.509 Public Key Infrastructure) and RFC2459 (Internet X.509 Public Key Infrastructure). RFC3280 obsoletes RFC2459.

This is a list of the Certificate Management Services (CMS) APIs:

- `gsk_add_record()` (see page 105)
- `gsk_change_database_password()` (see page 107)
- `gsk_change_database_record_length()` (see page 109)
- `gsk_close_database()` (see page 110)
- `gsk_close_directory()` (see page 111)
- `gsk_construct_certificate()` (see page 112)
- `gsk_construct_private_key_rsa()` (see page 115)
- `gsk_construct_renewal_request()` (see page 117)
- `gsk_construct_self_signed_certificate()` (see page 119)
- `gsk_construct_signed_certificate()` (see page 121)
- `gsk_copy_attributes_signers()` (see page 124)
- `gsk_copy_buffer()` (see page 125)
- `gsk_copy_certificate()` (see page 126)
- `gsk_copy_certificate_extension()` (see page 127)
- `gsk_copy_certification_request()` (see page 128)
- `gsk_copy_content_info()` (see page 129)
- `gsk_copy_crl()` (see page 130)
- `gsk_copy_name()` (see page 131)
- `gsk_copy_private_key_info()` (see page 132)
- `gsk_copy_public_key_info()` (see page 133)
- `gsk_copy_record()` (see page 134)
- `gsk_create_certification_request()` (see page 135)
- `gsk_create_database()` (see page 138)
- `gsk_create_database_renewal_request()` (see page 140)
- `gsk_create_database_signed_certificate()` (see page 143)
- `gsk_create_renewal_request()` (see page 147)
- `gsk_create_self_signed_certificate()` (see page 149)
- `gsk_create_signed_certificate()` (see page 152)
- `gsk_create_signed_certificate_record()` (see page 155)
- `gsk_create_signed_certificate_set()` (see page 158)
- `gsk_create_signed_crl()` (see page 162)
- `gsk_create_signed_crl_record()` (see page 164)
- `gsk_decode_base64()` (see page 167)
- `gsk_decode_certificate()` (see page 168)
- `gsk_decode_certificate_extension()` (see page 169)
- `gsk_decode_certification_request()` (see page 171)
- `gsk_decode_crl()` (see page 172)
- `gsk_decode_import_certificate()` (see page 173)
- `gsk_decode_import_key()` (see page 174)
- `gsk_decode_name()` (see page 176)
- `gsk_decode_private_key()` (see page 177)
- `gsk_decode_public_key()` (see page 178)
- `gsk_delete_record()` (see page 179)
- `gsk_dn_to_name()` (see page 180)
- `gsk_encode_base64()` (see page 183)
- `gsk_encode_certificate_extension()` (see page 184)
- `gsk_encode_export_certificate()` (see page 186)
- `gsk_encode_export_key()` (see page 188)
- `gsk_encode_export_request()` (see page 190)
- `gsk_encode_name()` (see page 191)
- `gsk_encode_private_key()` (see page 192)
- `gsk_encode_public_key()` (see page 193)
- `gsk_encode_signature()` (see page 194)
- `gsk_export_certificate()` (see page 195)
- `gsk_export_certification_request()` (see page 197)
- `gsk_export_key()` (see page 199)
- `gsk_factor_private_key_rsa()` (see page 202)
- `gsk_factor_public_key_rsa()` (see page 203)
- `gsk_fips_state_query()` (see page 204)
- `gsk_fips_state_set()` (see page 205)
- `gsk_free_attributes_signers()` (see page 206)
- `gsk_free_buffer()` (see page 207)
- `gsk_free_certificate()` (see page 208)
- `gsk_free_certificates()` (see page 209)
- `gsk_free_certificate_extension()` (see page 210)
- `gsk_free_certification_request()` (see page 211)
- `gsk_free_content_info()` (see page 212)
- `gsk_free_crl()` (see page 213)
- `gsk_free_crls()` (see page 214)
- `gsk_free_decoded_extension()` (see page 215)
- `gsk_free_name()` (see page 216)
- `gsk_free_private_key_info()` (see page 217)
- `gsk_free_public_key_info()` (see page 218)
- `gsk_free_record()` (see page 219)
- `gsk_free_records()` (see page 220)
- `gsk_free_string()` (see page 221)
- `gsk_free_strings()` (see page 222)
- `gsk_generate_key_agreement_pair()` (see page 223)
Chapter 8. Certificate Management Services (CMS) API Reference

- `gsk_generate_key_pair()` (see page 224)
- `gsk_generate_key_parameters()` (see page 226)
- `gsk_generate_random_bytes()` (see page 228)
- `gsk_generate_secret()` (see page 229)
- `gsk_get_certificate_algorithms()` (see page 230)
- `gsk_get_cms_vector()` (see page 231)
- `gsk_get_default_key()` (see page 232)
- `gsk_get_default_label()` (see page 233)
- `gsk_get_directory_certificates()` (see page 234)
- `gsk_get_directory_crls()` (see page 235)
- `gsk_get_directory_enum()` (see page 237)
- `gsk_get_record_by_id()` (see page 238)
- `gsk_get_record_by_index()` (see page 239)
- `gsk_get_record_by_label()` (see page 240)
- `gsk_get_record_by_subject()` (see page 241)
- `gsk_get_record_labels()` (see page 242)
- `gsk_get_update_code()` (see page 243)
- `gsk_import_certificate()` (see page 244)
- `gsk_import_key()` (see page 246)
- `gsk_make_content_msg()` (see page 249)
- `gsk_make_data_content()` (see page 250)
- `gsk_make_data_msg()` (see page 251)
- `gsk_make_encrypted_data_content()` (see page 252)
- `gsk_make_encrypted_data_msg()` (see page 254)
- `gsk_make_enveloped_data_content()` (see page 256)
- `gsk_make_enveloped_data_content_extended()` (see page 258)
- `gsk_make_enveloped_data_msg()` (see page 260)
- `gsk_make_enveloped_data_msg_extended()` (see page 262)
- `gsk_make_signed_data_content()` (see page 264)
- `gsk_make_signed_data_content_extended()` (see page 266)
- `gsk_make_signed_data_msg()` (see page 269)
- `gsk_make_signed_data_msg_extended()` (see page 271)
- `gsk_make_wrapped_content()` (see page 274)
- `gsk_mktime()` (see page 275)
- `gsk_name_compare()` (see page 276)
- `gsk_name_to_dn()` (see page 277)
- `gsk_open_database()` (see page 279)
- `gsk_open_database_using_stash_file()` (see page 281)
- `gsk_open_directory()` (see page 283)
- `gsk_open_keyring()` (see page 284)
- `gsk_perform_kat()` (see page 286)
- `gsk_query_crypto_level()` (see page 287)
- `gsk_query_database_label()` (see page 288)
- `gsk_query_database_record_length()` (see page 289)
- `gsk_rdtme()` (see page 290)
• `gsk_read_content_msg()` (see page 291)
• `gsk_read_data_content()` (see page 292)
• `gsk_read_data_msg()` (see page 293)
• `gsk_read_encrypted_data_content()` (see page 294)
• `gsk_read_encrypted_data_msg()` (see page 296)
• `gsk_read_enveloped_data_content()` (see page 298)
• `gsk_read_enveloped_data_content_extended()` (see page 300)
• `gsk_read_enveloped_data_msg()` (see page 302)
• `gsk_read_enveloped_data_msg_extended()` (see page 304)
• `gsk_read_signed_data_content()` (see page 306)
• `gsk_read_signed_data_content_extended()` (see page 308)
• `gsk_read_signed_data_msg()` (see page 311)
• `gsk_read_signed_data_msg_extended()` (see page 314)
• `gsk_read_wrapped_content()` (see page 317)
• `gsk_receive_certificate()` (see page 318)
• `gsk_replace_record()` (see page 319)
• `gsk_set_default_key()` (see page 321)
• `gsk_set_directory_enum()` (see page 322)
• `gsk_sign_certificate()` (see page 323)
• `gsk_sign_crl()` (see page 325)
• `gsk_sign_data()` (see page 327)
• `gsk_validate_certificate()` (see page 329)
• `gsk_validate_certificate_mode()` (see page 332)
• `gsk_validate_hostname()` (see page 336)
• `gsk_validate_server()` (see page 338)
• `gsk_verify_certificate_signature()` (see page 339)
• `gsk_verify_crl_signature()` (see page 341)
• `gsk_verify_data_signature()` (see page 343)
gsk_add_record()

Adds a record to a key or request database.

Format

```c
#include <gskcms.h>
gsk_status gsk_add_record (      
gsk_handle  db_handle,  
gskdb_record * record)
```

Parameters

`db_handle`

Specifies the database handle returned by the `gsk_create_database()` routine or the `gsk_open_database()` routine.

`record`

Specifies the database record.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_ALG_NOT_SUPPORTED]**
  The key algorithm or signature algorithm is not supported.

- **[CMSERR_BACKUP_EXISTS]**
  The backup file already exists.

- **[CMSERR_BAD_HANDLE]**
  The database handle is not valid.

- **[CMSERR_BAD_KEY_SIZE]**
  The key size is not valid.

- **[CMSERR_BAD_LABEL]**
  The record label is not valid.

- **[CMSERR_BAD_RNG_OUTPUT]**
  In FIPS mode, random bytes generation produced duplicate output.

- **[CMSERR_DUPLICATE_CERTIFICATE]**
  The database already contains the certificate.

- **[CMSERR_INCORRECT_DBTYPE]**
  The record type is not supported for the database type.

- **[CMSERR_IO_ERROR]**
  Unable to write record.

- **[CMSERR_LABEL_NOT_UNIQUE]**
  The record label is not unique.

- **[CMSERR_NO_MEMORY]**
  Insufficient storage is available.

- **[CMSERR_NO_PRIVATE_KEY]**
  No private key is provided for a record type that requires a private key.

- **[CMSERR_RECORD_TOO_BIG]**
  The record is larger than the database record length.
gsk_add_record()

[CMSERR_RECTYPE_NOT_VALID]
The record type is not valid.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage
The gsk_add_record() routine adds a record to a key or request database. The database must be open for update in order to add records. Unused and reserved fields in the gskdb_record structure must be initialized to zero. An error will be returned when adding a certificate to a key database if the database already contains the certificate. If the record has a private key, the encrypted private key will be generated from the private key supplied in the database record.

The recordType field identifies the database record type as follows:

gskdb_rectype_certificate
The record contains an X.509 certificate

gskdb_rectype_certKey
The record contains an X.509 certificate and private key

gskdb_rectype_keyPair
The record contains a PKCS #10 certification request and private key

The recordFlags field is a bit field with these values:

GSKDB_RECFLAG_TRUSTED
The certificate is trusted

GSKDB_RECFLAG_DEFAULT
This is the default key

A unique record identifier is assigned when the record is added to the database and will be returned to the application in the recordId field. If the record contains an X.509 certificate, the issuerRecordId field will be set to the record identifier of the certificate issuer.

The record label is used as a friendly name for the database entry and is in the local code page. It can be set to any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be set to an empty string.

If the record contains an X.509 certificate, the certificate will be validated and the record will not be added to the database if the validation check fails. If the database is a FIPS key database, then the certificate must use only FIPS algorithms and key sizes.

With the exception of the record label, all character strings are specified using UTF-8.

The database file is updated as part of the gsk_add_record() processing. A temporary database file is created using the same name as the database file with "_.new" appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.
gsk_change_database_password()

Changes the database password.

Format

```c
#include <gskcms.h>

gsk_status gsk_change_database_password (const char * filename, const char * old_password, const char * new_password, gsk_time pwd_expiration)
```

Parameters

*filename*

Specifies the database file name in the local code page. The length of the fully-qualified file name cannot exceed 251.

*old_password*

Specifies the current database password in the local code page. The user will be prompted to enter the password if NULL is specified for this parameter.

*new_password*

Specifies the new database password in the local code page. The user will be prompted to enter the password if NULL is specified for this parameter.

*pwd_expiration*

Specifies the new password expiration time as the number of seconds since the POSIX epoch. A value of 0 indicates the password does not expire.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_ACCESS_DENIED]**
  The file permissions do not allow access.

- **[CMSERR_BACKUP_EXISTS]**
  The backup file already exists.

- **[CMSERR_BAD_FILENAME]**
  The database file name is not valid.

- **[CMSERR_DB_CORRUPTED]**
  The database file is not valid.

- **[CMSERR_DB_FIPS_MODE_ONLY]**
  Key database can only be opened for update if running in FIPS mode.

- **[CMSERR_DB_LOCKED]**
  The database is open for update by another process.

- **[CMSERR_DB_NOT_FIPS]**
  Key database is not a FIPS mode database.

- **[CMSERR_FILE_NOT_FOUND]**
  The database file is not found.

- **[CMSERR_IO_CANCELED]**
  The user canceled the password prompt.
gsk_change_database_password()

[CMSERR_IO_ERROR]
   An input/output request failed.

[CMSERR_NO_MEMORY]
   Insufficient storage is available.

[CMSERR_OPEN_FAILED]
   Unable to open the database.

[CMSERR_PW_INCORRECT]
   The password is not correct.

Usage
The gsk_change_database_password() routine will change the password for the database and set a new password expiration time. gsk_mktime() can be used to convert a year/month/day time value to the number of seconds since the POSIX epoch.

A FIPS database password may only be changed while executing in FIPS mode. A non-FIPS database password can only be changed if not executing in FIPS mode.
gsk_change_database_record_length()

Changes the database record length.

Format

```c
#include <gskcms.h>

gsk_status gsk_change_record_length (  
gsk_handle  db_handle,  
gsk_size    record_length)
```

Parameters

- **db_handle**
  - Specifies the database handle returned by the `gsk_create_database()` routine or the `gsk_open_database()` routine.

- **record_length**
  - Specifies the new database record length. The default record length will be used if zero is specified for this parameter. All records in the database will have this length. The minimum record length is 2500. The default record length is 5000.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_BACKUP_EXISTS]**
  - The backup file already exists.

- **[CMSERR_BAD_HANDLE]**
  - The database handle is not valid.

- **[CMSERR_IO_ERROR]**
  - An input/output request failed.

- **[CMSERR_LENGTH_TOO_SMALL]**
  - The record length is less than the minimum value.

- **[CMSERR_NO_MEMORY]**
  - Insufficient storage is available.

- **[CMSERR_RECORD_TOO_BIG]**
  - A record in the database is larger than the new record length.

- **[CMSERR_UPDATE_NOT_ALLOWED]**
  - Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The `gsk_change_database_record_length()` routine will change the record length for the database. All records in the database have the same length and a database entry cannot span records. An error will be returned if the requested record length is smaller than the largest entry in the database.
gsk_close_database()

Closes a key or request database.

Format

#include <gskcms.h>

gsk_status gsk_close_database (       gsk_handle * db_handle)

Parameters

db_handle
Specifies the database handle returned by the gsk_create_database() routine or the
gsk_open_database() routine. The handle will be set to NULL upon successful completion.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes
listed in the gskcms.h include file. This is a possible error:

[CMSERR_BAD_HANDLE]
The database handle is not valid.

Usage

The gsk_close_database() routine will close a key or request database. The db_handle will not be valid
upon return from the gsk_close_database() routine.
gsk_close_directory()

Closes an LDAP directory.

Format

```c
#include <gskcms.h>

gsk_status gsk_close_directory ( gsk_handle * directory_handle )
```

Parameters

directory_handle

Specifies the directory handle returned by the `gsk_open_directory()` routine. The handle will be set to NULL upon successful completion.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. This is a possible error:

[CMSERR_BAD-handle]

The directory handle is not valid.

Usage

The `gsk_close_directory()` routine will close an LDAP directory opened by the `gsk_open_directory()` routine. The directory_handle will not be valid upon return from the `gsk_close_directory()` routine.
gsk_construct_certificate()

Constructs a signed certificate and returns it to the caller.

Format

```c
#include <gskcms.h>
gsk_status gsk_construct_certificate (  
    pkcs_cert_key * issuer_certificate,  
    x509_algorithm_type signature_algorithm,  
    const char * subject_name,  
    int num_days,  
    gsk_boolean ca_certificate,  
    x509_extensions * extensions,  
    x509_certificate * public_key,  
    x509_certificate * subject_certificate)
```

Parameters

`issuer_certificate`  
Specifies the issuing CA certificate with private key.

`signature_algorithm`  
Specifies the signature algorithm for the certificate.

`subject_name`  
Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

`num_days`  
Specifies the number of days for the certificate validity period as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified).

`ca_certificate`  
Specify TRUE if this is a certification authority certificate or FALSE if this is an end user certificate.

`extensions`  
Specifies the certificate extensions for the new certificate. Specify NULL for this parameter if no certificate extensions are supplied.

`public_key`  
Specifies the public key for the constructed certificate.

`subject_certificate`  
Contains the constructed certificate.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_ALG_NOT_SUPPORTED]**
  The signature algorithm is not valid.

- **[CMSERR_BAD_KEY_SIZE]**
  The key size is not valid.

- **[CMSERR_BAD_SUBJECT_NAME]**
  The subject name is not valid.
The `gsk_construct_certificate()` routine will construct an X.509 certificate as described in RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). The certificate will be signed using the certificate as supplied by the `issuer_certificate` parameter. A certification authority (CA) certificate will have basic constraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have basic constraints and key usage extensions as follows:

- An RSA key can be used for authentication, digital signature, and data encryption. An RSA key can be used for both CA certificates and end user certificates.
- A DSS key can be used for authentication and digital signature. A DSS key can be used for both CA certificates and end user certificates.
- A Diffie_Hellman key can be used for key agreement. A Diffie-Hellman key can be used only for end user certificates.

The new certificate is returned in the supplied x509_certificate structure.

These signature algorithms are supported:

- `x509_alg_md2WithRsaEncryption`
  
  RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

- `x509_alg_md5WithRsaEncryption`
  
  RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

- `x509_alg_sha1WithRsaEncryption`
  
  RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

- `x509_alg_sha224WithRsaEncryption`
  
  RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

- `x509_alg_sha256WithRsaEncryption`
  
  RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}
gsk_construct_certificate()

**x509_alg_sha384WithRsaEncryption**
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

**x509_alg_sha512WithRsaEncryption**
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

**x509_alg_dsaWithSha1**
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

A CA certificate will have SubjectKeyIdentifier, KeyUsage and BasicConstraints extensions while an end user certificate will have SubjectKeyIdentifier and KeyUsage extensions. An AuthorityKeyIdentifier extension will be created if the signing certificate has a SubjectKeyIdentifier extension. The application can supply additional extensions through the extensions parameter. An AuthorityKeyIdentifier, KeyUsage or BasicConstraints extension provided by the application will replace the default extension constructed for the certificate, however a SubjectKeyIdentifier extension provided by the application will be ignored.
gsk_construct_private_key_rsa()

Constructs an RSA private key from its component values.

Format

```
#include <gskcms.h>

#include <gskcms.h>
gsk_status gsk_construct_private_key_rsa (  
gsk_buffer * modulus,  
gsk_buffer * public_exponent,  
gsk_buffer * private_exponent,  
gsk_buffer * prime1,  
gsk_buffer * prime2,  
gsk_buffer * prime_exponent1,  
gsk_buffer * prime_exponent2,  
gsk_buffer * coefficient,  
pkcs_private_key_info * private_key)
```

Parameters

- `modulus`
  Specifies the modulus (n).

- `public_exponent`
  Specifies the public exponent (e).

- `private_exponent`
  Specifies the private exponent (d).

- `prime1`
  Specifies the 1st prime (p).

- `prime2`
  Specifies the 2nd prime (q).

- `prime_exponent1`
  Specifies the private exponent \(d\) modulo \(p-1\).

- `prime_exponent2`
  Specifies the private exponent \(d\) modulo \(q-1\).

- `coefficient`
  Specifies the CRT coefficient \(q^{-1} \mod p\).

- `private_key`
  Returns the private key

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- [ASN_ELEMENTS_MISSING]
  Required data element is missing.

Usage

The `gsk_construct_private_key_rsa()` function constructs `pkcs_private_key_info` from its RSA private key components. The `pkcs_private_key_info` structure's `x509_algorithm_identifier` is set with `x509_alg_rsaEncryption`, while `version` specifies 0.
gsk_construct_public_key_rsa()

Constructs an RSA public key from its component values.

Format

```c
#include <gskcms.h>

gsk_status gsk_construct_public_key_rsa (  
gsk_buffer * modulus,  
gsk_buffer * exponent,  
x509_public_key_info * public_key)
```

Parameters

- **modulus**
  - Specifies the modulus (n).

- **exponent**
  - Specifies the public exponent (e).

- **public_key**
  - Returns the public key.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

- **[ASN_ELEMENTS_MISSING]**
  - Required data element is missing.

Usage

The gsk_construct_public_key_rsa() function constructs pkcs_public_key_info from its RSA public key components. The x509_public_key_info structure's x509_algorithm_identifier is set with x509_alg_rsaEncryption.
gsk_construct_renewal_request()

Constructs a certification renewal request as described in PKCS #10.

Format

```c
#include <gskcms.h>
gsk_status gsk_construct_renewal_request (
    x509_public_key_info * public_key,
    pkcs_private_key_info * private_key,
    x509_algorithm_type signature_algorithm,
    const char * subject_name,
    x509_extensions * extensions,
    pkcs_cert_request * request)
```

Parameters

- **public_key**
  Specifies the public key for the certification request.

- **private_key**
  Specifies the private key for the certification request.

- **signature_algorithm**
  Specifies the signature algorithm used to sign the constructed request.

- **subject_name**
  Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

- **extensions**
  Specifies certificate extensions to be included in the certification request. Specify NULL for this parameter if no certificate extensions are provided.

- **request**
  Returns the certification renewal request as a pkcs_cert_request structure.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

- **[ASN_X500_NO_AVA_SEP]**
  An attribute value separator is missing.

- **[CMSERR_ALG_NOT_SUPPORTED]**
  The signature algorithm is not valid.

- **[CMSERR_BAD_KEY_SIZE]**
  The key size is not valid.

- **[CMSERR_KEY_MISMATCH]**
  The signing key type is not supported by the requested signature algorithm.

- **[CMSERR_NO_MEMORY]**
  Insufficient storage is available.

Usage

The `gsk_construct_renewal_request()` function constructs a certification request as described in PKCS #10 (Certification Request Syntax Standard) and returns the constructed request in the pkcs_cert_request structure `request`. 

Chapter 8. Certificate Management Services (CMS) API Reference 117
gsk_encode_export_request()

The gsk_encode_export_request() function can be called to create an export file containing the request for transmission to the certification authority.

The certification request will be signed using the key specified by the private_key parameter and the signature algorithm specified by the signature_algorithm parameter.

These signature algorithms are supported:

- **x509_alg_md2WithRsaEncryption**
  - RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

- **x509_alg_md5WithRsaEncryption**
  - RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

- **x509_alg_sha1WithRsaEncryption**
  - RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

- **x509_alg_sha224WithRsaEncryption**
  - RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

- **x509_alg_sha256WithRsaEncryption**
  - RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

- **x509_alg_sha384WithRsaEncryption**
  - RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

- **x509_alg_sha512WithRsaEncryption**
  - RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

- **x509_alg_dsaWithSha1**
  - Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

  When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

The extensions parameter can be used to provide certificate extensions for inclusion in the certification request. Whether or not a particular certificate extension will be included in the new certificate is determined by the certification authority.
gsk_construct_self_signed_certificate()

Constructs a self-signed certificate and returns it to the caller.

Format

```c
#include <gskcms.h>

gsk_status gsk_construct_self_signed_certificate (  
x509_algorithm_type  signature_algorithm,  
const_char *          subject_name,         
int                  num_days,              
gsk_boolean          ca_certificate,        
x509_extensions *     extensions,           
x509_public_key_info * public_key,          
pkcs_private_key_info * private_key,         
x509_certificate *    subject_certificate)
```

Parameters

`signature_algorithm`
Specifies the signature algorithm used to sign the constructed certificate.

`subject_name`
Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

`num_days`
Specifies the number of days for the certificate validity period as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified).

`ca_certificate`
Specify TRUE if this is a certification authority certificate or FALSE if this is an end user certificate.

`extensions`
Specifies the certificate extensions for the new certificate. Specify NULL for this parameter if no certificate extensions are supplied.

`public_key`
Specifies the public key for the constructed certificate.

`private_key`
Specifies the private key for the constructed certificate.

`subject_certificate`
Contains the constructed certificate.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- [CMSERR_ALG_NOT_SUPPORTED]
  The signature algorithm is not valid.

- [CMSERR_BAD_KEY_SIZE]
  The key size is not valid.

- [CMSERR_BAD_SUBJECT_NAME]
  The subject name is not valid.
Usage
The `gsk_construct_self_signed_certificate()` routine will construct an X.509 certificate as described in RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). A certification authority certificate will have basic constraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have no basic constraints limitations or key usage limitations. The constructed certificate is then returned in the X509_certificate structure `subject_certificate`.

These signature algorithms are supported:

- `x509_alg_md2WithRsaEncryption`
  RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

- `x509_alg_md5WithRsaEncryption`
  RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

- `x509_alg_sha1WithRsaEncryption`
  RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

- `x509_alg_sha224WithRsaEncryption`
  RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

- `x509_alg_sha256WithRsaEncryption`
  RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

- `x509_alg_sha384WithRsaEncryption`
  RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

- `x509_alg_sha512WithRsaEncryption`
  RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

- `x509_alg_dsaWithSha1`
  Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

When executing in FIPS mode, signature algorithms `x509_alg_md2WithRSAEncryption` and `x509_alg_md5WithRsaEncryption` are not supported.

If not in FIPS mode, an RSA key size must be between 512 and 4096 bits and will be rounded up to a multiple of 16 bits. A DSA key size must be between 512 and 1024 bits and will be rounded up to a multiple of 64 bits.

In FIPS mode, an RSA key size must be between 1024 and 4096 bits and will be rounded up to a multiple of 16 bits. A DSA key size must be 1024 bits.

Note: A self-signed end-entity certificate (server or client certificate) is not recommended for use in production environments and should only be used to facilitate test environments prior to production. Self-signed certificates do not imply any level of security or authenticity of the certificate because, as their name implies, they are signed by the same key that is contained in the certificate. On the other hand, certificates that are signed by a certificate authority indicate that, at least at the time of signature, the certificate authority approved the information contained in the certificate.
gsk_construct_signed_certificate()

Constructs a signed certificate for a certificate request.

Format

```c
#include <gskcms.h>

gsk_status gsk_construct_signed_certificate (
    pkcs_cert_key * signer_certificate,
    pkcs_cert_request * request,
    x509_algorithm_type signature_algorithm,
    int num_days,
    gsk_boolean ca_certificate,
    x509_extensions * extensions,
    x509_certificate * certificate)
```

Parameters

- `signer_certificate`
  Specifies the signing certificate with private key.

- `request`
  Specifies the PKCS #10 certification request stream in either binary DER-encoded format or in Base64 format. A Base64 stream is in the local code page.

- `signature_algorithm`
  Specifies the signature algorithm used to sign the constructed certificate.

- `num_days`
  Specifies the number of days for the certificate validity period as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified).

- `ca_certificate`
  Specify TRUE if this is a certification authority certificate or FALSE if this is an end user certificate.

- `extensions`
  Specifies the certificate extensions for the new certificate. Specify NULL for this parameter if no certificate extensions are supplied.

- `certificate`
  Contains the constructed signed certificate.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- `[CMSERR_ALG_NOT_SUPPORTED]`
  The key algorithm or the signature algorithm is not valid.

- `[CMSERR_BAD_ENCODING]`
  The certificate request stream is not valid.

- `[CMSERR_BAD_KEY_SIZE]`
  The key size is not valid.

- `[CMSERR_BAD_SIGNATURE]`
  The request signature is not correct.

- `CMSERR_CA_NOT_SUPPLIED[]`
  CA certificate is not supplied.
The gsk_construct_signed_certificate() routine will construct an X.509 certificate as described in RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). The new certificate will be signed using the certificate specified by the signer_certificate parameter. A certification authority certificate will have basic constraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have basic constraints and key usage extensions which allow the certificate to be used for authentication, digital signatures, and data encryption (except for a DSA key which cannot be used for data encryption). The certificate expiration will be set to the earlier of the requested expiration date and the expiration date of the signing certificate.

The signing certificate must have an associated private key, the Basic Constraints extension must either be omitted or must have the CA indicator set, and the KeyUsage extension must either be omitted or must allow signing certificates.

A CA certificate will have SubjectKeyIdentifier, KeyUsage and BasicConstraints extensions while an end user certificate will have SubjectKeyIdentifier and KeyUsage extensions. An AuthorityKeyIdentifier extension will be created if the signing certificate has a SubjectKeyIdentifier extension. The application can supply additional extensions through the extensions parameter. An AuthorityKeyIdentifier, KeyUsage or BasicConstraints extension provided by the application will replace the default extension constructed for the certificate, however a SubjectKeyIdentifier extension provided by the application will be ignored.

Certificate extensions can also be contained within the certification request. A certificate extension supplied by the application will override a certificate extension of the same type contained in the certification request. The certificate extension found in the certification request will be copied unmodified to the new certificate with these exceptions:

- The AuthorityInfoAccess, AuthorityKeyIdentifier, BasicConstraints, CrlDistributionPoints, IssuerAltName, NameConstraints, PolicyConstraints, PolicyMappings, and PrivateKeyUsagePeriod extensions will not be copied.
- The keyCertSign and crlSign flags in the KeyUsage extension will be modified based upon the value of the ca_certificate parameter.
These signature algorithms are supported:

- **x509_alg_md2WithRsaEncryption**
  - RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

- **x509_alg_md5WithRsaEncryption**
  - RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

- **x509_alg_sha1WithRsaEncryption**
  - RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

- **x509_alg_sha224WithRsaEncryption**
  - RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

- **x509_alg_sha256WithRsaEncryption**
  - RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

- **x509_alg_sha384WithRsaEncryption**
  - RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

- **x509_alg_sha512WithRsaEncryption**
  - RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

- **x509_alg_dsaWithSha1**
  - Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

When executing in FIPS mode, signature algorithms **x509_alg_md2WithRSAEncryption** and **x509_alg_md5WithRsaEncryption** are not supported.

No certification path validation is performed by the `gsk_construct_signed_certificate()` routine. An error will be returned if the requested subject name is the same as the subject name in the signing certificate.
gsk_copy_attributes_signers()

Copies a gsk_attributes_signers structure.

Format

```c
#include <gskcms.h>

gsk_status gsk_copy_attributes_signers (  
gsk_attributes_signers * in_attributesSigners,  
gsk_attributes_signers * out_attributesSigners)
```

Parameters

**in_attributesSigners**
Specifies the source gsk_attributes_signers structure.

**out_attributesSigners**
Specifies the destination gsk_attributes_signers structure. The application should call the **gsk_free_attributes_signers()** routine when the gsk_attributes_signers structure is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

**[CMSERR_NO_MEMORY]**
Insufficient storage is available.

Usage

The **gsk_copy_attributes_signers()** routine will allocate the output gsk_attributes_signers structure and then copy the input gsk_attributes_signers structure to the output gsk_attributes_signers structure. Storage for the base gsk_attributes_signers structure (**in_attributesSigners**) is provided by the application.
gsk_copy_buffer()

Copies a buffer.

Format

```c
#include <gskcms.h>
gsk_status gsk_copy_buffer (  
  gsk_buffer *  in_buffer,  
  gsk_buffer *  out_buffer)
```

Parameters

- **in_buffer**
  - Specifies the source buffer.

- **out_buffer**
  - Specifies the destination buffer. The application should call the `gsk_free_buffer()` routine when the buffer is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. This is a possible error:

- **[CMSERR_NO_MEMORY]**
  - Insufficient storage is available.

Usage

The `gsk_copy_buffer()` routine will allocate the output buffer and then copy the input buffer to the output buffer. Storage for the base gsk_buffer structure is provided by the caller.
gsk_copy_certificate()

Copies an X.509 certificate.

Format

```c
#include <gskcms.h>

#include <gskcms.h>

gsk_status gsk_copy_certificate ( x509_certificate * in_certificate, x509_certificate * out_certificate)
```

Parameters

`in_certificate`  
Specifies the source certificate.

`out_certificate`  
Specifies the destination certificate. The application should call the `gsk_free_certificate()` routine when the certificate is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. This is a possible error:

[CMSERR_NO_MEMORY]  
Insufficient storage is available.

Usage

The `gsk_copy_certificate()` routine will allocate the output certificate and then copy the input certificate to the output certificate. Storage for the base x509_certificate structure is provided by the caller.
gsk_copy_certificate_extension()

Copies an X.509 certificate extension.

Format

```
#include <gskcms.h>

gsk_status gsk_copy_certificate_extension (       
x509_extension * in_extension,       
x509_extension * out_extension)
```

Parameters

- **in_extension**: Specifies the source certificate extension.
- **out_extension**: Specifies the destination certificate extension. The application should call the `gsk_free_certificate_extension()` routine when the extension is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. This is a possible error:

- **[CMSERR_NO_MEMORY]**
  - Insufficient storage is available.

Usage

The `gsk_copy_certificate_extension()` routine will allocate the output certificate extension and then copy the input certificate extension to the output certificate extension. Storage for the base `x509_extension` structure is provided by the caller.
gsk_copy_certification_request()

Copies a PKCS #10 certification request.

Format

```c
#include <gskcms.h>

gsk_status gsk_copy_certification_request (pkcs_cert_request *in_request,
                                        pkcs_cert_request *out_request)
```

Parameters

- `in_request`: Specifies the source certification request.
- `out_request`: Specifies the destination certification request. The application should call the `gsk_free_certification_request()` routine when the certification request is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. This is a possible error:

- `[CMSERR_NO_MEMORY]`
  - Insufficient storage is available.

Usage

The `gsk_copy_certification_request()` routine will allocate the output certification request and then copy the input certification request to the output certification request. Storage for the base `pkcs_cert_request` structure is provided by the application.
gsk_copy_content_info()

Covers PKCS #7 content information.

**Format**

```c
#include <gskcms.h>

gsk_status gsk_copy_content_info (  
    pkcs_content_info * in_info,  
    pkcs_content_info * out_info)
```

**Parameters**

- **in_info**
  Specifies the source content information.

- **out_info**
  Specifies the destination content information. The application should call the `gsk_free_content_info()` routine when the content information is no longer needed.

**Results**

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. This is a possible error:

[CMSERR_NO_MEMORY]

Insufficient storage is available.

**Usage**

The `gsk_copy_content_info()` routine will allocate the output content information and then copy the input content information to the output content information. Storage for the base pkcs_content_info structure is provided by the application.
Copies an X.509 certificate revocation list.

Format

```c
#include <gskcms.h>

gsk_status gsk_copy_crl (
    x509_crl * in_crl,
    x509_crl * out_crl)
```

Parameters

- `in_crl`
  Specifies the source certificate revocation list.

- `out_crl`
  Specifies the destination certificate revocation list. The application should call the `gsk_free_crl()` routine when the certificate revocation list is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. This is a possible error:

- `[CMSERR_NO_MEMORY]`
  Insufficient storage is available.

Usage

The `gsk_copy_crl()` routine will allocate the output certificate revocation list and then copy the input list to the output list. Storage for the base `x509_crl` structure is provided by the caller.
gsk_copy_name()

Copies an X.509 name.

Format

```c
#include <gskcms.h>

gsk_status gsk_copy_name (  
    x509_name *  in_name,  
    x509_name *  out_name)
```

Parameters

*in_name*

Specifies the source name.

*out_name*

Specifies the destination name. The application should call the `gsk_free_name()` routine when the name is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. This is a possible error:

[CMSERR_NO_MEMORY]

Insufficient storage is available.

Usage

The `gsk_copy_name()` routine will allocate the output name and then copy the input name to the output name. Storage for the base `x509_name` structure is provided by the caller.
**gsk_copy_private_key_info()**

Copies the private key information.

**Format**

```c
#include <gskcms.h>

gsk_status gsk_copy_private_key_info (   pkcs_private_key_info * in_info,   pkcs_private_key_info * out_info)
```

**Parameters**

- **in_info**
  - Specifies the source private key information.

- **out_info**
  - Specifies the destination private key information. The application should call the `gsk_free_private_key_info()` routine when the private key is no longer needed.

**Results**

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. This is a possible error:

**[CMSERR_NO_MEMORY]**

Insufficient storage is available.

**Usage**

The `gsk_copy_private_key_info()` routine will allocate the output private key and then copy the input key to the output key. Storage for the base `pkcs_private_key_info` structure is provided by the caller.
**gsk_copy_public_key_info()**

Copies the public key information.

**Format**

```c
#include <gskcms.h>

gsk_status gsk_copy_public_key_info (  
    x509_public_key_info * in_info,  
    x509_public_key_info * out_info)
```

**Parameters**

- `in_info`
  - Specifies the source public key information.

- `out_info`
  - Specifies the destination public key information. The application should call the `gsk_free_public_key_info()` routine when the public key is no longer needed.

**Results**

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. This is a possible error:

**[CMSERR_NO_MEMORY]**

Insufficient storage is available.

**Usage**

The `gsk_copy_public_key_info()` routine will allocate the output public key and then copy the input key to the output key. Storage for the base `x509_public_key_info` structure is provided by the caller.
**gsk_copy_record()**

Copies a database record.

**Format**

```c
#include <gskcms.h>

gsk_status gsk_copy_record (  
gskdb_record * in_record,  
gskdb_record ** out_record)
```

**Parameters**

*in_record*
- Specifies the source record.

*out_record*
- Returns the copied record. The application should call the `gsk_free_record()` routine when the record is no longer needed.

**Results**

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. This is a possible error:

`[CMSERR_NO_MEMORY]`
Insufficient storage is available.

**Usage**

The `gsk_copy_record()` routine will allocate the output record and then copy the input record to the output record. The address of the copied record will then be returned to the application.
gsk_create_certification_request()

Creates a PKCS #10 certification request.

Format

```c
#include <gskcms.h>

gsk_status gsk_create_certification_request (  
gsk_handle db_handle,  
const char * label,  
x509_algorithm_type signature_algorithm,  
int key_size,  
const char * subject_name,  
x509_extensions * extensions)
```

Parameters

- **db_handle**
  Specifies the database handle returned by the `gsk_create_database()` routine or the `gsk_open_database()` routine. This must be a request database and not a key database.

- **label**
  Specifies the label for the new database record. The label is specified in the local code page.

- **signature_algorithm**
  Specifies the signature algorithm for the certificate.

- **key_size**
  Specifies the key size in bits.

- **subject_name**
  Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

- **extensions**
  Specifies certificate extensions to be included in the certification request. Specify NULL for this parameter if no certificate extensions are provided.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_ALG_NOT_SUPPORTED]**
  The signature algorithm is not supported.

- **[CMSERR_BACKUP_EXISTS]**
  The backup file already exists.

- **[CMSERR_BAD_HANDLE]**
  The database handle is not valid.

- **[CMSERR_BAD_KEY_SIZE]**
  The key size is not valid.

- **[CMSERR_BAD_LABEL]**
  The record label is not valid.

- **[CMSERR_FIPS_KEY_PAIR_CONSISTENCY]**
  FIPS mode key generation failed pair-wise consistency check.
gsk_create_certification_request()

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certification requests.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage
The gsk_create_certification_request() routine creates a request for a new certificate as described in PKCS #10 (Certification Request Syntax Standard). The request is then stored in the request database. The gsk_export_certification_request() routine can be called to create an export file containing the request for transmission to the certification authority.

The gsk_create_certification_request() routine is similar to the gsk_create_renewal_request() routine. Both routines create a PKCS #10 certification request. The difference is the gsk_create_certification_request() routine generates a new public/private key pair while the gsk_create_renewal_request() routine uses the public/private key pair provided by the application.

These signature algorithms are supported:

- **x509_alg_md2WithRsaEncryption**
  RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

- **x509_alg_md5WithRsaEncryption**
  RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

- **x509_alg_sha1WithRsaEncryption**
  RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

- **x509_alg_sha224WithRsaEncryption**
  RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

- **x509_alg_sha256WithRsaEncryption**
  RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

- **x509_alg_sha384WithRsaEncryption**
  RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

- **x509_alg_sha512WithRsaEncryption**
  RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

- **x509_alg_dsaWithSha1**
  Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

When executing in FIPS mode, signature algorithms x509_alg_md2WithRsaEncryption and x509_alg_md5WithRsaEncryption are not supported.

If not in FIPS mode, an RSA key size must be between 512 and 4096 bits and will be rounded up to a multiple of 16 bits. A DSA key size must be between 512 and 1024 bits and will be rounded up to a multiple of 64 bits.
In FIPS mode, an RSA key size must be between 1024 and 4096 bits and will be rounded up to a multiple of 16 bits. A DSA key size must be 1024 bits.

The record label is used as a friendly name for the database entry. It can be any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string.

The *extensions* parameter can be used to provide certificate extensions for inclusion in the certification request. Whether or not a particular certificate extension will be included in the new certificate is determined by the certification authority.

The database must be open for update in order to add the new request. The database file is updated as part of the *gsk_create_certification_request()* processing. A temporary database file is created using the same name as the database file with "new" appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.
gsk_create_database() creates a key or request database.

Format

```c
#include <gskcms.h>
gsk_status gsk_create_database (  
    char * filename,  
    char * password,  
    gskdb_database_type db_type,  
    gsk_size record_length,  
    gsk_time pwd_expiration,  
    gsk_handle * db_handle)
```

Parameters

- **filename**: Specifies the database file name in the local code page. The length of the fully-qualified file name cannot exceed 251.
- **password**: Specifies the database password in the local code page. The password must consist of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string. The user will be prompted to enter the password if NULL is specified for this parameter.
- **db_type**: Specifies the database type and must be gskdb_dbtype_key for a key database or gskdb_dbtype_request for a certification request database.
- **record_length**: Specifies the database record length. The default record length will be used if zero is specified for this parameter. All records in the database will have this length. The minimum record length is 2500. The default record length is 5000.
- **pwd_expiration**: Specifies the database password expiration time as the number of seconds since the POSIX epoch. A value of 0 indicates the password does not expire.
- **db_handle**: Returns the database handle. The application should call the `gsk_close_database()` routine when it no longer needs access to the database.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_BAD_FILENAME]**
  The database file name is not valid.
- **[CMSERR_DB_EXISTS]**
  The database already exists.
- **[CMSERR_INCORRECT_DBTYPE]**
  The database type is not valid.
- **[CMSERR_IO_CANCELED]**
  The user canceled the password prompt.
- **[CMSERR_IO_ERROR]**
  An input/output request failed.
gsk_create_database()

[CMSERR_LENGTH_TOO_SMALL]
The record length is less than the minimum value.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_OPEN_FAILED]
Unable to open the key database.

Usage
The gsk_create_database() routine will create a key or request database. The database must not already exist. A new key database will contain an initial set of Certificate Authority certificates for use in validating certificate signatures.

- If this function is called while executing in FIPS mode, the new database will meet FIPS 140-2 criteria.
- Such a database:
  - Can be read while executing in FIPS mode and when not in FIPS mode.
  - Can be updated only when executing in FIPS mode.

- A database created while not executing in FIPS mode:
  - Can be updated or read when not in FIPS mode
  - Cannot be used while executing in FIPS mode
gsk_create_database_renewal_request()

Creates a PKCS #10 certification renewal request.

Format

#include <gskcms.h>

gsk_status gsk_create_database_renewal_request (
    gsk_handle db_handle,
    const char *label,
    x509_public_key_info *public_key,
    pkcs_private_key_info *private_key,
    x509_algorithm_type signature_algorithm,
    const char *subject_name,
    x509_extensions *extensions)

Parameters

db_handle
    Specifies the database handle returned by the gsk_create_database() routine or the
    gsk_open_database() routine. This must be a request database and not a key database.

label
    Specifies the label for the request database record. The label is specified in the local code page.

public_key
    Specifies the public key for the certification request.

private_key
    Specifies the private key for the certification request.

signature_algorithm
    Specifies the signature algorithm to be used for the request signature.

subject_name
    Specifies the distinguished name for the certificate subject. The distinguished name is specified in the
    local code page and consists of one or more relative distinguished name components separated by
    commas.

extensions
    Specifies certificate extensions to be included in the certification request. Specify NULL for this
    parameter if no certificate extensions are provided.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes
listed in the gskcms.h include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
    The signature algorithm is not valid.

[CMSERR_BACKUP_EXISTS]
    The backup file already exists.

[CMSERR_BAD_HANDLE]
    The database handle is not valid.

[CMSERR_BAD_KEY_SIZE]
    The key size is not valid.

[CMSERR_BAD_LABEL]
    The record label is not valid.
gsk_create_database_renewal_request()

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certification requests.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_KEY_MISMATCH]
The supplied private key cannot be used to sign a certificate or the private key type is not supported for the requested signature algorithm.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage
The gsk_create_database_renewal_request() routine creates a certification request as described in PKCS #10 (Certification Request Syntax Standard). The request is then stored in the request database. The gsk_export_certification_request() routine can be called to create an export file containing the request for transmission to the certification authority.

The gsk_create_database_renewal_request() routine is similar to the gsk_create_certification_request() routine. Both routines create a PKCS #10 certification request. The difference is the gsk_create_certification_request() routine generates a new public/private key pair while the gsk_create_database_renewal_request() routine uses the public/private key pair provided by the application.

The renewal request will be signed using the key specified by the private_key parameter and the signature algorithm specified by the signature_algorithm parameter.

These signature algorithms are supported:

x509_alg_md2WithRsaEncryption
  RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
  RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
  RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
  RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
  RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
  RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
  RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}
gsk_create_database_renewal_request()

x509_alg_dsaWithSha1
   Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

- When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and
- x509_alg_md5WithRsaEncryption are not supported.
gsk_create_database_signed_certificate()

Creates a signed certificate as part of a set of certificates.

Format

```
#include <gskcms.h>

gsk_status gsk_create_database_signed_certificate (  
  gsk_handle db_handle,  
  const char * ca_label,  
  const char * record_label,  
  x509_algorithm_type key_algorithm,  
  int key_size,  
  gsk_buffer * key_parameters,  
  x509_algorithm_type signature_algorithm,  
  const char * subject_name,  
  int num_days,  
  gsk_boolean ca_certificate,  
  x509_extensions * extensions )
```

Parameters

- **db_handle**
  Specifies the database handle returned by the `gsk_create_database()` routine or the `gsk_open_database()` routine. This must be a key database and not a request database.

- **ca_label**
  Specifies the label of the certificate to be used to sign the new certificate. The key usage for the certificate must allow certificate signing. The label is specified in the local code page.

- **record_label**
  Specifies the label for the new database record. The label is specified in the local code page.

- **key_algorithm**
  Specifies the certificate key algorithm.

- **key_size**
  Specifies the certificate key size in bits.

- **key_parameters**
  Specifies the key generation parameters. Specify NULL for this parameter if the key algorithm does not require any key parameters.

- **signature_algorithm**
  Specifies the signature algorithm used for the certificate signature.

- **subject_name**
  Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

- **num_days**
  Specifies the number of days for the certificate validity period as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified).

- **ca_certificate**
  Specify TRUE if this is a certification authority certificate or FALSE if this is an end user certificate.

- **extensions**
  Specifies the certificate extensions for the new certificate. Specify NULL for this parameter if no certificate extensions are supplied.
gsk_create_database_signed_certificate()

Results
The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The key algorithm or the signature algorithm is not valid.

[CMSERR_BACKUP_EXISTS]
The backup file already exists.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_LABEL]
The record label or CA certificate label is not valid.

[CMSERR_BAD_SUBJECT_NAME]
The subject name is not valid.

[CMSERR_DUPLICATE_EXTENSION]
Supplied extensions contain a duplicate extension.

[CMSERR_EXPIRED]
The signer certificate is expired.

[CMSERR_FIPS_KEY_PAIR_CONSISTENCY]
FIPS mode key generation failed pair-wise consistency check.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_USAGE]
The signer certificate key usage does not allow signing certificates.

[CMSERR_IO_ERROR]
Unable to read or write a database record.

[CMSERR_ISSUER_NOT_CA]
The signer certificate is not for a certification authority.

[CMSERR_KEY_MISMATCH]
The signer certificate key cannot be used to sign a certificate.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
The signer certificate does not have a private key.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_SUBJECT_IS_CA]
The requested subject name is the same as the signer name.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.
Usage
The gsk_create_database_signed_certificate() routine will generate an X.509 certificate as described in RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). The certificate will be signed using an existing certificate as specified by the ca_label parameter and the signature algorithm specified by the signature_algorithm parameter.

These signature algorithms are supported:

<table>
<thead>
<tr>
<th>x509_alg_md2WithRsaEncryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x509_alg_md5WithRsaEncryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x509_alg_sha1WithRsaEncryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x509_alg_sha224WithRsaEncryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x509_alg_sha256WithRsaEncryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x509_alg_sha384WithRsaEncryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x509_alg_sha512WithRsaEncryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x509_alg_dsaWithSha1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}</td>
</tr>
</tbody>
</table>

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

A certification authority (CA) certificate will have basic constraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have basic constraints and key usage extensions as follows:

- An RSA key can be used for authentication, digital signature, and data encryption.
- A DSS key can be used for authentication and digital signature.
- A Diffie-Hellman key can be used for key agreement.

The new certificate will be stored in the key database using the supplied record label. The gsk_export_certificate() routine can be called to create an export file containing the certificate for transmission to another system.

The following key algorithms are supported:

<table>
<thead>
<tr>
<th>x509_alg_rsaEncryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA encryption - {1.2.840.113549.1.1.1}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x509_alg_idDsa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Signature Standard (DSS) - {1.2.840.10040.4.1}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x509_alg_dhPublicNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diffie-Hellman (DH) - {1.2.840.10046.2.1}</td>
</tr>
</tbody>
</table>

RSA keys
- Can be used for both CA certificates and end user certificates
- Key size when not in FIPS mode is between 512 and 4096 bits rounded up to a multiple of 16
Key size in FIPS mode is between 1024 and 4096 bits rounded up to a multiple of 16

- No key parameters

DSS keys
- Can be used for both CA certificates and end user certificates
- Key size when not in FIPS mode is between 512 and 1024 bits rounded up to a multiple of 64
- Key size in FIPS mode of 1024 bits
- Key parameters encoded as an ASN.1 sequence consisting of the prime P, the subprime Q and the base G. Refer to FIPS 186-2 (Digital Signature Standard) for more information on the key parameters. The `gsk_generate_key_parameters()` routine can be used to generate the key parameters.

DH keys
- Can be used only for end user certificates
- Key size when not in FIPS mode is between 512 and 2048 bits rounded up to a multiple of 64
- Key size in FIPS mode of 2048 bits
- Key parameters encoded as an ASN.1 sequence consisting of the prime P, the base G, the subprime Q and the subgroup factor J. Refer to RFC 2631 (Diffie-Hellman Key Agreement Method) for more information on the key parameters. The `gsk_generate_key_parameters()` routine can be used to generate the key parameters.

The record label is used as a friendly name for the database entry. It can be any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string.

A CA certificate will have SubjectKeyIdentifier, KeyUsage and BasicConstraints extensions while an end user certificate will have SubjectKeyIdentifier and KeyUsage extensions. An AuthorityKeyIdentifier extension will be created if the signing certificate has a SubjectKeyIdentifier extension. The application can supply additional extensions through the extensions parameter. An AuthorityKeyIdentifier, KeyUsage or BasicConstraints extension provided by the application will replace the default extension constructed for the certificate, however a SubjectKeyIdentifier extension provided by the application will be ignored.

The database must be open for update in order to add the new certificate. The database file is updated as part of the `gsk_create_database_signed_certificate()` processing. A temporary database file is created using the same name as the database file with ".new" appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.
gsk_create_renewal_request()

Creates a PKCS #10 certification renewal request.

This function is deprecated. Use gsk_create_database_renewal_request() instead.

Format

```c
#include <gskcms.h>
gsk_status gsk_create_renewal_request (
    gsk_handle db_handle,
    const char *label,
    x509_public_key_info *public_key,
    pkcs_private_key_info *private_key,
    const char *subject_name,
    x509_extensions *extensions)
```

Parameters

- **db_handle**
  Specifies the database handle returned by the gsk_create_database() routine or the gsk_open_database() routine. This must be a request database and not a key database.

- **label**
  Specifies the label for the request database record. The label is specified in the local code page.

- **public_key**
  Specifies the public key for the certification request.

- **private_key**
  Specifies the private key for the certification request.

- **subject_name**
  Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

- **extensions**
  Specifies certificate extensions to be included in the certification request. Specify NULL for this parameter if no certificate extensions are provided.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

- **[CMSERR_BACKUP_EXISTS]**
  The backup file already exists.

- **[CMSERR_BAD_HANDLE]**
  The database handle is not valid.

- **[CMSERR_BAD_KEY_SIZE]**
  The key size is not valid.

- **[CMSERR_BAD_LABEL]**
  The record label is not valid.

- **[CMSERR_INCORRECT_DBTYPE]**
  The database type does not support certification requests.

- **[CMSERR_IO_ERROR]**
  Unable to write record.
Usage

The `gsk_create_renewal_request()` routine creates a certification request as described in PKCS #10 (Certification Request Syntax Standard). The request is then stored in the request database. The `gsk_export_certification_request()` routine can be called to create an export file containing the request for transmission to the certification authority.

The `gsk_create_renewal_request()` routine is similar to the `gsk_create_certification_request()` routine. Both routines create a PKCS #10 certification request. The difference is the `gsk_create_certification_request()` routine generates a new public/private key pair while the `gsk_create_renewal_request()` routine uses the public/private key pair provided by the application.

The record label is used as a friendly name for the database entry. It can be any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string.

The extensions parameter can be used to provide certificate extensions for inclusion in the certification request. Whether or not a particular certificate extension will be included in the new certificate is determined by the certification authority.
gsk_create_self_signed_certificate()

Creates a self-signed certificate.

Format

```c
#include <gskcms.h>

gsk_status gsk_create_self_signed_certificate (  
gsk_handle db_handle,  
const char * label,  
x509_algorithm_type signature_algorithm,  
int key_size,  
const char * subject_name,  
int num_days,  
gsk_boolean ca_certificate,  
x509_extensions * extensions)
```

Parameters

- **db_handle**
  Specifies the database handle returned by the `gsk_create_database()` routine or the `gsk_open_database()` routine. This must be a key database and not a request database.

- **label**
  Specifies the label for the new database record. The label is specified in the local code page.

- **signature_algorithm**
  Specifies the certificate signature algorithm.

- **key_size**
  Specifies the key size in bits.

- **subject_name**
  Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

- **num_days**
  Specifies the number of days for the certificate validity period as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified).

- **ca_certificate**
  Specify TRUE if this is a certification authority certificate or FALSE if this is an end user certificate.

- **extensions**
  Specifies the certificate extensions for the new certificate. Specify NULL for this parameter if no certificate extensions are supplied.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_ALG_NOT_SUPPORTED]**
  The signature algorithm is not valid.

- **[CMSERR_BACKUP_EXISTS]**
  The backup file already exists.

- **[CMSERR_BAD_HANDLE]**
  The database handle is not valid.
gsk_create_self_signed_certificate()

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_LABEL]
The record label is not valid.

[CMSERR_BAD_SUBJECT_NAME]
The subject name is not valid.

[CMSERR_DUPLICATE_EXTENSION]
Supplied extensions contain a duplicate extension.

[CMSERR_FIPS_KEY_PAIR_CONSISTENCY]
FIPS mode key generation failed pair-wise consistency check.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_KEY_MISMATCH]
The signer certificate key cannot be used to sign a certificate.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage
The gsk_create_self_signed_certificate() routine will generate a self-signed X.509 certificate as described in RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). A certification authority certificate will have basic constraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have no basic constraints or key usage limitations. The new certificate is then stored in the key database. The gsk_export_certificate() routine can be called to create an export file containing the certificate for transmission to another system.

These signature algorithms are supported:

- **x509_alg_md2WithRsaEncryption**
  RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

- **x509_alg_md5WithRsaEncryption**
  RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

- **x509_alg_sha1WithRsaEncryption**
  RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

- **x509_alg_sha224WithRsaEncryption**
  RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

- **x509_alg_sha256WithRsaEncryption**
  RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}
x509_alg_sha384WithRsaEncryption
   RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
   RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
   Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and
x509_alg_md5WithRsaEncryption are not supported.

If not in FIPS mode, an RSA key size must be between 512 and 4096 bits and will be rounded up to a
multiple of 16 bits. A DSA key size must be between 512 and 1024 bits and will be rounded up to a
multiple of 64 bits.

In FIPS mode, an RSA key size must be between 1024 and 4096 bits and will be rounded up to a multiple
of 16 bits. A DSA key size must be 1024 bits.

The record label is used as a friendly name for the database entry. It can be any value and consists of
characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be
an empty string.

Both a CA certificate and an end user certificate will have SubjectKeyIdentifier, AuthorityKeyIdentifier,
KeyUsage and BasicConstraints extensions. The application can supply additional extensions through the
extensions parameter. An AuthorityKeyIdentifier, KeyUsage or BasicConstraints extension provided by the
application will replace the default extension created for the certificate, however a SubjectKeyIdentifier
extension provided by the application will be ignored.

The database must be open for update in order to add the new certificate. The database file is updated as
part of the gsk_create_self_signed_certificate() processing. A temporary database file is created using
the same name as the database file with "_.new" appended to the name. The database file is then
overwritten and the temporary database file is deleted. The temporary database file will not be deleted if
an error occurs while rewriting the database file.

Note: A self-signed end-entity certificate (server or client certificate) is not recommended for use in
production environments and should only be used to facilitate test environments prior to production.
Self-signed certificates do not imply any level of security or authenticity of the certificate because,
as their name implies, they are signed by the same key that is contained in the certificate. On the
other hand, certificates that are signed by a certificate authority indicate that, at least at the time of
signature, the certificate authority approved the information contained in the certificate.
gsk_create_signed_certificate()

Creating a signed certificate.

This function is deprecated. Use gsk_create_signed_certificate_record() instead.

Format

```c
#include <gskcms.h>

gsk_status gsk_create_signed_certificate (  
db_handle,  
label,  
num_days,  
ca_certificate,  
extensions,  
cert_request,  
signed_certificate)
```

Parameters

- **db_handle**
  Specifies the database handle returned by the gsk_create_database() routine, the gsk_open_database() routine, or the gsk_open_keyring() routine. This must be a key database and not a request database.

- **label**
  Specifies the label for the certificate to be used to sign the new certificate. The label is specified in the local code page.

- **num_days**
  Specifies the number of days for the certificate validity period as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified).

- **ca_certificate**
  Specify TRUE if this is a certification authority certificate or FALSE if this is an end user certificate.

- **extensions**
  Specifies the certificate extensions for the new certificate. Specify NULL for this parameter if no certificate extensions are supplied.

- **cert_request**
  Specifies the PKCS #10 certification request stream in either binary DER-encoded format or in Base64 format. A Base64 stream is in the local code page.

- **signed_certificate**
  Returns the signed certificate in Base64 format. The Base64 stream will be in the local code page. The application should call the gsk_free_buffer() routine to release the certificate stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

- **[CMSERR_ALG_NOT_SUPPORTED]**
  The signature algorithm is not valid.

- **[CMSERR_BAD_ENCODING]**
  The certificate request stream is not valid.
gsk_create_signed_certificate()

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_LABEL]
The record label is not valid.

[CMSERR_BAD_SIGNATURE]
The request signature is not correct.

[CMSERR_DUPLICATE_EXTENSION]
Supplied extensions contain a duplicate extension.

[CMSERR_EXPIRED]
The signer certificate is expired.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_USAGE]
The signer certificate key usage does not allow signing certificates.

[CMSERR_ISSUER_NOT_CA]
The signer certificate is not for a certification authority.

[CMSERR_KEY_MISTMATCH]
The signer certificate key cannot be used to sign a certificate.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
The signer certificate does not have a private key.

[CMSERR_RECORD_NOT_FOUND]
The signer certificate is not found in the key database.

[CMSERR_SUBJECT_IS_CA]
The requested subject name is the same as the signer name.

Usage

The **gsk_create_signed_certificate()** routine will generate an X.509 certificate as described in RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). The new certificate will be signed using the certificate specified by the **label** parameter. A certification authority certificate will have basic constraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have basic constraints and key usage extensions which allow the certificate to be used for authentication, digital signatures, and data encryption (except for a DSA key which cannot be used for data encryption). The certificate expiration date will be set to the earlier of the requested expiration date and the expiration date of the signing certificate.

The signing certificate must have an associated private key, the BasicConstraints extension must either be omitted or must have the CA indicator set, and the KeyUsage extension must either be omitted or must allow signing certificates.

A CA certificate will have SubjectKeyIdentifier, KeyUsage and BasicConstraints extensions while an end user certificate will have SubjectKeyIdentifier and KeyUsage extensions. An AuthorityKeyIdentifier extension will be created if the signing certificate has a SubjectKeyIdentifier extension. The application can supply additional extensions through the extensions parameter. An AuthorityKeyIdentifier, KeyUsage or BasicConstraints extension provided by the application will replace the default extension created for the certificate, however a SubjectKeyIdentifier extension provided by the application will be ignored.
Certificate extensions can also be contained within the certification request. A certificate extension supplied by the application will override a certificate extension of the same type contained in the certification request. The certificate extensions found in the certification request will be copied unmodified to the new certificate with these exceptions:

- The AuthorityInfoAccess, AuthorityKeyIdentifier, BasicConstraints, CrlDistributionPoints, IssuerAltName, NameConstraints, PolicyConstraints, PolicyMappings, and PrivateKeyUsagePeriod extensions will not be copied
- The keyCertSign and crlSign flags in the KeyUsage extension will be modified based upon the value of the ca_certificate parameter.

No certification path validation is performed by the gsk_create_signed_certificate() routine. An error will be returned if the requested subject name is the same as the subject name in the signing certificate.
Creates a signed certificate.

Format

```c
#include <gskcms.h>
gsk_status gsk_create_signed_certificate_record (  
    gsk_handle      db_handle,  
    const char *    label,  
    int             num_days,  
    gsk_boolean     ca_certificate,  
    x509_algorithm_type signature_algorithm,  
    x509_extensions * extensions,  
    gsk_buffer *    cert_request,  
    gsk_buffer *    signed_certificate)
```

Parameters

- **db_handle**
  Specifies the database handle returned by the `gsk_create_database()` routine, the `gsk_open_database()` routine, or the `gsk_open_keyring()` routine. This must be a key database and not a request database.

- **label**
  Specifies the label for the certificate to be used to sign the new certificate. The label is specified in the local code page.

- **num_days**
  Specifies the number of days for the certificate validity period as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified).

- **ca_certificate**
  Specify TRUE if this is a certification authority certificate or FALSE if this is an end user certificate.

- **signature_algorithm**
  Specifies the signature algorithm to be used for the certificate signature.

- **extensions**
  Specifies the certificate extensions for the new certificate. Specify NULL for this parameter if no certificate extensions are supplied.

- **cert_request**
  Specifies the PKCS #10 certification request stream in either binary DER-encoded format or in Base64 format. A Base64 stream is in the local code page.

- **signed_certificate**
  Returns the signed certificate in Base64 format. The Base64 stream will be in the local code page. The application should call the `gsk_free_buffer()` routine to release the certificate stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_ALG_NOT_SUPPORTED]**
  The signature algorithm is not valid.

- **[CMSERR_BACKUP EXISTS]**
  The backup file already exists.
**gsk_create_signed_certificate_record()**

- **[CMSERR_BAD_HANDLE]**
  The database handle is not valid.

- **[CMSERR_BAD_KEY_SIZE]**
  The key size is not valid.

- **[CMSERR_BAD_LABEL]**
  The record label is not valid.

- **[CMSERR_BAD_SUBJECT_NAME]**
  The subject name is not valid.

- **[CMSERR_DUPLICATE_EXTENSION]**
  Supplied extensions contain a duplicate extension.

- **[CMSERR_INCORRECT_DBTYPE]**
  The database type does not support certificates.

- **[CMSERR_INCORRECT_KEY_USAGE]**
  The signer certificate key usage does not allow signing certificates.

- **[CMSERR_IO_ERROR]**
  Unable to write record.

- **[CMSERR_KEY_MISMATCH]**
  The signer certificate key cannot be used to sign a certificate or the signer's key type is not supported for the requested signature algorithm.

- **[CMSERR_LABEL_NOT_UNIQUE]**
  The record label is not unique.

- **[CMSERR_NO_MEMORY]**
  Insufficient storage is available.

- **[CMSERR_RECORD_TOO_BIG]**
  The record is larger than the database record length.

- **[CMSERR_UPDATE_NOT_ALLOWED]**
  Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

**Usage**

The `gsk_create_signed_certificate_record()` routine will generate an X.509 certificate as described in RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). The new certificate will be signed using the certificate specified by the `label` parameter and the signature algorithm specified by the `signature_algorithm` parameter.

The following signature algorithms are supported:

- **x509_alg_md2WithRsaEncryption**
  RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

- **x509_alg_md5WithRsaEncryption**
  RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

- **x509_alg_sha1WithRsaEncryption**
  RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

- **x509_alg_sha224WithRsaEncryption**
  RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

- **x509_alg_sha256WithRsaEncryption**
  RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}
gsk_create_signed_certificate_record()

x509_alg_sha384WithRsaEncryption
   RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
   RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
   Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

- When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and
  x509_alg_md5WithRsaEncryption are not supported.

- If not in FIPS mode, an RSA key size must be between 512 and 4096 bits and will be rounded up to a
  multiple of 16 bits. A DSA key size must be between 512 and 1024 bits and will be rounded up to a
  multiple of 64 bits.

- In FIPS mode, an RSA key size must be between 1024 and 4096 bits and will be rounded up to a multiple
  of 16 bits. A DSA key size must be 1024 bits.

A certification authority certificate will have basic constraints and key usage extensions which allow the
certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will
have basic constraints and key usage extensions which allow the certificate to be used for authentication,
digital signatures, and data encryption (except for a DSA key which cannot be used for data encryption).
The certificate expiration date will be set to the earlier of the requested expiration date and the expiration
date of the signing certificate.

The signing certificate must have an associated private key, the BasicConstraints extension must either be
omitted or must have the CA indicator set, and the KeyUsage extension must either be omitted or must
allow signing certificates.

A CA certificate will have SubjectKeyIdentifier, KeyUsage and BasicConstraints extensions while an end
user certificate will have SubjectKeyIdentifier and KeyUsage extensions. An AuthorityKeyIdentifier
extension will be created if the signing certificate has a SubjectKeyIdentifier extension. The application can
supply additional extensions through the extensions parameter. An AuthorityKeyIdentifier, KeyUsage or
BasicConstraints extension provided by the application will replace the default extension created for the
certificate, however a SubjectKeyIdentifier extension provided by the application will be ignored.

Certificate extensions can also be contained within the certification request. A certificate extension supplied
by the application will override a certificate extension of the same type contained in the certification
request. The certificate extensions found in the certification request will be copied unmodified to the new
certificate with these exceptions:

- The AuthorityInfoAccess, AuthorityKeyIdentifier, BasicConstraints, CrlDistributionPoints, IssuerAltName,
  NameConstraints, PolicyConstraints, PolicyMappings, and PrivateKeyUsagePeriod extensions will not
  be copied

- The keyCertSign and crlSign flags in the KeyUsage extension will be modified based upon the value of
  the ca_certificate parameter.

No certification path validation is performed by the gsk_create_signed_certificate_record() routine. An
error will be returned if the requested subject name is the same as the subject name in the signing
certificate.
gsk_create_signed_certificate_set()

Creates a signed certificate as part of a set of certificates.

This function is deprecated. Use `gsk_create_database_signed_certificate()` instead.

Format

```
#include <gskcms.h>

gsk_status gsk_create_signed_certificate_set (  
gsk_handle             db_handle,  
const char *           ca_label,  
const char *           record_label,  
x509_algorithm_type   key_algorithm,  
int                    key_size,  
gsk_buffer *          key_parameters,  
const char *           subject_name,  
int                    num_days,  
gsk_boolean           ca_certificate,  
x509_extensions       extensions )
```

Parameters

db_handle
   Specifies the database handle returned by the `gsk_create_database()` routine, the `gsk_open_database()` routine. This must be a key database and not a request database.

certificate
   Specifies the label of the certificate to be used to sign the new certificate. The key usage for the certificate must allow certificate signing. The label is specified in the local code page.

record_label
   Specifies the label for the new database record. The label is specified in the local code page.

key_algorithm
   Specifies the certificate key algorithm.

key_size
   Specifies the certificate key size in bits.

key_parameters
   Specifies the key generation parameters. Specify NULL for this parameter if the key algorithm does not require any key parameters.

subject_name
   Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

num_days
   Specifies the number of days for the certificate validity period as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified).

certificate
   Specify TRUE if this is a certification authority certificate or FALSE if this is an end user certificate.

extensions
   Specifies the certificate extensions for the new certificate. Specify NULL for this parameter if no certificate extensions are supplied.
gsk_create_signed_certificate_set()

Results
The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The key algorithm or the signature algorithm is not valid.

[CMSERR_BACKUP_EXISTS]
The backup file already exists.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_LABEL]
The record label or CA certificate label is not valid.

[CMSERR_BAD_SUBJECT_NAME]
The subject name is not valid.

[CMSERR_DUPLICATE_EXTENSION]
Supplied extensions contain a duplicate extension.

[CMSERR_EXPIRED]
The signer certificate is expired.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_USAGE]
The signer certificate key usage does not allow signing certificates.

[CMSERR_IO_ERROR]
Unable to read or write a database record.

[CMSERR_ISSUER_NOT_CA]
The signer certificate is not for a certification authority.

[CMSERR_KEY_MISMATCH]
The signer certificate key cannot be used to sign a certificate.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
The signer certificate does not have a private key.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_SUBJECT_IS_CA]
The requested subject name is the same as the signer name.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.
**gsk_create_signed_certificate_set()**

**Usage**

The `gsk_create_signed_certificate_set()` routine will generate an X.509 certificate as described in RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). The certificate will be signed using an existing certificate as specified by the `ca_label` parameter. A certification authority (CA) certificate will have basic constraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have basic constraints and key usage extensions as follows:

- An RSA key can be used for authentication, digital signature, and data encryption
- A DSS key can be used for authentication and digital signature
- A Diffie-Hellman key can be used for key agreement

The new certificate will be stored in the key database using the supplied record label. The `gsk_export_certificate()` routine can be called to create an export file containing the certificate for transmission to another system.

These key algorithms are supported:

- **x509_alg_rsaEncryption**
  - RSA encryption - {1.2.840.113549.1.1.1}

- **x509_alg_idDsa**
  - Digital Signature Standard (DSS) - {1.2.840.10040.4.1}

- **x509_alg_dhPublicKey**
  - Diffie-Hellman (DH) - {1.2.840.10046.2.1}

**RSA keys**

- Can be used for both CA certificates and end user certificates
- Key size when not in FIPS mode is between 512 and 4096 bits rounded up to a multiple of 16
- Key size in FIPS mode is between 1024 and 4096 bits rounded up to a multiple of 16
- No key parameters

**DSS keys**

- Can be used for both CA certificates and end user certificates
- Key size when not in FIPS mode is between 512 and 1024 bits rounded up to a multiple of 64
- Key size in FIPS mode of 1024 bits
- Key parameters encoded as an ASN.1 sequence consisting of the prime P, the subprime Q and the base G. Refer to FIPS 186-2 (Digital Signature Standard) for more information on the key parameters. The `gsk_generate_key_parameters()` routine can be used to generate the key parameters.

**DH keys**

- Can be used only for end user certificates
- Key size when not in FIPS mode is between 512 and 2048 bits rounded up to a multiple of 64
- Key size in FIPS mode of 2048 bits
- Key parameters encoded as an ASN.1 sequence consisting of the prime P, the base G, the subprime Q and the subgroup factor J. Refer to RFC 2631 (Diffie-Hellman Key Agreement Method) for more information on the key parameters. The `gsk_generate_key_parameters()` routine can be used to generate the key parameters.

The record label is used as a friendly name for the database entry. It can be any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string.
A CA certificate will have SubjectKeyIdentifier, KeyUsage and BasicConstraints extensions while an end
user certificate will have SubjectKeyIdentifier and KeyUsage extensions. An AuthorityKeyIdentifier
extension will be created if the signing certificate has a SubjectKeyIdentifier extension. The application can
supply additional extensions through the extensions parameter. An AuthorityKeyIdentifier, KeyUsage or
BasicConstraints extension provided by the application will replace the default extension created for the
certificate, however a SubjectKeyIdentifier extension provided by the application will be ignored.

The database must be open for update in order to add the new certificate. The database file is updated as
part of the `gsk_create_signed_certificate_set()` processing. A temporary database file is created using
the same name as the database file with "_.new" appended to the name. The database file is then
overwritten and the temporary database file is deleted. The temporary database file will not be deleted if
an error occurs while rewriting the database file.
gsk_create_signed_crl()

Creates a signed certificate revocation list.

This function is deprecated. Use gsk_create_signed_crl_record() instead.

Format

```
#include <gskcms.h>

gsk_status gsk_create_signed_crl (gsk_handle db_handle, const char *label, gsk_int32 crl_number, int num_days, x509_revoked_certificates *revoked_certificates, x509_extensions *extensions, gsk_buffer *signed_crl)
```

Parameters

`db_handle`
Specifies the database handle returned by the gsk_create_database() routine, the gsk_open_database() routine, or the gsk_open_keyring() routine. This must be a key database and not a request database.

`label`
Specifies the label for the certificate to be used to sign the certificate revocation list. The label is specified in the local code page.

`crl_number`
Specifies the CRL number. Each CRL is numbered with each successive revocation list having a larger CRL number than all previous revocation lists.

`num_days`
Specifies the number of days until the next CRL will be issued and is specified as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified).

`revoked_certificates`
Specifies the revoked list of certificates to be included in the CRL. This list consists of the certificate serial numbers and not the actual certificates.

`extensions`
Specifies the CRL extensions for the new CRL. Specify NULL for this parameter if no CRL extensions are supplied.

`signed_crl`
Returns the signed certificate revocation list in Base64 format. The Base64 stream will be in the local code page. The application should call the gsk_free_buffer() routine to release the stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_LABEL]
The record label is not valid.
### gsk_create_signed_crl()

**[CMSERR_BAD_SIGNATURE]**
- The request signature is not correct.

**[CMSERR_DUPLICATE_EXTENSION]**
- Supplied extensions contain a duplicate extension.

**[CMSERR_EXPIRED]**
- The signer certificate is expired.

**[CMSERR_INCORRECT_DBTYPE]**
- The database type does not support certificates.

**[CMSERR_INCORRECT_KEY_USAGE]**
- The signer certificate key usage does not allow signing a CRL.

**[CMSERR_ISSUER_NOT_CA]**
- The signer certificate is not for a certification authority.

**[CMSERR_NO_MEMORY]**
- Insufficient storage is available.

**[CMSERR_NO_PRIVATE_KEY]**
- The signer certificate does not have a private key.

**[CMSERR_RECORD_NOT_FOUND]**
- The signer certificate is not found in the key database.

### Usage

The `gsk_create_signed_crl()` routine will generate an X.509 certificate revocation list (CRL) as described in RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). The new CRL will be signed using the certificate specified by the `label` parameter. The number of days until the next CRL is issued will be set to the earlier of the requested date and the expiration of the signing certificate.

The signing certificate must have an associated private key, the BasicConstraints extension must either be omitted or must have the CA indicator set, and the KeyUsage extension must either be omitted or must allow signing certificate revocation lists.

The CRL will have a CRLNumber extension containing the value specified by the `crl_number` parameter. It will also have an AuthorityKeyIdentifier extension if the signing certificate has a SubjectKeyIdentifier extension. The application can supply additional extensions through the `extensions` parameter. An AuthorityKeyIdentifier or CRLNumber extension provided by the application will replace the default extension created for the CRL.

No certification path validation is performed by the `gsk_create_signed_crl()` routine.
gsk_create_signed_crl_record()

Creates a signed certificate revocation list.

Format

```c
#include <gskcms.h>
gsk_status gsk_create_signed_crl_record (  
gsk_handle  
const char *  
x509_algorithm_type  
gsk_int32  
int  
x509_revoked_certificates *  
x509_extensions *  
gsk_buffer *  
    db_handle,  
label,  
signature_algorithm,  
crl_number,  
num_days,  
revoked_certificates,  
extensions,  
signed_crl)
```

Parameters

- **db_handle**
  Specifies the database handle returned by the `gsk_create_database()` routine, the `gsk_open_database()` routine, or the `gsk_open_keyring()` routine. This must be a key database and not a request database.

- **label**
  Specifies the label for the certificate to be used to sign the certificate revocation list. The label is specified in the local code page.

- **signature_algorithm**
  Specifies the signature algorithm to be used for the crl signature.

- **crl_number**
  Specifies the CRL number. Each CRL is numbered with each successive revocation list having a larger CRL number than all previous revocation lists.

- **num_days**
  Specifies the number of days until the next CRL will be issued and is specified as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified).

- **revoked_certificates**
  Specifies the revoked list of certificates to be included in the CRL. This list consists of the certificate serial numbers and not the actual certificates.

- **extensions**
  Specifies the CRL extensions for the new CRL. Specify NULL for this parameter if no CRL extensions are supplied.

- **signed_crl**
  Returns the signed certificate revocation list in Base64 format. The Base64 stream will be in the local code page. The application should call the `gsk_free_buffer()` routine to release the stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_ALG_NOT_SUPPORTED]**
  The signature algorithm is not supported.
gsk_create_signed_crl_record()

<table>
<thead>
<tr>
<th>CMSERR_BAD_HANDLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>The database handle is not valid.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CMSERR_BAD_KEY_SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>The key size is not valid.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CMSERR_BAD_LABEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>The record label is not valid.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CMSERR_BAD_SIGNATURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>The request signature is not correct.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CMSERR_EXPIRED</th>
</tr>
</thead>
<tbody>
<tr>
<td>The signer certificate is expired.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CMSERR_INCORRECT_DBTYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>The database type does not support certificates.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CMSERR_INCORRECT_KEY_USAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>The signer certificate key usage does not allow signing a CRL.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CMSERR_ISSUER_NOT_CA</th>
</tr>
</thead>
<tbody>
<tr>
<td>The signer certificate is not for a certification authority.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CMSERR_NO_MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insufficient storage is available.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CMSERR_NO_PRIVATE_KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>The signer certificate does not have a private key.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CMSERR_RECORD_NOT_FOUND</th>
</tr>
</thead>
<tbody>
<tr>
<td>The signer certificate is not found in the key database.</td>
</tr>
</tbody>
</table>

Usage

The `gsk_create_signed_crl_record()` routine will generate an X.509 certificate revocation list (CRL) as described in RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). The new CRL will be signed using the certificate specified by the `label` parameter and the signature algorithm specified by the `signature_algorithm` parameter.

The following signature algorithms are supported:

- `x509_alg_md2WithRsaEncryption`
  - RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

- `x509_alg_md5WithRsaEncryption`
  - RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

- `x509_alg_sha1WithRsaEncryption`
  - RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

- `x509_alg_sha224WithRsaEncryption`
  - RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

- `x509_alg_sha256WithRsaEncryption`
  - RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

- `x509_alg_sha384WithRsaEncryption`
  - RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

- `x509_alg_sha512WithRsaEncryption`
  - RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

- `x509_alg_dsaWithSha1`
  - Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}
When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

The number of days until the next CRL is issued will be set to the earlier of the requested date and the expiration of the signing certificate.

The signing certificate must have an associated private key, the BasicConstraints extension must either be omitted or must have the CA indicator set, and the KeyUsage extension must either be omitted or must allow signing certificate revocation lists.

The CRL will have a CRLNumber extension containing the value specified by the `crl_number` parameter. It will also have an AuthorityKeyIdentifier extension if the signing certificate has a SubjectKeyIdentifier extension. The application can supply additional extensions through the `extensions` parameter. An AuthorityKeyIdentifier or CRLNumber extension provided by the application will replace the default extension created for the CRL.

No certification path validation is performed by the `gsk_create_signed_crl_record()` routine.
gsk_decode_base64()

Decodes a Base64-encoded stream.

Format

```
#include <gskcms.h>

gsk_status gsk_decode_base64 (  
gsk_buffer * encoded_stream,  
gsk_buffer * decoded_stream)
```

Parameters

- **encoded_stream**
  Specifies the Base64-encoded stream. The encoded data must be in the local code page.

- **decoded_stream**
  Returns the decoded stream. The application should call the `gsk_free_buffer()` routine to release the decoded stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_BAD_BASE64_ENCODING]**
  Incorrect Base64 encoding.

- **[CMSERR_NO_MEMORY]**
  Insufficient storage is available.

Usage

The `gsk_decode_base64()` routine will decode a Base64-encoded stream created by the `gsk_encode_base64()` routine. The encoded stream must be in the local code page and must not include any header or trailer lines added by the application to identify the stream contents (such as '-----BEGIN CERTIFICATE-----' or '-----END CERTIFICATE-----'). New line characters and whitespace characters (tabs and spaces) are ignored.
gsk_decode_certificate()

Decodes an X.509 certificate.

Format

```c
#include <gskcms.h>

gsk_status gsk_decode_certificate ( gsk_buffer* stream, x509_certificate* certificate )
```

Parameters

- `stream`
  Specifies the encoded certificate.

- `certificate`
  Returns the decoded certificate information. The application should call the `gsk_free_certificate()` routine to release the decoded certificate when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. This is a possible error:

- **[ASN_NO_MEMORY]**
  Insufficient storage is available.

Usage

The `gsk_decode_certificate()` routine decodes an X.509 certificate and returns the decoded information to the application. The certificate must have been encoded as described in RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). The `derCertificate` field will contain the undecoded TBSCertificate ASN.1 sequence for use in verifying the certificate signature, the `tbsCertificate` field will contain the decoded TBSCertificate ASN.1 sequence, and the `signatureAlgorithm` and `signatureValue` fields will contain the certificate signature. The `gsk_encode_signature()` routine can be used to recreate the encoded certificate from the `x509_certificate` structure returned by the `gsk_decode_certificate()` routine.

Character strings contained in the certificate will be returned using UTF-8 encoding. The application can call `iconv()` to convert the string to a different encoding as needed.

The certificate extensions will be returned with the extension values in ASN.1 encoded format. The `gsk_decode_certificate_extension()` routine can be called to decode a particular certificate extension. This allows all of the certificate extensions to be returned even when one or more extensions cannot be processed by the System SSL runtime.
gsk_decode_certificate_extension()

Decodes an X.509 certificate extension.

Format

```c
#include <gskcms.h>

int gsk_decode_certificate_extension (
    x509_extension * encoded_extension,
    x509_decoded_extension * decoded_extension)
```

Parameters

- `encoded_extension`
  Specifies the encoded X.509 extension as returned by the `gsk_decode_certificate()` or `gsk_decode_crl()` routine.

- `decoded_extension`
  Returns the decoded extension data. The application should call the `gsk_free_decoded_extension()` routine to release the decoded extension when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- `ASN_NO_MEMORY`
  Insufficient memory is available.

- `CMSERR_EXT_NOT_SUPPORTED`
  The certificate extension is not supported.

- `CMSERR_NO_MEMORY`
  Insufficient memory is available.

Usage

The `gsk_decode_certificate()` and `gsk_decode_crl()` routines returns all of the certificate extensions in the x509_extensions structure with the extension values still in ASN.1 encoded format. The application then calls the `gsk_decode_certificate_extension()` routine to decode a specific certificate extension.

The `gsk_decode_certificate_extension()` routine returns character strings using UTF-8 encoding. If necessary, the application can call the `iconv()` routine to convert the strings to a different encoding.

These certificate extensions are supported:
- AuthorityInfoAccess
- AuthorityKeyIdentifier
- BasicConstraints
- CertificatIssuer
- CertificatePolicies
- CrlDistributionPoints
- CrlNumber
- CrlReasonCode
- DeltaCrlIndication
- ExtKeyUsage
- FreshestCRL
gsk_decode_certificate_extension()

- HoldInstructionCode
- InhibitAnyPolicy
- InvalidityDate
- IssuerAltName
- IssuingDistributionPoint
- KeyUsage
- NameConstraints
- PolicyConstraints
- PolicyMappings
- PrivateKeyUsagePeriod
- SubjectAltName
- SubjectDirectoryAttributes
- SubjectInfoAccess
- SubjectKeyIdentifier

These general name types are supported:
- DirectoryName
- DnsName
- IpAddress
- RegisteredId
- Rfc822Name
- UniformResourceIdentifier

These general name types are not supported and will be copied to the decoded extension data as an
ASN.1-encoded sequence:
- otherName
- x400Address
- ediPartyName

Refer to RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List
(CRL) Profile) for more information about the various certificate extensions.
gsk_decode_certification_request()

Decodes a PKCS #10 certification request.

Format

```c
#include <gskcms.h>

gsk_status gsk_decode_certification_request (
    gsk_buffer * stream,
    pkcs_cert_request * request)
```

Parameters

- `stream`
  Specifies the encoded certification request.

- `request`
  Returns the decoded certification request. The application should call the `gsk_free_certification_request()` routine to release the decoded certification request when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. This is a possible error:

- **[ASN_NO_MEMORY]**
  Insufficient memory is available.

Usage

The `gsk_decode_certification_request()` routine decodes a Public Key Cryptography Standards (PKCS) certification request and returns the decoded information to the application. The request must have been encoded as described in PKCS #10 (Certification Request Syntax Standard). The `derRequestInfo` field will contain the undecoded CertificationRequestInfo ASN.1 sequence for use in verifying the request signature, the `certificationRequestInfo` field will contain the decoded CertificationRequestInfo ASN.1 sequence, and the `signatureAlgorithm` and `signatureValue` fields will contain the request signature. The `gsk_encode_signature()` routine can be used to recreate the encoded certification request from the `pkcs_cert_request` structure returned by the `gsk_decode_certification_request()` routine.

Character strings contained in the request will be returned using UTF-8 encoding. If necessary, the application can call `iconv()` to convert the string to a different encoding.
gsk_decode_crl()

Decodes an X.509 certificate revocation list.

Format

```c
#include <gskcms.h>

gsk_status gsk_decode_crl (         gsk_status gsk_decode_crl (  
gsk_buffer * stream,  
x509_crl * crl)
gsk_buffer * stream,  
x509_crl * crl)
```

Parameters

- `stream`  
  Specifies the encoded certificate revocation list.

- `crl`  
  Returns the decoded information. The application should call the `gsk_free_crl()` routine to release the decoded certificate revocation list when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. This is a possible error:

- [ASN_NO_MEMORY]
  Insufficient memory is available.

Usage

The `gsk_decode_crl()` routine decodes an X.509 certificate revocation list (CRL) and returns the decoded information to the application. The CRL must have been encoded as described in RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). The `derCertList` field will contain the undecoded TBSCertList ASN.1 sequence for use in verifying the certificate signature, the `tbsCertList` field will contain the decoded TBSCertList ASN.1 sequence, and the `signatureAlgorithm` and `signatureValue` fields will contain the certificate signature. The `gsk_encode_signature()` routine can be used to recreate the encoded CRL from the `x509_crl` structure returned by the `gsk_decode_crl()` routine.

Character strings will be returned using UTF-8 encoding. If necessary, the application can call `iconv()` to convert the string to a different encoding.

The certificate extensions will be returned with the extension values in ASN.1 encoded format. The `gsk_decode_certificate_extension()` routine can be called to decode a particular certificate extension. This allows all of the certificate extensions to be returned even when one or more extensions cannot be processed by the System SSL runtime.
gsk_decode_import_certificate()

Decodes certificate from DER-encoded or PKCS #7-encoded data stream.

Format

```c
#include <gskcms.h>

gsk_status gsk_decode_import_certificate (  
gsk_buffer * stream,  
pkcs_certificate * subject_certificate,  
pkcs_certificates * issuer_certificates)
```

Parameters

* stream
  Specifies the byte stream of the encoded certificate.

* subject_certificate
  Returns the decoded certificate.

* issuer_certificates
  Returns the decoded certificate chain for the subject certificate.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_BAD_BASE64_ENCODING]**
  The Base64 encoding of the import stream is not correct.

- **[CMSERR_BAD_ENCODING]**
  The certificate request stream is not valid.

- **[CMSERR_NO_MEMORY]**
  Insufficient storage is available.

- **[CMSERR_NO_IMPORT_CERTIFICATE]**
  No certificate in import file.

Usage

The `gsk_decode_import_certificate()` function decodes a data stream into a `pkcs_certificate` structure. The `pkcs_certificate` structure `subject_certificate` returns the subject certificate, and the `pkcs_certificates` structure `issuer_certificates` returns the certificate chain for the subject certificate (all other certificates not part of the subject certificate's chain are discarded). The root certificate for the chain is the final entry in the array.

The supplied stream can represent either the ASN.1 DER encoding for the certificate or the Cryptographic Message Syntax (PKCS #7) encoding for the certificate. This can be either the binary value or the Base64 encoding of the binary value. A Base64 Encoded stream must be in the local code page and must include the encoding header and footer lines.

The `gsk_decode_import_certificate()` function decodes a single certificate. If the PKCS #7 message contains multiple certificates, only the first certificate and its certificate chain will be decoded.
gsk_decode_import_key()

Decodes certificate and key from PKCS #12-encoded data stream.

Format

```c
#include <gskcms.h>
gsk_status gsk_decode_import_key (  
gsk_buffer * stream,  
const char * password,  
pkcs_cert_key * subject_certificate,  
pkcs_certificates * issuer_certificates)
```

Parameters

stream
Specifies the byte stream of the encoded certificate.

password
Specifies the password for the import file. The password is single-byte EBCDIC in the local code page and must consist of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string.

subject_certificate
Returns the decoded certificate and key.

issuer_certificates
Returns the decoded certificate chain for the subject certificate.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The decryption algorithm is not valid.

[CMSERR_BAD_ENCODING]
The certificate request stream is not valid.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_IMPORT_CERTIFICATE]
No certificate in input stream.

[CMSERR_PW_INCORRECT]
The password is not correct.

Usage

The gsk_decode_import_key() function decodes a data stream into a pkcs_cert_key structure. The pkcs_cert_key structure subject_certificate returns the subject certificate and key, while the pkcs_certificates structure issuer_certificates returns the certificate chain for the subject certificate (all other certificates not part of the subject certificate's chain are discarded). The root certificate for the chain is the final entry in the array.

The certificate and key must have been encoded according to the Personal Information Exchange Syntax (PKCS #12). The supplied stream can be the binary ASN.1 sequence or the Base64 encoding of the ASN.1 sequence. A Base64 encoded stream is assumed to be in the local code page and must include the encoding header and footer lines.
In FIPS mode, the only supported decryption algorithm for the import file is:

- `x509_alg_pbeWithSha1And3DesCbc` - Triple DES with SHA-1 digest.
gsk_decode_name()

Decodes an X.509 name.

Format

```
#include <gskcms.h>

#define gsk_decode_name (
Gsk_status  gsk_decode_name (stream, x509_name * name)

Parameters

stream
  Specifies the ASN.1 stream for the name.

name
  Returns the decoded X.509 name. The application should release the name when it is no longer needed by calling the gsk_free_name() routine.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. This is a possible error:

[ASN_NO_MEMORY]
  Insufficient memory is available.

Usage

The gsk_decode_name() routine will decode an ASN.1 DER-encoded X.509 name. The name must have been encoded as described in RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). Character strings will be stored in UTF-8 format and the stringType field in the x509_rdn_attribute structure will be set to indicate the ASN.1 encoded string type.
gsk_decode_private_key()

Decodes a private key.

Format

```c
#include <gskcms.h>

gsk_status gsk_decode_private_key (                         
    gsk_buffer *            stream,                     
    pkcs_private_key_info * private_key)
```

Parameters

`stream`
Specifies the ASN.1 stream for the encoded private key.

`private_key`
Returns the decoded private key. The application should release the private key when it is no longer needed by calling the `gsk_free_private_key_info()` routine.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. This is a possible error:

`[ASN_NO_MEMORY]`
Insufficient memory is available.

Usage

The `gsk_decode_private_key()` routine will decode an ASN.1 DER-encoded private key. The private key must have been encoded as described in PKCS #8 (Private Key Information Syntax Standard).
gsk_decode_public_key()

Decodes a public key.

Format

```c
#include <gskcms.h>

gsk_status gsk_decode_public_key (gsk_buffer * stream, x509_public_key_info * public_key)
```

Parameters

`stream`

Specifies the ASN.1 stream for the encoded public key.

`public_key`

Returns the decoded public key. The application should release the public key when it is no longer needed by calling the `gsk_free_public_key_info()` routine.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. This is a possible error:

`[ASN_NO_MEMORY]`

Insufficient memory is available.

Usage

The `gsk_decode_public_key()` routine will decode an ASN.1 DER-encoded public key. The public key must have been encoded as described in RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile).
**gsk_delete_record()**

Deletes a record from a key or request database.

**Format**

```c
#include <gskcms.h>
gsk_status gsk_delete_record (gsk_handle db_handle, gsk_int32 record_id)
```

**Parameters**

- `db_handle`
  - Specifies the database handle return by the `gsk_create_database()` routine or the `gsk_open_database()` routine.

- `record_id`
  - Specifies the database record to be deleted.

**Results**

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_BACKUP_EXISTS]**
  - The backup file already exists.

- **[CMSERR_BAD_HANDLE]**
  - The database handle is not valid.

- **[CMSERR_IO_ERROR]**
  - Unable to write record.

- **[CMSERR_NO_MEMORY]**
  - Insufficient storage is available.

- **[CMSERR_RECORD_NOT_FOUND]**
  - Record is not found.

- **[CMSERR_SIGNED_CERTS]**
  - The database contains records signed using the certificate.

- **[CMSERR_UPDATE_NOT_ALLOWED]**
  - Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

**Usage**

The `gsk_delete_record()` routine deletes a record from a key or request database. The database must be open for update in order to delete records. The unique record identifier identifies the record to be deleted. A certificate record cannot be deleted from a key database if the database contains records that were signed using the certificate.

The database file is updated as part of the `gsk_delete_record()` processing. A temporary database file is created using the same name as the database file with "_.new" appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.
**gsk_dn_to_name()**

Converts a DN string to an X.509 name.

**Format**

```c
#include <gskcms.h>

int gsk_status gsk_dn_to_name (const char *dn, x509_name *name)
```

**Parameters**

- **dn**
  Specifies the distinguished name in the local code page.

- **name**
  Returns the X.509 name. The X.509 strings use UTF-8 encoding. The application should call the `gsk_free_name()` routine to release the name when it is no longer needed.

**Results**

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[ASN_ATTR_NOT_FOUND]**
  An attribute type is not recognized.

- **[ASN_CANT_CONVERT]**
  An encoded attribute value contains characters from the wrong character set.

- **[ASN_INVALID_VALUE]**
  An attribute value is not valid.

- **[ASN_NO_MEMORY]**
  Insufficient storage is available.

- **[ASN_WRONG_TYPE]**
  An encoded attribute value does not represent a character string.

- **[ASN_X500_NO_AVA_SEP]**
  An attribute value separator is missing.

- **[ASN_X500_OID_SYNTAX_ERROR]**
  An object identifier is not valid.

- **[ASN_X500_SYNTAX_ERROR]**
  The DN string format is not valid.

**Usage**

The `gsk_dn_to_name()` routine converts a distinguished name (DN) string to an X.509 name in accordance with RFC 2253 (UTF-8 String Representation of Distinguished Names). The input string consists of single-byte characters in the local code page. A double-byte character is represented using the escaped UTF-8 encoding of the double-byte character in the Unicode character set.

Attribute types may be specified using either attribute names or numeric object identifiers. Attribute values must represent string values.

These DN attribute names are recognized by the System SSL runtime. An error will be returned if the DN contains an unrecognized attribute name.
This is an example of a DN using attribute names and string values:
CN=Ronald Hoffman,OU=Endicott,O=IBM,C=US

This is the same DN using object identifiers and encoded string values. The encoded string values represent the ASN.1 DER encoding of the string. The System SSL runtime supports these ASN.1 string types: PRINTABLE, VISIBLE, TELETEX, IA5, UTF8, BMP, and UCS.

Individual characters can be represented using escape sequences. This is useful when the character cannot be represented in a single-byte character set. The hexadecimal value for the escape sequence is the UTF-8 encoding of the character in the Unicode character set.

<table>
<thead>
<tr>
<th>Unicode Letter Description</th>
<th>10646 code</th>
<th>UTF-8</th>
<th>Quoted</th>
</tr>
</thead>
<tbody>
<tr>
<td>LATIN CAPITAL LETTER L</td>
<td>U0000004C</td>
<td>0x4C</td>
<td>L</td>
</tr>
<tr>
<td>LATIN SMALL LETTER U</td>
<td>U00000075</td>
<td>0x75</td>
<td>u</td>
</tr>
<tr>
<td>LATIN SMALL LETTER C WITH CARON</td>
<td>U00000100</td>
<td>0xC4BD</td>
<td>\C4\8D</td>
</tr>
<tr>
<td>LATIN SMALL LETTER I</td>
<td>U00000069</td>
<td>0x69</td>
<td>i</td>
</tr>
<tr>
<td>LATIN SMALL LETTER C WITH ACUTE</td>
<td>U00000107</td>
<td>0xC487</td>
<td>\C4\87</td>
</tr>
</tbody>
</table>

SN=Lu\C4\8D\C4\87

An escape sequence can also be used for special characters which are part of the name and are not to be interpreted as delimiters. For example:
gsk_dn_to_name()

CN=L. Eagle,OU=Jones\, Dale and Mian,O=IBM,C=US
gsk_encode_base64()

Encodes binary data using Base64 encoding.

Format

```
#include <gskcms.h>

gsk_status gsk_encode_base64 (    gsk_buffer * input_data,
    gsk_buffer * encoded_data)
```

Parameters

- **input_data**: Specifies the data to be encoded.
- **encoded_data**: Returns the encoded stream in the local code page. The application should call the `gsk_free_buffer()` routine to release the encoded stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. This is a possible error:

**[CMSERR_NO_MEMORY]**

Insufficient storage is available.

Usage

The `gsk_encode_base64()` routine will encode binary data using Base64 encoding. The encoded stream will consist of printable characters in the local code page. A new line will be inserted after each group of 64 encoded characters with a final new line at the end of the encoded stream. The `gsk_decode_base64()` routine can be used to decode the data.
gsk_encode_certificate_extension()

Encodes an X.509 certificate extension.

Format

```c
#include <gskcms.h>

gsk_status gsk_encode_certificate_extension (  
    x509_decoded_extension * decoded_extension,  
    gsk_boolean critical,  
    x509_extension * encoded_extension)
```

Parameters

- `decoded_extension`: Specifies the decoded extension data.
- `critical`: Specify TRUE if this is a critical extension or FALSE if it is not a critical extension.
- `encoded_extension`: Returns the encoded X.509 extension. The application should call the `gsk_free_certificate_extension()` routine to release the extension when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- `[ASN_NO_MEMORY]` Insufficient memory is available.
- `[CMSERR_EXT_NOT_SUPPORTED]` The certificate extension is not supported.
- `[CMSERR_NO_MEMORY]` Insufficient memory is available.

Usage

The `gsk_encode_certificate_extension()` routine encodes a certificate extension and returns the encoded extension in a format that can be used as input to the `gsk_encode_certificate()` routine.

The `gsk_encode_certificate_extension()` routine assumes character strings use UTF-8 encoding. The application is responsible for providing character data in this format.

These certificate extensions are supported:
- AuthorityInfoAccess
- AuthorityKeyIdentifier
- BasicConstraints
- CertificateIssuer
- CertificatePolicies
- CrlDistributionPoints
- CrlNumber
- CrlReasonCode
- DeltaCrlIndicator
- ExtKeyUsage
These general name types are supported:
- DirectoryName
- DnsName
- IpAddress
- RegisteredId
- Rfc822Name
- UniformResourceIdentifier

Refer to RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile) for more information about the various certificate extensions.
gsk_encode_export_certificate()

Encodes an X.509 certificate into a DER or PKCS #7 data stream.

Format

```c
#include <gskcms.h>

gsk_status gsk_encode_export_certificate(
    pkcs_certificate * subject_certificate,
    pkcs_certificates * issuer_certificates,
    gskdb_export_format format,
    gsk_buffer * stream)
```

Parameters

- **subject_certificate**
  Specifies the certificate.

- **issuer_certificates**
  Specifies the certificate chain for the subject certificate.

- **format**
  Specifies the export format. These values may be specified:

  - **gskdb_export_der_binary**
    Binary ASN.1 DER-encoded

  - **gskdb_export_der_base64**
    Base64 ASN.1 DER-encoded

  - **gskdb_export_pkcs7_binary**
    Binary PKCS #7 Cryptographic Message Syntax

  - **gskdb_export_pkcs7_base64**
    Base64 PKCS #7 Cryptographic Message Syntax

- **stream**
  Returns the byte stream for the encoded certificate. The application should call the `gsk_free_buffer` function to release the storage when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_BAD_RNG_OUTPUT]**
  In FIPS mode, random bytes generation produced duplicate output.

- **[CMSERR_FMT_NOT_SUPPORTED]**
  An unsupported export file stream format is specified.

- **[CMSERR_NO_MEMORY]**
  Insufficient storage is available.

Usage

The `gsk_encode_export_certificate()` function encodes an X.509 certificate using either the ASN.1 DER encoding for the certificate or the Cryptographic Message Syntax (PKCS #7) encoding for the certificate. This can be either the binary value or the Base64 encoding of the binary value. A Base64 encoded stream will be in the local code page and will include the encoding header and footer lines.

The export data stream contains just the requested certificate when the DER format is selected. The export data stream contains the requested certificate and its certification chain when the PKCS #7 format is selected.
is selected. The certificate chain for the subject certificate is supplied via the pkcs_certificates structure
issuer_certificates with the root certificate being the final entry in the array. A partial certification chain will
be exported if the complete chain is not supplied in issuer_certificates.

\[gsk\_encode\_export\_certificate()\]
gsk_encode_export_key()

Encodes an X.509 certificate and its private key into a PKCS #12 data stream.

Format

```c
#include <gskcms.h>

gsk_status gsk_encode_export_key (  
  pkcs_cert_key * subject_certificate,  
  pkcs_certificates * issuer_certificates,  
  gskdb_export_format format,  
  x509_algorithm_type algorithm,  
  const char * password,  
  const char * nickname,  
  gsk_buffer * stream)
```

Parameters

subject_certificate
Specifies the certificate and key.

issuer_certificates
Specifies the certificate chain for the subject certificate.

format
Specifies the export format. These values may be specified:

- **gskdb_export_pkcs12v1_binary**
  Binary PKCS #12 Version 1.
- **gskdb_export_pkcs12v1_base64**
  Base64 PKCS# 12 Version 1.
- **gskdb_export_pkcs12v3_binary**
  Binary PKCS #12 Version 3.
- **gskdb_export_pkcs12v3_base64**
  Base64 PKCS #12 Version 3.

algorithm
Specifies the encryption algorithm for the export file. The strong encryption algorithms may not be available depending upon government export regulations. These values may be specified:

- **x509_alg_pbeWithSha1And40BitRc2Cbc**
  40-bit RC2 with SHA-1 digest.
- **x509_alg_pbeWithSha1And128BitRc2Cbc**
  128-bit RC2 with SHA-1 digest.
- **x509_alg_pbeWithSha1And40BitRc4**
  40-bit RC4 with SHA-1 digest.
- **x509_alg_pbeWithSha1And128BitRc4**
  128-bit RC4 with SHA-1 digest.
- **x509_alg_pbeWithSha1And3DesCbc**
  Triple DES with SHA-1 digest.

In FIPS mode, the only supported encryption algorithm for the export file is:

- **x509_alg_pbeWithSha1And3DesCbc**
  Triple DES with SHA-1 digest.
gsk_encode_export_key()

**password**
Specifies the password for the export file. The password is in the local code page and must consist of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string.

**nickname**
Specifies the nickname assigned to the exported key in the bagAttributes field for a PKCS #12 Version 1 format file. The nickname is in the local code page. It may not be an empty string. If a PKCS #12 Version 3 export file format is specified, this parameter is ignored.

**stream**
Returns the byte stream for the encoded certificate. The application should call the `gsk_free_buffer` function to release the storage when it is no longer needed.

**Results**
The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_FMT_NOT_SUPPORTED]**
  An unsupported export file format is specified.

- **[CMSERR_NO_MEMORY]**
  Insufficient storage is available.

- **[CMSERR_ALG_NOT_SUPPORTED]**
  The signature algorithm is not valid.

- **[CMSERR_NO_PRIVATE_KEY]**
  The certificate does not have a private key.

- **[CMSERR_PW_INCORRECT]**
  The password is not correct.

**Usage**
The `gsk_encode_export_key()` function encodes an X.509 certificate and its private key into a PKCS #12 data stream. The certificate chain for the subject certificate is supplied via the `pkcs_certificates` structure `issuer_certificates`, with the root certificate being the final entry in the array.

The export byte stream contains the requested certificate, its private key, and the certification chain. A partial certification chain is exported if the complete chain is not supplied in `issuer_certificates`. 
gsk_encode_export_request()

`gsk_encode_export_request()` encodes a certification renewal request as described in PKCS #10.

**Format**

```c
#include <gskcms.h>

gsk_status gsk_encode_export_request (  
    pkcs_cert_request * request,  
    gskdb_export_format format,  
    gsk_buffer * stream)
```

**Parameters**

- `request` specifies the certification renewal request.
- `format` specifies the export format. These values may be specified:
  - `gskdb_export_der_binary` Binary ASN.1 DER-encoded.
  - `gskdb_export_der_base64` Base64 ASN.1 DER-encoded.
- `stream` returns the byte stream for the encoded certification request. The application should call the `gsk_free_buffer` function to release the storage when it is no longer needed.

**Results**

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. This is a possible error:

`[CMSERR_NO_MEMORY]` Insufficient storage is available.

**Usage**

The `gsk_encode_export_request()` function exports a PKCS #10 certification request. The request can be exported using either the ASN.1 DER encoding for the request or the Base64 encoding of the binary value. A Base64 encoded stream will be in the local code page and will include the encoding header and footer lines.
gsk_encode_name()  
Encodes an X.509 name.

Format

```c
#include <gskcms.h>
gsk_status gsk_encode_name (  
x509_name * name,  
gsk_buffer * stream)
```

Parameters

name

Specifies X.509 name.

stream

Returns the ASN.1 stream for the name. The application should release the stream when it is no longer needed by calling the `gsk_free_buffer()` routine.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[ASN_CANT_CONVERT]**
  A character string contains characters not allowed for the string type.

- **[ASN_NO_MEMORY]**
  Insufficient memory is available.

Usage

The `gsk_encode_name()` routine will encode an X.509 name as an ASN.1 stream. The name will be encoded as described in RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile).

The `stringType` field in the x509_rdn_attribute structure will be used to determine the format for an encoded directory string. If it is set to `x509_string_unknown`, the `gsk_encode_name()` routine will attempt to encode the string as an ASN.1 printable string. If the string contains characters not included in the printable string set, the string will be encoded as an ASN.1 UTF-8 string. There are a couple of mandatory exceptions:

- The `countryName` attribute is always encoded as a printable string
- The `dnQualifier` attribute is always encoded as a printable string
- The `emailAddress` attribute is always encoded as an IA5 string
- The `domainComponent` attribute is always encoded as an IA5 string
gsk_encode_private_key()

Encode a private key.

Format

```
#include <gskcms.h>

#define gsk_encode_private_key (  
    gsk_status gsk_encode_private_key (  
        pkcs_private_key_info * private_key,  
        gsk_buffer * stream)
```

Parameters

- **private_key**
  - Specifies the private key.

- **stream**
  - Returns the ASN.1 stream for the private key. The application should release the stream when it is no longer needed by calling the `gsk_free_buffer()` routine.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[ASN_NO_MEMORY]**
  - Insufficient memory is available.

Usage

The `gsk_encode_private_key()` routine will encode a private key as an ASN.1 stream. The name will be encoded as described in PKCS #8 (Private Key Information Syntax Standard). The encoded private key will not be usable on another system if the private key information contains an ICSF key token.
gsk_encode_public_key()

Encode a public key.

Format

```
#include <gskcms.h>

gsk_status gsk_encode_public_key (
    x509_public_key_info * public_key,
    gsk_buffer * stream)
```

Parameters

`public_key`

Specifies the public key.

`stream`

Returns the ASN.1 stream for the public key. The application should release the stream when it is no longer needed by calling the `gsk_free_buffer()` routine.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

`[ASN_NO_MEMORY]`

Insufficient memory is available.

Usage

The `gsk_encode_public_key()` routine will encode a public key as an ASN.1 stream. The name will be encoded as described in RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile).
gsk_encode_signature()

Encodes an ASN.1 stream and the accompanying signature.

Format

```c
#include <gskcms.h>

gsk_status gsk_encode_signature (
    gsk_buffer * unsigned_stream,
    x969_algorithm_identifier * algorithm,
    gsk_bitstring * signature,
    gsk_buffer * signed_stream)
```

Parameters

- `unsigned_stream` specifies the unsigned ASN.1 stream.
- `algorithm` specifies the algorithm used to compute the signature.
- `signature` specifies the signature for the ASN.1 stream.
- `signed_stream` returns the encoded signature stream. The application should call the `gsk_free_buffer()` routine to release the encoded stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. This is a possible error:

```
[ASN_NO_MEMORY]
Insufficient memory is available.
```

Usage

The `gsk_encode_signature()` routine is used to encode an unsigned ASN.1 stream and the digital signature generated for the stream. The signature is encoded using ASN.1 DER (Distinguished Encoding Rules). The application is responsible for ensuring the validity of the supplied information.
gsk_export_certificate()

Exports a certificate.

Format

```c
#include <gskcms.h>

gsk_status gsk_export_certificate (  
gsk_handle db_handle,  
const char * label,  
gskdb_export_format format,  
gsk_buffer * stream)
```

Parameters

- `db_handle`
  Specifies the database handle returned by the `gsk_create_database()` routine, the `gsk_open_database()` routine, or the `gsk_open_keyring()` routine. The database must be a key database and not a request database.

- `label`
  Specifies the label for the database record. The label is specified in the local code page.

- `format`
  Specifies the export format. These values may be specified:

  - `gskdb_export_der_binary`
    Binary ASN.1 DER-encoded

  - `gskdb_export_der_base64`
    Base64 ASN.1 DER-encoded

  - `gskdb_export_pkcs7_binary`
    Binary PKCS #7 Cryptographic Message Syntax

  - `gskdb_export_pkcs7_base64`
    Base64 PKCS #7 Cryptographic Message Syntax

- `stream`
  Return the byte stream for the encoded certificate. The application should call the `gsk_free_buffer()` routine to release the storage when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- `[CMSERR_BAD_HANDLE]`
  The database handle is not valid.

- `[CMSERR_BAD_LABEL]`
  No database record label is supplied.

- `[CMSERR_FMT_NOT_SUPPORTED]`
  An unsupported export file format is specified.

- `[CMSERR_INCORRECT_DBTYPE]`
  The database type does not support certificates.

- `[CMSERR_NO_MEMORY]`
  Insufficient storage is available.

- `[CMSERR_RECORD_NOT_FOUND]`
  The requested record is not found.
gsk_export_certificate()

Usage

The `gsk_export_certificate()` routine exports an X.509 certificate. The certificate can be exported using either the ASN.1 DER encoding for the certificate or the Cryptographic Message Syntax (PKCS #7) encoding for the certificate. This can be either the binary value or the Base64 encoding of the binary value. A Base64 encoded stream will be in the local code page and will include the encoding header and footer lines.

The export file will contain just the requested certificate when the DER format is selected. The export file will contain the requested certificate and its certification chain when the PKCS #7 format is selected. A partial certification chain will be exported if the complete chain is not in the database.
gsk_export_certification_request()

Exports a PKCS #10 certification request.

Format

```c
#include <gskcms.h>
gsk_status gsk_export_certification_request (  
gsk_handle  db_handle,  
const char *  label,  
gskdb_export_format  format,  
gsk_buffer *  stream)
```

Parameters

`db_handle`
Specifies the database handle returned by the `gsk_create_database()` routine or the `gsk_open_database()` routine. The database must be a request database and not a key database.

`label`
Specifies the label for the database record. The label is specified in the local code page.

`format`
Specifies the export format. These values may be specified:

- `gskdb_export_der_binary`
  Binary ASN.1 DER-encoded

- `gskdb_export_der_base64`
  Base64 ASN.1 DER-encoded

`stream`
Return the byte stream for the encoded certification request. The application should call the `gsk_free_buffer()` routine to release the storage when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- `[CMSERR_BAD_HANDLE]`  
The database handle is not valid.

- `[CMSERR_BAD_LABEL]`  
No database record label is supplied.

- `[CMSERR_FMT_NOT_SUPPORTED]`  
An unsupported export file format is specified.

- `[CMSERR_INCORRECT_DBTYPE]`  
The database type does not support certification requests.

- `[CMSERR_NO_MEMORY]`
  Insufficient storage is available.

- `[CMSERR_RECORD_NOT_FOUND]`
  The requested record is not found.

Usage

The `gsk_export_certification_request()` routine exports a PKCS #10 certification request. The request can be exported using either the ASN.1 DER encoding for the request or the Base64 encoding of the binary value. A Base64 encoded stream will be in the local code page and will include the encoding
gsk_export_certification_request()

header and footer lines.
gsk_export_key()

Exports a certificate and the associated private key.

Format

```c
#include <gskcms.h>

gsk_status gsk_export_key (  
gsk_handle db_handle,  
const char * label,  
gskdb_export_format format,  
x509_algorithm_type algorithm,  
const char * password,  
gsk_buffer * stream,)
```

Parameters

db_handle

Specifies the database handle returned by the `gsk_create_database()` routine, the `gsk_open_database()` routine, or the `gsk_open_keyring()` routine. The database must be a key database and not a request database. For a SAF key ring database, the private key must be stored in the SAF database and not in ICSF.

label

Specifies the label for the database record. The label is specified in the local code page.

format

Specifies the export format. These values may be specified:

- `gskdb_export_pkcs12v1_binary`
  Binary PKCS #12 Version 1
- `gskdb_export_pkcs12v1_base64`
  Base64 PKCS #12 Version 1
- `gskdb_export_pkcs12v3_binary`
  Binary PKCS #12 Version 3
- `gskdb_export_pkcs12v3_base64`
  Base64 PKCS #12 Version 3

algorithm

Specifies the encryption algorithm for the export file. The strong encryption algorithms may not be available depending upon government export regulations.

These values may be specified for the PKCS #12 Version 1 format:

- `x509_alg_pb1WithSha1And40BitRc2Cbc`
  40-bit RC2 with SHA-1 digest
- `x509_alg_pb1WithSha1And128BitRc2Cbc`
  128-bit RC2 with SHA-1 digest
- `x509_alg_pb1WithSha1And40BitRc4`
  40-bit RC4 with SHA-1 digest
- `x509_alg_pb1WithSha1And128BitRc4`
  128-bit RC4 with SHA-1 digest
- `x509_alg_pb1WithSha1And3DesCbc`
  Triple DES with SHA-1 digest

These values may be specified for the PKCS #12 Version 3 format:
gsk_export_key()

- `x509_alg_pbeWithSha1And40BitRc2Cbc`
  - 40-bit RC2 with SHA-1 digest
- `x509_alg_pbeWithSha1And128BitRc2Cbc`
  - 128-bit RC2 with SHA-1 digest
- `x509_alg_pbeWithSha1And40BitRc4`
  - 40-bit RC4 with SHA-1 digest
- `x509_alg_pbeWithSha1And128BitRc4`
  - 128-bit RC4 with SHA-1 digest
- `x509_alg_pbeWithSha1And3DesCbc`
  - Triple DES with SHA-1 digest

In FIPS mode, there is only one supported encryption algorithm for the export file.

For PKCS#12 Version 3:
- `x509_alg_pbeWithSha1And3DesCbc`
  - Triple DES with SHA-1 digest.

**password**

- Specifies the password for the export file. The password is in the local code page and must consist of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string. The user will be prompted to enter the password if NULL is specified for this parameter.

**stream**

- Return the byte stream for the encoded certificate. The application should call the `gsk_free_buffer()` routine to release the storage when it is no longer needed.

**Results**

- The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:
  - `[CMSERR_ALG_NOT_SUPPORTED]` The encryption algorithm is not supported.
  - `[CMSERR_BAD_HANDLE]` The database handle is not valid.
  - `[CMSERR_BAD_LABEL]` The record label or CA certificate label is not valid.
  - `[CMSERR_BAD_RNG_OUTPUT]` In FIPS mode, random bytes generation produced duplicate output.
  - `[CMSERR_FMT_NOT_SUPPORTED]` An unsupported export file format is specified.
  - `[CMSERR_INCORRECT_DBTYPE]` The database type does not support certificates.
  - `[CMSERR_NO_MEMORY]` Insufficient storage is available.
  - `[CMSERR_NO_PRIVATE_KEY]` The signer certificate does not have a private key.
  - `[CMSERR_RECORD_NOT_FOUND]` The requested record is not found.
gsk_export_key()

Usage
The **gsk_export_key()** routine exports an X.509 certificate and the associated private key. The certificate can be exported using either the PKCS #12 Version 1 format or the PKCS #12 Version 3 format. This can be either the binary value or the Base64 encoding of the binary value. A Base64 encoded stream will be in the local code page and will include the encoding header and footer lines.

The PKCS #12 Version 1 format is obsolete. However, it is the only format supported by some SSL implementations and must be used when moving a certificate and key to one of those systems. If not running in FIPS mode, you should use either x509_alg_pb1WithSha1And40BitRc2Cbc or x509_alg_pb1WithSha1And3DesCbc for interoperability with these older SSL implementations.

The export file will contain the requested certificate, its private key, and the certification chain. A partial certification chain will be exported if the complete chain is not in the database.
gsk_factor_private_key_rsa()

Factorizes an RSA private key into its component values.

Format

```c
#include <gskcms.h>
gsk_status gsk_factor_private_key_rsa (    pkcs_private_key_info * private_key,
    gsk_buffer * modulus,
    gsk_buffer * public_exponent,
    gsk_buffer * private_exponent,
    gsk_buffer * prime1,
    gsk_buffer * prime2,
    gsk_buffer * prime_exponent1,
    gsk_buffer * prime_exponent2,
    gsk_buffer * coefficient)
```

Parameters

- **private_key**: Specifies the private key.
- **modulus**: Returns the modulus (n).
- **public_exponent**: Returns the public exponent (e).
- **private_exponent**: Returns the private exponent (d).
- **prime1**: Returns the 1st prime (p).
- **prime2**: Returns the 2nd prime (q).
- **prime_exponent1**: Returns the private exponent d modulo p-1.
- **prime_exponent2**: Returns the private exponent d modulo q-1.
- **coefficient**: Returns the CRT coefficient q-1 mod p.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[ASN_ELEMENTS_MISSING]**: Required data element is missing.

Usage

The `gsk_factor_private_key_rsa()` function deconstructs the `pkcs_private_key_info` into its RSA private key components.
**gsk_factor_public_key_rsa()**

Factorizes an RSA public key into its component values.

**Format**

```c
#include <gskcms.h>
gsk_status gsk_factor_public_key_rsa ( 
    x509_public_key_info * public_key, 
    gsk_uint32 * modulus_bits, 
    gsk_buffer * modulus, 
    gsk_buffer * exponent)
```

**Parameters**

- **public_key**
  - Specifies the public key.

- **modulus_bits**
  - Returns the length of the modulus in bits.

- **modulus**
  - Returns the modulus (n).

- **exponent**
  - Returns the public exponent (e).

**Results**

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[ASN_ELEMENTS_MISSING]**
  - Required data element is missing.

**Usage**

The `gsk_factor_public_key_rsa()` function deconstructs the pkcs_public_key_info into its RSA public key components.
gsk_fips_state_query()

Queries the current state of FIPS mode.

Format

```c
void gsk_fips_state_query(
    GSKitStatus *status,
    GSK_FIPS_STATE_ENUM_VALUE *enum_value);
```

Parameters

- `enum_value`
  - Returns the FIPS state enumeration value.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file.

Usage

The `gsk_fips_state_query` function returns an enumerated value indicating the current FIPS mode state of System SSL. One of the following enumerated values will be returned:

- `GSK_FIPS_STATE_NOTSET`
  - FIPS mode state has not yet been set.

- `GSK_FIPS_STATE_ON`
  - FIPS mode state has been set to FIPS mode.

- `GSK_FIPS_STATE_OFF`
  - FIPS mode state has been set to non-FIPS mode.
gsk_fips_state_set()

Sets the state of FIPS mode for System SSL.

Format

```
    gsk_status gsk_fips_state_set(
        GSK_FIPS_STATE_ENUM_VALUE * enum_value)
```

Parameters

`enum_value` Specifies the FIPS state enumeration value.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. The following are some possible errors:

- **[CMSERR_ATTRIBUTE_INVALID_ENUMERATION]**
  The enumeration value is not valid or it cannot be set due to the current state.

- **[CMSERR_FIPS_MODE_EXECUTE_FAILED]**
  The request to execute in FIPS mode failed because the Cryptographic Services Security Level 3 FMID is not installed so that the required System SSL DLLs could not be loaded.

- **[CMSERR_FIPS_MODE_SWITCH]**
  The System SSL FIPS mode state cannot be changed to FIPS mode because it is currently not in FIPS mode.

- **[CMSERR_KATPW_FAILED]**
  The power on known answer tests failed. FIPS mode cannot be set.

- **[CMSERR_SIGNVER_UNAVAIL]**
  The SAF callable service R_PgmSignVer is not installed or is unavailable. FIPS mode cannot be set.

Usage

The gsk_fips_state_set function sets the enumerated value for the System SSL FIPS mode state.

In order to set FIPS mode, this function must be executed prior to all other System SSL API functions with the exception of gsk_get_cms_vector, gsk_get_ssl_vector and gsk_fips_state_query. It is possible to switch to a non-FIPS mode at a later time. It is not possible to switch from non-FIPS mode to FIPS mode at any time.

The following enumerated value are supported:

- **GSK_FIPS_STATE_ON**
  FIPS mode state has been set to FIPS mode.

- **GSK_FIPS_STATE_OFF**
  FIPS mode state has been set to non-FIPS mode.


`gsk_free_attributes_signers()`

Releases storage allocated for gsk_attributes_signers structure.

**Format**

```c
#include <gskcms.h>

void gsk_free_attributes_signers ( gsk_attributes_signers * attributesSigners )
```

**Parameters**

- `attributesSigners`
  
  Specifies the gsk_attributes_signers structure to be released. The gsk_attributes_signers structure will be initialized to zero upon completion.

**Usage**

The `gsk_free_attributes_signers()` routine is used to release storage allocated for gsk_attributes_signers structure.
gsk_free_buffer()

Releases storage allocated for a buffer.

Format

```c
#include <gskcms.h>

void gsk_free_buffer ( gsk_buffer * buffer )
```

Parameters

- `buffer`
  Specifies the buffer to be released. The gsk_buffer structure will be initialized to zero upon completion.

Usage

The `gsk_free_buffer()` routine is used to release storage allocated for a buffer.
gsk_free_certificate()

Releases storage allocated for an X.509 certificate.

Format

```c
#include <gskcms.h>

void gsk_free_certificate (x509 certificate * certificate)
```

Parameters

certificate
Specifies the certificate to be released. The x509_certificate structure will be initialized to zero upon completion.

Usage

The `gsk_free_certificate()` routine is used to release storage allocated for an X.509 certificate.
gsk_free_certificates()
Releases storage allocated for an array of certificates.

Format

```c
#include <gskcms.h>

void gsk_free_certificates (
    pkcs_certificates * certificates)
```

Parameters

certificates
    Specifies the certificates to be released. The pkcs_certificates structure will be initialized to zero upon completion.

Usage

The **gsk_free_certificates()** routine is used to release storage allocated for an array of certificates.
gsk_free_certificate_extension()

Releases storage allocated for an X.509 certificate extension.

Format

```c
#include <gskcms.h>
void gsk_free_certificate_extension ( 
        x509_extension * extension)
```

Parameters

- **extension**
  
  Specifies the certificate extension to be released. The x509_extension structure will be initialized to zero upon completion.

Usage

The **gsk_free_certificate_extension()** routine is used to release storage allocated for an X.509 certificate extension.
gsk_free_certification_request()

Releases storage allocated for a PKCS certification request.

Format

```c
#include <gskcms.h>

void gsk_free_certification_request (pkcs_cert_request * request)
```

Parameters

- **request**
  - Specifies the certification request to be released. The pkcs_cert_request structure will be initialized to zero upon completion.

Usage

The `gsk_free_certification_request()` routine is used to release storage allocated for a Public Key Cryptography Standards (PKCS) certification request.
gsk_free_content_info() 

Releases storage allocated for PKCS #7 content information.

Format

```c
#include <gskcms.h>

void gsk_free_content_info ( pkcs_content_info * content_info )
```

Parameters

- `content_info`
  Specifies the content information to be released. The `pkcs_content_info` structure will be initialized to zero upon completion.

Usage

The `gsk_free_content_info()` routine is used to release storage allocated for a Public Key Cryptography Standards (PKCS) content information.
**gsk_free_crl()**

Releases storage allocated for an X.509 certificate revocation list.

**Format**

```c
#include <gskcms.h>

void gsk_free_crl (x509_crl *crl)
```

**Parameters**

`crl`

Specifies the certificate revocation list to be released. The x509_crl structure will be initialized to zero upon completion.

**Usage**

The **gsk_free_crl()** routine is used to release storage allocated for an X.509 certificate revocation list.
gsk_free_crls()

Releases storage allocated for an array of X.509 certificate revocation lists.

Format

```c
#include <gskcms.h>

void gsk_free_crls (
    x509_crls * crls)
```

Parameters

crls

Specifies the array of certificate revocation lists to be released. The x509_crls structure will be initialized to zero upon completion.

Usage

The `gsk_free_crls()` routine is used to release storage allocated for an array of X.509 certificate revocation lists.
gsk_free_decoded_extension()

Frees a decoded certificate extension.

Format

```c
#include <gskcms.h>

void gsk_free_decoded_extension(
    x509_decoded_extension * decoded_extension)
```

Parameters

`decoded_extension`
- Specifies the certificate extension to be released. The x509_decoded_extension structure will be initialized to zero upon completion.

Usage

The `gsk_free_decoded_extension()` routine is used to release storage allocated for a decoded X.509 certificate extension.
**gsk_free_name()**

Releases storage allocated for an X.509 name.

**Format**

```c
#include <gskcms.h>

void gsk_free_name (  
x509_name * name)
```

**Parameters**

name

Specifies the name to be released. The x509_name structure will be initialized to zero upon completion.

**Usage**

The `gsk_free_name()` routine is used to release storage allocated for an X.509 name.
gsk_free_private_key_info()

Releases storage allocated for private key information.

Format

```c
#include <gskcms.h>

void gsk_free_private_key_info (pkcs_private_key_info * info)
```

Parameters

`info`  
Specifies the private key information to be released. The pkcs_private_key_info structure will be initialized to zero upon completion.

Usage

The `gsk_free_private_key_info()` routine is used to release storage allocated for private key information.
gsk_free_public_key_info()

Releases storage allocated for public key information.

Format

```
#include <gskcms.h>

void gsk_free_public_key_info (x509_public_key_info * info)
```

Parameters

*info*

Specifies the public key information to be released. The x509_public_key_info structure will be initialized to zero upon completion.

Usage

The gsk_free_public_key_info() routine is used to release storage allocated for public key information.
gsk_free_record()

Releases storage allocated for a database record.

Format

```c
#include <gskcms.h>

void gsk_free_record (gskdb_record *record)
```

Parameters

- `record`
  Specifies the database record to be released. The gskdb_record structure is released in addition to the record data.

Usage

The `gsk_free_record()` routine is used to release storage allocated for a database record.
gsk_free_records()

Releases storage allocated for an array of database records.

Format

```c
#include <gskcms.h>

void gsk_free_records (int num_records,
                       gskdb_record **records)
```

Parameters

- **num_records**
  - Specifies the number of records in the array.

- **records**
  - Specifies the database record array to be released. The gskdb_record structures are released in addition to the record data.

Usage

The **gsk_free_records()** routine is used to release storage allocated for an array of database records.
gsk_free_string()
Releases storage allocated for a string.

Format
```c
#include <gskcms.h>

void gsk_free_string (char * string)
```

Parameters
- `string`
  Specifies the string to be released.

Usage
The `gsk_free_string()` routine is used to release storage allocated for a string.
gsk_free_strings()

Releases storage allocated for an array of strings.

Format

```
#include <gskcms.h>

void gsk_free_strings (  
    int       num_strings,
    char **   strings)
```

Parameters

- `num_strings`
  - Specifies the number of strings in the array.

- `strings`
  - Specifies the array of strings to be released.

Usage

The `gsk_free_strings()` routine is used to release storage allocated for an array of strings.
gsk_generate_key_agreement_pair()

Generates a Diffie-Hellman public/private key pair.

Format

```c
#include <gskcms.h>

gsk_status gsk_generate_key_agreement_pair (
    gsk_buffer * key_params,
    gsk_buffer * public_value,
    gsk_buffer * private_value)
```

Parameters

key_params
- Specifies the Diffie-Hellman key parameters as an ASN.1-encoded sequence.

public_value
- Returns the generated public value as a binary byte string. The application should call the `gsk_free_buffer()` routine to release the public value when it is no longer needed.

private_value
- Returns the generated private value as a binary byte string. The application should call the `gsk_free_buffer()` routine to release the private value when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_BAD_DH_PARAMS]**
  - The Diffie-Hellman group parameters are not valid.

- **[CMSERR_BAD_KEY_SIZE]**
  - The key size is not valid.

- **[CMSERR_BAD_RNG_OUTPUT]**
  - In FIPS mode, random bytes generation produced duplicate output.

- **[CMSERR_NO_MEMORY]**
  - Insufficient storage is available.

Usage

The `gsk_generate_key_agreement_pair()` routine will generate a Diffie-Hellman public/private key pair as defined in PKCS #3 (Diffie-Hellman Key Agreement Standard) and RFC 2631 (Diffie-Hellman Key Agreement Method). The required key parameters P and G and the optional key parameters Q and J are supplied as an ASN.1-encoded sequence as defined in either PKCS #3 or RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). The return values will be the binary values for Y and X. The key size is determined by the size of the modulus P and must be between 512 and 2048 bits if not executing in FIPS mode, and must be 2048 bits if executing in FIPS mode. The private value X will be less than Q-1 if Q is present in the key parameters, otherwise the private value X will be less than P-1.

Multiple Diffie-Hellman Key Agreement key pairs can share the same group parameters (P and G). This is useful when generating multiple keys of the same type since it is very time-consuming to compute values for P and G. In addition, the Diffie-Hellman key agreement algorithm requires both parties to use the same group parameters when computing the shared secret value.
gsk_generate_key_pair()

Generates a public/private key pair.

Format

```c
#include <gskcms.h>

gsk_status gsk_generate_key_pair (
    x509_algorithm_type     key_algorithm, 
    int                     key_size, 
    gsk_buffer *            key_params, 
    x509_public_key_info *  public_key, 
    pkcs_private_key_info * private_key, 
    gsk_buffer *            key_identifier
)
```

Parameters

- **key_algorithm**
  Specifies the key algorithm.

- **key_size**
  Specifies the key size in bits.

- **key_params**
  Specifies the key parameters as an ASN.1-encoded sequence. Specify NULL for this parameter if the key algorithm does not require any parameters.

- **public_key**
  Returns the generated public key. The application should call the `gsk_free_public_key_info()` routine to release the public key when it is no longer needed.

- **private_key**
  Returns the generated private key. The application should call the `gsk_free_private_key_info()` routine to release the private key when it is no longer needed.

- **key_identifier**
  Returns the key identifier for the generated public key. The application should call the `gsk_free_buffer()` routine to release the key identifier when it is no longer needed. Specify NULL for this parameter if the key identifier is not needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_ALG_NOT_SUPPORTED]**
  The key algorithm is not supported.

- **[CMSERR_BAD_DH_PARAMS]**
  The Diffie-Hellman group parameters are not valid.

- **[CMSERR_BAD_DSA_PARAMS]**
  The DSS parameters are not valid.

- **[CMSERR_BAD_KEY_SIZE]**
  The key size is not valid.

- **[CMSERR_FIPS_KEY_PAIR_CONSISTENCY]**
  FIPS mode key generation failed pair-wise consistency check.

- **[CMSERR_NO_MEMORY]**
  Insufficient storage is available.
Usage

The gsk_generate_key_pair() routine will generate a public/private key pair. The format of the public and
private key values returned by the gsk_generate_key_pair() routine is defined in RFC 3280 (Internet

These key algorithms are supported:

- **x509_alg_rsaEncryption - RSA Encryption - {1.2.840.113549.1.1.1}**
  - The key size must be between 512 and 4096 bits if not executing in FIPS mode, and must be between 1024 and 4096 bits if executing in FIPS mode, and will be rounded up to a multiple of 16 bits if necessary. No key parameters are used. The key size determines the size of the modulus N.

- **x509_alg_idDsa - Digital Signature Standard - {1.2.840.10040.4.1}**
  - The key size must be between 512 and 1024 bits if not executing in FIPS mode, and must be 1024 bits if executing in FIPS mode, and will be rounded up to a multiple of 64 bits if necessary. The key parameters are the prime P, the subprime Q, and the base G. The requested key size must be the same as the size of the prime P. The gsk_generate_key_parameters() routine can be used to generate the key parameters.

- **x509_alg_dhPublicNumber - Diffie-Hellman Key Exchange - {1.2.840.10046.2.1}**
  - The key size must be between 512 and 2048 bits if not executing in FIPS mode, and must be 2048 bits if executing in FIPS mode, and will be rounded up to a multiple of 64 bits if necessary. The key parameters are the prime P, the base G, the subprime Q, and the subgroup factor J. The requested key size must be the same as the size of the prime P. The gsk_generate_key_parameters() routine can be used to generate the key parameters.

The subprime Q and the subgroup factor J are optional key parameters. This allows the gsk_generate_key_pair() routine to accept key parameters generated in accordance with PKCS #3 (Diffie-Hellman Key Agreement Standard) as well as key parameters generated in accordance with RFC 2631 (Diffie-Hellman Key Agreement Method). The private value X will be less than Q-1 if Q is present in the key parameters, otherwise the private value X will be less than P-1.

Multiple Digital Signature Standard keys or Diffie-Hellman Key Exchange keys can share the same group parameters (P, Q, and G). This is useful when generating multiple keys of the same type since it is very time-consuming to compute values for P, Q, and G. In addition, the Diffie-Hellman key agreement algorithm requires both parties to use the same group parameters when computing the secret value.


gsk_generate_key_parameters()

Generates a key parameters.

Format

```c
#include <gskcms.h>

gsk_status gsk_generate_key_parameters (
    x509_algorithm_type key_algorithm,
    int key_size,
    gsk_buffer *key_params )
```

Parameters

- `key_algorithm` Specifies the key algorithm.
- `key_size` Specifies the key size in bits.
- `key_params` Specifies the key parameters as an ASN.1-encoded sequence. The application should call the `gsk_free_buffer()` routine to release the key parameters when they are no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- `[CMSERR_ALG_NOT_SUPPORTED]` The key algorithm is not supported.
- `[CMSERR_BAD_KEY_SIZE]` The key size is not valid.
- `[CMSERR_NO_MEMORY]` Insufficient storage is available.

Usage

The `gsk_generate_key_parameters()` routine will generate key parameters that can then be used with the `gsk_generate_key_pair()` routine to generate one or more public/private key pairs.

These key algorithms are supported:

- **x509_alg_idDsa - Digital Signature Standard - {1.2.840.10040.4.1}**
  - The key size must be between 512 and 1024 bits if not executing in FIPS mode, and must be 1024 bits if executing in FIPS mode, and will be rounded up to a multiple of 64 bits if necessary. The generated ASN.1 sequence will consist of the prime P, the subprime Q, and the base G. Refer to FIPS 186-2 (Digital Signature Standard) for more information on the generation of the key parameters.

- **x509_alg_dhPublicNumber - Diffie-Hellman Key Exchange - {1.2.840.10046.2.1}**
  - The key size must be between 512 and 2048 bits if not executing in FIPS mode, and must be 2048 bits if executing in FIPS mode, and will be rounded up to a multiple of 64 bits if necessary. The generated ASN.1 sequence will consist of the prime P, the base G, the subprime Q, and the subgroup factor J. Refer to RFC 2631 (Diffie-Hellman Key Agreement Method) for more information on the generation of the key parameters and RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile) for more information on the ASN.1 encoding.

Multiple Digital Signature Standard keys or Diffie-Hellman Key Exchange keys can share the same group parameters (P, Q, and G). This is useful when generating multiple keys of the same type since it is very time-consuming to compute values for P, Q, and G. In addition, the Diffie-Hellman key
agreement algorithm requires both parties to use the same group parameters when computing the secret value (an SSL client will generate temporary Diffie-Hellman values if the group parameters in the client certificate are not the same as the group parameters in the server certificate).
gsk_generate_random_bytes()

Generates a random byte stream.

Format

```c
#include <gskcms.h>

gsk_status gsk_generate_random_bytes ( gsk_buffer * buffer)
```

Parameters

- **buffer**
  Specifies the buffer for the random byte stream. The application is responsible for providing the buffer and setting the `length` and `data` fields appropriately.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_RNG]**
  Generate random bytes input buffer not valid.

- **[CMSERR_BAD_RNG_OUTPUT]**
  Generate random bytes produced duplicate output.

Usage

The `gsk_generate_random_bytes()` routine will generate a random byte stream. The application provides the buffer for the byte stream. The length value determines how many bytes will be generated.

The contents of the generated byte stream can be modified by setting the `GSK_RNG_ALLOW_ZERO_BYTES` environment variable. A `GSK_RNG_ALLOW_ZERO_BYTES` setting of "TRUE", "ON" or "1" will retain bytes with a zero value in the random byte stream. A setting of "FALSE", "OFF" or "0" will remove bytes with a zero value from the random byte stream. The default setting is "TRUE".

**Note:** The `GSK_RNG_ALLOW_ZERO_BYTES` environment variable is processed during System SSL initialization and is not checked afterwards.
gsk_generate_secret()

Generates the Diffie-Hellman shared secret.

Format

```c
#include <gskcms.h>
gsk_status gsk_generate_secret (gsk_buffer *key_params, gsk_buffer *public_value, gsk_buffer *private_value, gsk_buffer *secret_value);
```

Parameters

- `key_params`
  Specifies the Diffie-Hellman key parameters as an ASN.1-encoded sequence.

- `public_value`
  Specifies the public value for the partner application as a binary byte string.

- `private_value`
  Specifies the private value for the local application as a binary byte string.

- `secret_value`
  Returns the secret value as a binary byte string. The application should call the `gsk_free_buffer()` buffer() routine to release the secret value when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- `[CMSERR_BAD_DH_PARAMS]`
  The Diffie-Hellman group parameters are not valid.

- `[CMSERR_BAD_KEY_SIZE]`
  The key size is not valid.

- `[CMSERR_NO_MEMORY]`
  Insufficient storage is available.

Usage

The `gsk_generate_secret()` routine will generate the Diffie-Hellman shared secret value as defined in PKCS #3 (Diffie-Hellman Key Agreement Standard) and RFC 2631 (Diffie-Hellman Key Agreement Method). The required key parameters P and G and the optional key parameters Q and J are supplied as an ASN.1-encoded sequence as defined in either PKCS #3 or RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). The return value will be the binary value for Z. The key size is determined by the size of the modulus P and must be between 512 and 2048 bits if not executing in FIPS mode, or it must be 2048 bits if in FIPS mode.
gsk_get_certificate_algorithms()

Get the public key and certificate signature algorithms for a database record.

Format

```c
#include <gskcms.h>

gsk_status gsk_get_certificate_algorithms (  
gsk_handle * db_handle,  
const char * label,  
x509_algorithm_type * public_key_algorithm,  
x509_algorithm_type * signature_algorithm,  
x509_algorithm_type * signature_key_algorithm)
```

Parameters

- `db_handle` Specifies the database handle returned by the `gsk_create_database()` routine, the `gsk_open_database()` routine, or the `gsk_open_keyring()` routine. The database must be a key database, SAF key ring or z/OS PKCS #11 token.
- `label` Specifies the label for the database record. The label is specified in the local code page.
- `public_key_algorithm` Returns the key algorithm for the subject public key in the certificate.
- `signature_algorithm` Returns the signature algorithm used to sign the certificate.
- `signature_key_algorithm` Returns the signature key algorithm used to sign the certificate.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- `[CMSERR_BAD_HANDLE]` The database handle is not valid.
- `[CMSERR_INCORRECT_DBTYPE]` The database does not support this operation.
- `[CMSERR_MULTIPLE_LABEL]` Multiple certificates exist for label.
- `[CMSERR_NO_MEMORY]` Insufficient storage is available.
- `[CMSERR_RECORD_DELETED]` The requested record is deleted.
- `[CMSERR_RECORD_NOT_FOUND]` The request record is not found.

Usage

The `gsk_get_certificate_algorithms()` routine returns the public key algorithm, certificate signature algorithm, and signature key algorithm for the database record specified by the label parameter.
The function mask indicates the capabilities of the version of the CMS DLL. These values have been defined:

**GSKCMS_API_LVL1**
CMS functions provided as part of z/OS Version 1 Release 4 are available.

**GSKCMS_API_LVL2**
CMS functions provided as part of z/OS Version 1 Release 6 are available.

**GSKCMS_API_LVL3**
CMS functions provided as part of z/OS Version 1 Release 8 are available.

**GSKCMS_API_LVL4**
CMS functions provided as part of z/OS Version 1 Release 9 are available.

**GSKCMS_API_LVL5**
CMS functions provided as part of z/OS Version 1 Release 10 are available.

**GSKCMS_API_LVL6**
CMS functions provided as part of z/OS Version 1 Release 11 are available.
gsk_get_default_key()

gsk_get_default_key()

Gets the default key record.

Format

```
#include <gskcms.h>

gsk_status gsk_get_default_key (gsk_handle     db_handle,
                              gskdb_record **  record)
```

Parameters

- **db_handle**: Specifies the database handle returned by the `gsk_create_database()` routine, the `gsk_open_database()` routine, or the `gsk_open_keyring()` routine.
- **record**: Returns the database record. The application should call the `gsk_free_record()` routine to release the record when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_BAD_HANDLE]**: The database handle is not valid.
- **[CMSERR_INCORRECT_DBTYPE]**: The database does not support this operation.
- **[CMSERR_MULTIPLE_DEFAULT]**: Multiple keys are marked as the default.
- **[CMSERR_NO_MEMORY]**: Insufficient storage is available.
- **[CMSERR_RECORD_DELETED]**: The requested record is deleted.
- **[CMSERR_RECORD_NOT_FOUND]**: There is no default key for the database.

Usage

The `gsk_get_default_key()` routine retrieves the record for the default key. An error will be returned if there is no default key.
gsk_get_default_label()

Gets the label of the default key record.

Format

```c
#include <gskcms.h>

gsk_status gsk_get_default_label (   gsk_handle db_handle,
   char ** label)
```

Parameters

db_handle
- Specifies the database handle returned by the `gsk_create_database()` routine, the `gsk_open_database()` routine, or the `gsk_open_keyring()` routine.

label
- Returns the label of the default key record. The application should call the `gsk_free_string()` routine to release the label when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

[CMSERR_BAD_HANDLE]
- The database handle is not valid.

[CMSERR_INCORRECT_DBTYPE]
- The database does not support this operation.

[CMSERR_MULTIPLE_DEFAULT]
- Multiple keys are marked as the default.

[CMSERR_NO_MEMORY]
- Insufficient storage is available.

[CMSERR_RECORD_DELETED]
- The requested record is deleted.

[CMSERR_RECORD_NOT_FOUND]
- There is no default key for the database.

Usage

The `gsk_get_default_label()` routine returns the label of the default key record. An error will be returned if there is no default key.
gsk_get_directory_certificates()

Gets the certificates stored in the LDAP directory for the subject.

Format

```
#include <gskcms.h>

int gsk_get_directory_certificates (
    gsk_handle directory_handle, 
    x509_name *subject_name, 
    gsk_boolean ca_certificates, 
    pkcs_certificates *certificates)
```

Parameters

directory_handle
    Specifies the directory handle returned by the gsk_open_directory() routine.

subject_name
    Specifies the certificate subject.

ca_certificates
    Specify TRUE if the subject is a certification authority or FALSE if the subject is an end entity.

certificates
    Returns the certificates for the subject. The application should call the gsk_free_certificates() routine to release the certificates when they are no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

[CMSERR_BAD_HANDLE]
    The directory handle is not valid.

[CMSERR_LDAP]
    An error is detected by the LDAP runtime support.

[CMSERR_LDAP_NOT_AVAILABLE]
    The LDAP server is not available.

[CMSERR_NO_MEMORY]
    Insufficient storage is available.

[CMSERR_RECORD_NOT_FOUND]
    The requested certificate is not found.

Usage

The gsk_get_directory_certificates() routine retrieves the certificates stored in the LDAP directory for the specified subject name. The directory schema is defined by RFC 2587 (PKIX LDAPV2 Schema). The certificates are stored as attributes of the subject directory entry. Each certificate is encoded as defined by RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). The userCertificate attribute is used to retrieve end-entity certificates while the caCertificate attribute is used to retrieve certification authority certificates.

Retrieved certificates are cached so that it is not necessary to contact the LDAP server for subsequent requests for the same certificates. The cached certificates will be released when the gsk_close_directory() routine is called to close the directory handle.
gsk_get_directory_crls()

Gets the certificate revocation lists stored in the LDAP directory for the issuer.

Format

```c
#include <gskcms.h>
gsk_status gsk_get_directory_crls (
    gsk_handle directory_handle,
    x509_name * dist_point_name,
    x509_name * issuer_name,
    gsk_boolean ca_lists,
    x509_crls * crls)
```

Parameters

directory_handle
   Specifies the directory handle returned by the gsk_open_directory() routine.

dist_point_name
   Specifies the CRL distribution point name.

issuer_name
   Specifies the CRL issuer name.

ca_lists
   Specify TRUE to retrieve the revocation lists for CA certificates or FALSE to retrieve the revocation list for end entity certificates.

crls
   Returns the certificate revocation lists. The application should call the gsk_free_crls() routine to release the lists when they are no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

[CMSERR_BAD_HANDLE]
   The directory handle is not valid.

[CMSERR_LDAP]
   An error is detected by the LDAP runtime support.

[CMSERR_LDAP_NOT_AVAILABLE]
   The LDAP server is not available.

[CMSERR_NO_MEMORY]
   Insufficient storage is available.

[CMSERR_RECORD_NOT_FOUND]
   The requested CRL is not found.

Usage

The gsk_get_directory_crls() routine retrieves the certificate revocation lists (CRLs) stored in the LDAP directory for the specified issuer name. The directory schema is defined by RFC 2587 (PKIX LDAPV2 Schema). The revocation lists are stored as attributes of the issuer directory entry. Each CRL is encoded as defined by RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). The certificateRevocationList attribute is used to retrieve revocation lists for end-entity certificates while the authorityRevocationList attribute is used to retrieve revocation lists for certification authority certificates.
gsk_get_directory_crls()

The `dist_point_name` parameter specifies the CRL distribution point name. This name is used as the distinguished name for the LDAP directory entry. The `issuer_name` parameter specifies the CRL issuer name. This name must match the issuer name stored in the CRL.

Retrieved certificate revocation lists are cached so that it is not necessary to contact the LDAP server for subsequent requests for the same issuer. The cached revocation lists will be released when the `gsk_close_directory()` routine is called to close the directory handle. The cached entries will also be discarded at the end of the cache timeout specified by the GSK_CRL_CACHE_TIMEOUT environment variable (the default timeout is 24 hours).
gsk_get_directory_enum()

Gets an enumerated value from an LDAP directory.

Format
#include <gskcms.h>

```c
#include <gskcms.h>
gsk_status gsk_get_directory_enum ( gsk_handle directory_handle, GSKCMS_DIRECTORY_ENUM_ID enum_id, GSKCMS_DIRECTORY_ENUM_VALUE * enum_value)
```

Parameters

directory_handle
  Specifies an LDAP directory handle returned by gsk_open_directory().

enum_id
  Specifies the directory enumeration identifier.

enum_value
  Specifies the directory enumeration value.

Results

The function return value will be 0 (GSK_OK) if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

[CMSERR_ATTRIBUTE_INVALID_ID]
  The enumeration identifier is not valid or cannot be used with the specified handle.

[CMSERR_ATTRIBUTE_INVALID_ENUMERATION]
  The enumeration value is not valid or cannot be used with the specified enumeration ID.

[CMSERR_BAD_HANDLE]
  The handle is not valid.

Usage

The gsk_get_directory_enum() routine returns an enumerated value for an LDAP directory.

These enumeration identifiers are supported:

**GSKCMS_CRL_SECURITY_LEVEL**
  Returns the level of security set for the LDAP directory when contact is attempted between the application and an LDAP server that may contain a Certificate Revocation List (CRL).

  One of three possible settings for GSKCMS_CRL_SECURITY_LEVEL will be returned:
  - GSKCMS_CRL_SECURITY_LEVEL_LOW - Certificate validation will not fail if the LDAP server cannot be contacted.
  - GSKCMS_CRL_SECURITY_LEVEL_MEDIUM - Certificate validation requires the LDAP server to be contactable, but does not require a CRL to be defined. This is the default setting.
  - GSKCMS_CRL_SECURITY_LEVEL_HIGH - Certificate validation requires the LDAP server to be contactable, and a CRL to be defined.
gsk_get_record_by_id()

Gsk_get_record_by_id()

Gets a database record using the record identifier.

Format

```c
#include <gskcms.h>

int gsk_status gsk_get_record_by_id (
    gsk_handle db_handle,
    gsk_int32 record_id,
    gskdb_record ** record)
```

Parameters

- `db_handle`
  Specifies the database handle returned by the `gsk_create_database()` routine, the `gsk_open_database()` routine, or the `gsk_open_keyring()` routine.

- `record_id`
  Specifies the record identifier.

- `record`
  Returns the database record. The application should call the `gsk_free_record()` routine to release the record when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_BAD_HANDLE]**
  The database handle is not valid.

- **[CMSERR_NO_MEMORY]**
  Insufficient storage is available.

- **[CMSERR_RECORD_NOT_FOUND]**
  The requested record is not found.

Usage

The `gsk_get_record_by_id()` routine retrieves a record from a key or request database based upon the unique record identifier. The record identifier is assigned when the record is added to the database and does not change as records are added and deleted.
gsk_get_record_by_index()

Gets a database record using a sequential index.

Format

```c
#include <gskcms.h>
gsk_status gsk_get_record_by_index (gsk_handle db_handle, int index, gskdb_record **record)
```

Parameters

- `db_handle`
  Specifies the database handle returned by the `gsk_create_database()` routine, the `gsk_open_database()` routine, or the `gsk_open_keyring()` routine.

- `index`
  Specifies the sequential index of the record. The first record in the database is record 1.

- `record`
  Returns the database record. The application should call the `gsk_free_record()` routine to release the record when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- [CMSERR_BAD_HANDLE]
  The database handle is not valid.

- [CMSERR_NO_MEMORY]
  Insufficient storage is available.

- [CMSERR_RECORD_NOT_FOUND]
  The requested record is not found.

Usage

The `gsk_get_record_by_index()` routine retrieves a record from a key or request database based upon a sequential index number. The first record in the database is record 1. The index numbers will change as records are added and deleted.
gsk_get_record_by_label()

`gsk_get_record_by_label()`

Gets a database record using the record label.

**Format**

```c
#include <gskcms.h>

#include <gskcms.h>
gsk_status gsk_get_record_by_label (gsk_handle db_handle, const char * label, gskdb_record ** record)
```

**Parameters**

- **db_handle**
  Specifies the database handle returned by the `gsk_create_database()` routine, the `gsk_open_database()` routine, or the `gsk_open_keyring()` routine.

- **label**
  Specifies the label of the database record. The label is specified in the local code page.

- **record**
  Returns the database record. The application should call the `gsk_free_record()` routine to release the record when it is no longer needed.

**Results**

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_BAD_HANDLE]**
  The database handle is not valid.

- **[CMSERR_BAD_LABEL]**
  No label specified.

- **[CMSERR_MULTIPLE_LABEL]**
  Multiple certificates exist for label.

- **[CMSERR_NO_MEMORY]**
  Insufficient storage is available.

- **[CMSERR_RECORD_NOT_FOUND]**
  The requested record is not found.

**Usage**

The `gsk_get_record_by_label()` routine retrieves a record from a key or request database based upon the record label. The record label is a character string assigned when the record is added to the database. The label comparison is case sensitive.
gsk_get_record_by_subject()

Gets one or more database records using the certificate subject.

Format

```c
#include <gskcms.h>
gsk_status gsk_get_record_by_subject (gsk_handle db_handle, x509_name * name, int * num_records, gskdb_record *** records)
```

Parameters

- **db_handle**
  - Specifies the database handle returned by the `gsk_create_database()` routine, the `gsk_open_database()` routine, or the `gsk_open_keyring()` routine.

- **name**
  - Specifies the certificate subject.

- **num_records**
  - Returns the number of records in the array.

- **records**
  - Returns the array of database records. The application should call the `gsk_free_records()` routine to release the array when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_BAD_HANDLE]**
  - The database handle is not valid.

- **[CMSERR_INCORRECT_DBTYPE]**
  - The database does not support this operation.

- **[CMSERR_NO_MEMORY]**
  - Insufficient storage is available.

- **[CMSERR_RECORD_NOT_FOUND]**
  - The requested record is not found.

Usage

The `gsk_get_record_by_subject()` routine retrieves all records from a key database with the specified subject name.
gsk_get_record_labels()

gsk_get_record_labels()

Gets the record labels for a key or request database.

Format

```c
#include <gskcms.h>

gsk_status gsk_get_record_labels (  
  gsk_handle  db_handle,  
  gsk_boolean private_key,  
  int * num_labels,  
  char *** labels)
```

Parameters

`db_handle`
Specify the database handle returned by the `gsk_create_database()` routine, the `gsk_open_database()` routine, or the `gsk_open_keyring()` routine.

`private_key`
Specify TRUE if labels for records containing a private key are to be returned. Specify FALSE if labels for records without a private key are to be returned.

`num_labels`
Returns the number of record labels.

`labels`
Returns an array of string addresses. The labels are returned using the local code page. The application should call the `gsk_free_strings()` routine to release the record labels when they are no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

**[CMSERR_BAD_HANDLE]**
The database handle is not valid.

**[CMSERR_NO_MEMORY]**
Insufficient storage is available.

Usage

The `gsk_get_record_labels()` routine returns all of the record labels for a key or request database. The `gsk_get_record_by_label()` routine can then be used to retrieve a specific database record. The array address will be set to NULL and the number of labels will be set to 0 if there are no records in the database.
gsk_get_update_code()

Gets the database update code.

Format

```c
#include <gskcms.h>

gsk_status gsk_get_update_code (gsk_handle db_handle, gsk_uint32 * update_code)
```

Parameters

- **db_handle**: Specifies the database handle returned by the `gsk_create_database()` routine, the `gsk_open_database()` routine, or the `gsk_open_keyring()` routine.

- **update_code**: Returns the current update code for the database.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_BAD_HANDLE]**
  - The database handle is not valid.

- **[CMSERR_NO_MEMORY]**
  - Insufficient storage is available.

Usage

The `gsk_get_update_code()` routine returns the current update code for the database. For a file-based database or z/OS PKCS #11 token, this is the modification timestamp. For a SAF key ring, this is the ring sequence number. If an update has occurred, the application can close and then re-open the database to pick up the updates.
gsk_import_certificate()

Imports a certificate.

Format

```c
#include <gskcms.h>

gsk_status gsk_import_certificate (  
  gsk_handle  db_handle,  
  const char * label,  
  gsk_buffer * stream)
```

Parameters

- **db_handle**
  Specifies the database handle returned by the `gsk_create_database()` routine or the `gsk_open_database()` routine.

- **label**
  Specifies the label for the new database record. The label is specified in the local code page.

- **stream**
  Specifies the byte stream of the encoded certificate.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gscms.h` include file. These are some possible errors:

- **[CMSERR_ALG_NOT_SUPPORTED]**
  The key algorithm or signature algorithm is not supported.

- **[CMSERR_BAD_KEY_SIZE]**
  The algorithm key size is not valid.

- **[CMSERR_BACKUP_EXISTS]**
  The backup file already exists.

- **[CMSERR_BAD_BASE64_ENCODING]**
  The Base64 encoding of the import file is not correct.

- **[CMSERR_BAD_ENCODING]**
  The import file format is not recognized.

- **[CMSERR_BAD_HANDLE]**
  The database handle is not valid.

- **[CMSERR_BAD_LABEL]**
  The record label is not valid.

- **[CMSERR_BAD_SIGNATURE]**
  The certificate signature is not correct.

- **[CMSERR_DUPLICATE_CERTIFICATE]**
  The database already contains the certificate.

- **[CMSERR_EXPIRED]**
  The certificate is expired.

- **[CMSERR_INCORRECT_DBTYPE]**
  The database type does not support certificates.

- **[CMSERR_INCORRECT_KEY_USAGE]**
  The issuer certificate does not allow signing certificates.
[CMSERR_ISSUER_NOT_CA]
The certificate issuer is not a certification authority.

[CMSERR_ISSUER_NOT_FOUND]
The issuer certificate is not in the key database.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NOT_YET_VALID]
The certificate is not yet valid.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage
The gsk_import_certificate() routine imports an X.509 certificate and creates a new database record. An error will be returned if the certificate is already in the database. The database must be a key database and must be open for update in order to import certificates.

The supplied stream can represent either the ASN.1 DER encoding for the certificate or the Cryptographic Message Syntax (PKCS #7) encoding for the certificate. This can be either the binary value or the Base64 encoding of the binary value. A Base64 encoded stream must be in the local code page and must include the encoding header and footer lines.

The gsk_import_certificate() routine imports a single certificate. If the PKCS #7 message contains multiple certificates, only the first certificate and its certificate chain will be imported. The certificate subject name will be used as the label for certificates added from the certification chain. A chain certificate will not be added to the database if the label is not unique or if the certificate is already in the database.

A unique record identifier is assigned when the record is added to the database. The certificate signature will be verified using the certificate of the issuer. An error will be returned if the issuer certificate is not already in the key database and is not contained in the PKCS #7 message stream. The certificate will be marked as a trusted certificate when it is added to the database.

The record label is used as a friendly name for the database entry. It can be any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string.

An existing certificate can be replaced by specifying the label of the existing certificate. The issuer name, subject name, and subject public key in the new certificate must be the same as the existing certificate. If the existing certificate has a private key, the private key is not changed when the certificate is replaced.

The database file is updated as part of the gsk_import_certificate() processing. A temporary database file is created using the same name as the database file with "new" appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.
gsk_import_key()

Imports a certificate and associated private key.

Format

```c
#include <gskcms.h>

gsk_status gsk_import_key(
    gsk_handle   db_handle,
    const char * label,
    const char * password,
    gsk_buffer * stream)
```

Parameters

- **db_handle**
  Specifies the database handle returned by the `gsk_create_database()` routine or the `gsk_open_database()` routine.

- **label**
  Specifies the label for the new database record. The label is specified in the local code page.

- **password**
  Specifies the password for the import file. The password is in the local code page and must consist of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string. The user will be prompted to enter the password if NULL is specified for this parameter.

- **stream**
  Specifies the byte stream for the encoded certificate and private key.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_ALG_NOT_SUPPORTED]**
  The key algorithm or signature algorithm is not supported.

- **[CMSERR_BACKUP_EXISTS]**
  The backup file already exists.

- **[CMSERR_BAD_BASE64_ENCODING]**
  The Base64 encoding of the import file is not correct.

- **[CMSERR_BAD_ENCODING]**
  The import file format is not recognized.

- **[CMSERR_BAD_HANDLE]**
  The database handle is not valid.

- **[CMSERR_BAD_KEY_SIZE]**
  The key size is not valid.

- **[CMSERR_BAD_LABEL]**
  The record label is not valid.

- **[CMSERR_BAD_SIGNATURE]**
  The certificate signature is not correct.

- **[CMSERR_DUPLICATE_CERTIFICATE]**
  The database already contains the certificate.
The certificate is expired.

The database type does not support certificates.

The issuer certificate does not allow signing certificates.

The certificate issuer is not a certification authority.

The issuer certificate is not in the key database.

Unable to write record.

The record label is not unique.

Insufficient storage is available.

The certificate is not yet valid.

The record is larger than the database record length.

Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The `gsk_import_key()` routine imports an X.509 certificate and its private key and creates a new database record. An error will be returned if the database already contains the certificate. The database must be open for update in order to import certificates.

The certificate and key must have been encoded according to the Personal Information Exchange Syntax (PKCS #12). If executing in FIPS mode, the only supported encryption is the x509_alg_pbeWithSha1And3DesCbc algorithm. The supplied stream can be the binary ASN.1 sequence or the Base64 encoding of the ASN.1 sequence. A Base64 encoded stream is assumed to be in the local code page and must include the encoding header and footer lines.

The record label is used as a friendly name for the database entry. It can be any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string. An error will be returned if the certificate already exists in the key database or the record label is not unique.

A unique record identifier is assigned when the record is added to the database. The certificate signature will be verified using the certificate of the issuer. The certificate will be marked as a trusted certificate when it is added to the database.

Each certificate in the certification chain will be imported if it is present in the import file. The certificate subject name will be used as the label for certificates added from the certification chain. A chain certificate will not be added to the database if the label is not unique or if the certificate is already in the database.

The database file is updated as part of the `gsk_import_key()` processing. A temporary database file is created using the same name as the database file with ".new" appended to the name. The database file is
\texttt{gsk\_import\_key()}

then overwritten and the temporary database file is deleted. The temporary database file will not be
deleted if an error occurs while rewriting the database file.
gsk_make_content_msg()

Creates a PKCS #7 content information message.

Format

```c
#include <gskcms.h>
gsk_status gsk_make_content_msg (
    pkcs_content_info * content_info,
    gsk_buffer * stream)
```

Parameters

- **content_info**
  Specifies the content information for the message.

- **stream**
  Returns the ASN.1 DER-encoded stream. The application should call the `gsk_free_buffer()` routine to release the stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- [CMSERR_CONTENT_NOT_SUPPORTED]
  The content type is not supported

- [CMSERR_NO_MEMORY]
  Insufficient storage is available

Usage

The `gsk_make_content_msg()` routine creates a PKCS #7 (Cryptographic Message Syntax) message using the supplied content information and returns the ASN.1 DER-encoded ContentInfo sequence. The message content type can be any of the types defined by the PKCS #7 specification. The `gsk_read_content_msg()` routine can be used to extract the content information from the stream.
**gsk_make_data_content()**

Creates PKCS #7 Data content information from application data.

**Format**

```c
#include <gskcms.h>
gsk_status gsk_make_data_content (
    gsk_buffer * data,
    pkcs_content_info * content_info)
```

**Parameters**

- `data`: Specifies the application data.
- `content_info`: Returns the Data content information. The application should call the `gsk_free_content_info()` routine to release the content information when it is no longer needed.

**Results**

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_NO_CONTENT_DATA]**
  - The application data length is zero
- **[CMSERR_NO_MEMORY]**
  - Insufficient storage is available

**Usage**

The `gsk_make_data_content()` routine creates PKCS #7 (Cryptographic Message Syntax) Data content information. The `gsk_read_data_content()` routine can be used to extract the application data from the content information.
gsk_make_data_msg()

Creates a PKCS #7 Data message from application data.

Format

```c
#include <gskcms.h>

gsk_status gsk_make_data_msg (  
gsk_buffer * data,  
gsk_buffer * stream)
```

Parameters

data
  Specifies the application data.

stream
  Returns the ASN.1 DER-encoded stream. The application should call the gsk_free_buffer() routine to release the stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

[CMSERR_NO_CONTENT_DATA]
  The application data length is zero

[CMSERR_NO_MEMORY]
  Insufficient storage is available

Usage

The gsk_make_data_msg() routine creates a PKCS #7 (Cryptographic Message Syntax) Data message and returns the ASN.1 DER-encoded ContentInfo sequence. The message content type will be Data. The gsk_read_data_msg() routine can be used to extract the application data from the stream.

Calling the gsk_make_data_msg() routine is equivalent to calling the gsk_make_data_content() routine followed by the gsk_make_content_msg() routine.
gsk_make_encrypted_data_content()

Creates PKCS #7 EncryptedData content information.

Format

#include <gskcms.h>

int gsk_make_encrypted_data_content (int version, x509_algorithm_type pbe_algorithm, const char * password, int iterations, pkcs_content_info * content_data, pkcs_content_info * content_info)

Parameters

version
- Specifies the PKCS #7 EncryptedData version number. This must be 0.

pbe_algorithm
- Specifies the password-based encryption algorithm.

password
- Specifies the encryption password as a null-terminated string in the local code page. The user will be prompted to enter the password if NULL is specified for this parameter.

iterations
- Specifies the number of iterations used to derive the encryption key from the password. It is recommended that iterations be specified as 1024 or greater.

content_data
- Specifies the EncryptedData content. This must be one of the content information types defined in PKCS #7.

content_info
- Returns the EncryptedData content information. The application should call the gsk_free_content_info() routine to release the content information when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

[CMSERR_ALG_NOT_AVAILABLE]
- Encryption algorithm is not available

[CMSERR_ALG_NOT_SUPPORTED]
- Encryption algorithm is not supported

[CMSERR_API_NOT_SUPPORTED]
- The API is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
- The content type is not supported

[CMSERR_NO_CONTENT_DATA]
- The content data length is zero

[CMSERR_NO_MEMORY]
- Insufficient storage is available
gsk_make_encrypted_data_content()

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid

Usage
The `gsk_make_encrypted_data_content()` routine creates PKCS #7 (Cryptographic Message Syntax) EncryptedData content information. The data content type must be one of the types defined by PKCS #7. The `gsk_read_encrypted_data_content()` routine can be used to extract the content data from the content information.

- `gsk_make_encrypted_data_content()` is not supported when executing in FIPS mode and will return CMSERR_API_NOT_SUPPORTED.

The encryption key is derived from the password as described in PKCS #5 (Password-based Encryption) and PKCS #12 (Personal Information Exchange). The selected algorithm determines how the key is derived from the password.

These password-based encryption algorithms are supported. The strong encryption algorithms may not be available depending upon government export regulations.

- `x509_alg_pbeWithMd2AndDesCbc` - 56-bit DES encryption with MD2 digest - `{1.2.840.113549.1.5.1}`
- `x509_alg_pbeWithMd5AndDesCbc` - 56-bit DES encryption with MD5 digest - `{1.2.840.113549.1.5.3}`
- `x509_alg_pbeWithSha1AndDesCbc` - 56-bit DES encryption with SHA-1 digest - `{1.2.840.113549.1.5.10}`
- `x509_alg_pbeWithMd2AndRc2Cbc` - 64-bit RC2 encryption with MD2 digest - `{1.2.840.113549.1.5.4}`
- `x509_alg_pbeWithMd5AndRc2Cbc` - 64-bit RC2 encryption with MD5 digest - `{1.2.840.113549.1.5.6}`
- `x509_alg_pbeWithSha1AndRc2Cbc` - 64-bit RC2 encryption with SHA-1 digest - `{1.2.840.113549.1.5.11}`
- `x509_alg_pbeWithSha1And40BitRc2Cbc` - 40-bit RC2 encryption with SHA-1 digest - `{1.2.840.113549.1.12.1.6}`
- `x509_alg_pbeWithSha1And128BitRc2Cbc` - 128-bit RC2 encryption with SHA-1 digest - `{1.2.840.113549.1.12.1.5}`
- `x509_alg_pbeWithSha1And40BitRc4` - 40-bit RC4 encryption with SHA-1 digest - `{1.2.840.113549.1.12.1.2}`
- `x509_alg_pbeWithSha1And128BitRc4` - 128-bit RC4 encryption with SHA-1 digest - `{1.2.840.113549.1.12.1.1}`
- `x509_alg_pbeWithSha1And3DesCbc` - 168-bit 3DES encryption with SHA-1 digest - `{1.2.840.113549.1.12.1.3}`
gsk_make_encrypted_data_msg()

Creates a PKCS #7 EncryptedData message from application data.

**Format**

```c
#include <gskcms.h>
gsk_status gsk_make_encrypted_data_msg (  
    int version,  
    x509_algorithm_type pbe_algorithm,  
    const char * password,  
    int iterations,  
    gsk_buffer * data,  
    gsk_buffer * stream)
```

**Parameters**

- `version`
  - Specifies the PKCS #7 EncryptedData version number. This must be 0.

- `pbe_algorithm`
  - Specifies the password-based encryption algorithm.

- `password`
  - Specifies the encryption password as a null-terminated string in the local code page. The user will be prompted to enter the password if NULL is specified for this parameter.

- `iterations`
  - Specifies the number of iterations used to derive the encryption key from the password. It is recommended that iterations be specified as 1024 or greater.

- `data`
  - Specifies the application data for the EncryptedData message.

- `stream`
  - Returns the ASN.1 DER-encoded stream. The application should call the `gsk_free_buffer()` routine to release the stream when it is no longer needed.

**Results**

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- `[CMSERR_ALG_NOT_AVAILABLE]` Encryption algorithm is not available
- `[CMSERR_ALG_NOT_SUPPORTED]` Encryption algorithm is not supported
- `[CMSERR_API_NOT_SUPPORTED]` The API is not supported.
- `[CMSERR_CONTENT_NOT_SUPPORTED]` The content type is not supported
- `[CMSERR_NO_CONTENT_DATA]` The content data length is zero
- `[CMSERR_NO_MEMORY]` Insufficient storage is available
- `[CMSERR_VERSION_NOT_SUPPORTED]` The version is not valid
gsk_make_encrypted_data_msg()

Usage

The gsk_make_encrypted_data_msg() routine creates a PKCS #7 (Cryptographic Message Syntax) EncryptedData message and returns the ASN.1 DER-encoded ContentInfo sequence. The encrypted data content type will be Data. The gsk_read_encrypted_data_msg() routine can be used to extract the application data from the stream.

Calling the gsk_make_encrypted_data_msg() routine is equivalent to calling the gsk_make_data_content() routine, the gsk_make_encrypted_data_content() routine, and the gsk_make_content_msg() routine.

The encryption key is derived from the password as described in PKCS #5 (Password-based Encryption) and PKCS #12 (Personal Information Exchange). The selected algorithm determines how the key is derived from the password.

These password-based encryption algorithms are supported. The strong encryption algorithms may not be available depending upon government export regulations.

- x509_alg_pbeWithMd2AndDesCbc - 56-bit DES encryption with MD2 digest - {1.2.840.113549.1.5.1}
- x509_alg_pbeWithMd5AndDesCbc - 56-bit DES encryption with MD5 digest - {1.2.840.113549.1.5.3}
- x509_alg_pbeWithSha1AndDesCbc - 56-bit DES encryption with SHA-1 digest - {1.2.840.113549.1.5.10}
- x509_alg_pbeWithMd2AndRc2Cbc - 64-bit RC2 encryption with MD2 digest - {1.2.840.113549.1.5.4}
- x509_alg_pbeWithMd5AndRc2Cbc - 64-bit RC2 encryption with MD5 digest - {1.2.840.113549.1.5.6}
- x509_alg_pbeWithSha1AndRc2Cbc - 64-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.5.11}
- x509_alg_pbeWithSha1And40BitRc2Cbc - 40-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.6}
- x509_alg_pbeWithSha1And128BitRc2Cbc - 128-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.5}
- x509_alg_pbeWithSha1And40BitRc4 - 40-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.2}
- x509_alg_pbeWithSha1And128BitRc4 - 128-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.1}
- x509_alg_pbeWithSha1And3DesCbc - 168-bit 3DES encryption with SHA-1 digest - {1.2.840.113549.1.12.1.3}
gsk_make_enveloped_data_content()

Create PKCS #7 EnvelopedData content information

Format

#include <gskcms.h>

gsk_status gsk_make_enveloped_data_content ( 
    int version, 
    pkcs_session_key * session_key, 
    pkcs_certificates * recipient_certificates, 
    pkcs_content_info * content_data, 
    pkcs_content_info * content_info)

Parameters

version
Specifies the PKCS #7 EnvelopedData version number. Specify 0 to create EnvelopedData content as described in PKCS #7 Version 1.5. Specify 1 to create EnvelopedData content as described in PKCS #7 Version 1.6.

d(session_key)
Specifies the session encryption key as follows:
- The encryptionType field specifies the encryption algorithm.
- The encryptionKey.length field specifies the encryption key length in bytes.
- The encryptionKey.data field specifies the address of the encryption key. A new key will be generated and returned in this parameter if the key address is NULL. If a new key is generated, the application should call the gsk_free_buffer() routine to release the key when it is no longer needed. Note that the encryptionType and encryptionKey.length fields must be set by the application even when a new session key is to be generated.

recipient_certificates
Specifies the certificates for the message recipients. There must be at least one recipient.

content_data
Specifies the EnvelopedData content. This must be one of the content information types defined in PKCS #7.

content_info
Returns the EnvelopedData content information. The application should call the gsk_free_content_info() routine to release the content information when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

[CMSERR_ALG_NOT_AVAILABLE]
The encryption algorithm is not available

[CMSERR_ALG_NOT_SUPPORTED]
The encryption algorithm is not supported

[CMSERR_BAD_KEY_SIZE]
The encryption key size is not supported

[CMSERR_BAD_RNG_OUTPUT]
In FIPS mode, random bytes generation produced duplicate output.
gsk_make_enveloped_data_content()

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported

[CMSERR_INCORRECT_KEY_USAGE]
A recipient certificate does not allow key encryption

[CMSERR_KEY_MISMATCH]
A recipient public key does not support data encryption

[CMSERR_NO_CONTENT_DATA]
The content data length is zero

[CMSERR_NO_MEMORY]
Insufficient storage is available

[CMSERR_RECIPIENT_NOT_FOUND]
No recipient certificates provided

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid

Usage
The gsk_make_enveloped_data_content() routine creates PKCS #7 (Cryptographic Message Syntax) EnvelopedData content information. The data content type must be one of the types defined by PKCS #7. The gsk_read_enveloped_data_content() routine can be used to extract the content data from the EnvelopedData content information. No validity checking is performed on the recipient certificates. It is assumed that the application has already validated the recipient certificates.

The session key is used to encrypt the message content. A new session key is generated and returned to the application if no key is provided. For each recipient, the session key is encrypted with the recipient’s public key and stored in the EnvelopedData message. This means the public key algorithm must support data encryption. Currently, only RSA public keys support data encryption. In addition, the certificate key usage must allow key encryption.

These encryption algorithms are supported. Strong encryption may not be available depending upon government export regulations.

- x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - Key lengths 5 and 16 - {1.2.840.113549.3.2}
- x509_alg_rc4 - 40-bit and 128-bit RC4 - Key lengths 5 and 16 - {1.2.840.113549.3.4}
- x509_alg_desCbcPad - 56-bit DES - Key length 8 - {1.3.14.3.2.7}
- x509_alg_desEde3CbcPad - 168-bit 3DES - Key length 24 - {1.2.840.113549.3.7}

When executing in FIPS mode, encryption algorithms x509_alg_rc2CbcPad, x509_alg_rc4 and x509_alg_desCbcPad are not supported.
gsk_make_enveloped_data_content_extended()

Create PKCS #7 EnvelopedData content information

Format

```c
#include <gskcms.h>
gsk_status gsk_make_enveloped_data_content_extended (    gsk_process_option option_flag,    int version,    pkcs_session_key * session_key,    pkcs_certificates * recipient_certificates,    pkcs_content_info * content_data,    pkcs_content_info * content_info)
```

Parameters

- **option_flag**
  - Specifies process options to customize process behavior:
    - Enforce recipient certificate has key encryption capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate keyEncryption.

- **version**
  - Specifies the PKCS #7 EnvelopedData version number. Specify 0 to create EnvelopedData content as described in PKCS #7 Version 1.5. Specify 1 to create EnvelopedData content as described in PKCS #7 Version 1.6.

- **session_key**
  - Specifies the session encryption key as follows:
    - The `encryptionType` field specifies the encryption algorithm.
    - The `encryptionKey.length` field specifies the encryption key length in bytes.
    - The `encryptionKey.data` field specifies the address of the encryption key. A new key will be generated and returned in this parameter if the key address is NULL. If a new key is generated, the application should call the `gsk_free_buffer()` routine to release the key when it is no longer needed. Note that the `encryptionType` and `encryptionKey.length` fields must be set by the application even when a new session key is to be generated.

- **recipient_certificates**
  - Specifies the certificates for the message recipients. There must be at least one recipient.

- **content_data**
  - Specifies the EnvelopedData content. This must be one of the content information types defined in PKCS #7.

- **content_info**
  - Returns the EnvelopedData content information. The application should call the `gsk_free_content_info()` routine to release the content information when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_ALG_NOT_AVAILABLE]**
  - The encryption algorithm is not available

- **[CMSERR_ALG_NOT_SUPPORTED]**
  - The encryption algorithm is not supported
Usage

The `gsk_make_enveloped_data_content_extended()` routine creates PKCS #7 (Cryptographic Message Syntax) EnvelopedData content information. Processing is equivalent to `gsk_make_enveloped_data_content()`, except that the recipient certificate key usage need not assert key encipherment. The data content type must be one of the types defined by PKCS #7. The `gsk_make_enveloped_data_content()` routine can be used to extract the content data from the EnvelopedData content information. No validity checking is performed on the recipient certificates. It is assumed that the application has already validated the recipient certificates.

The session key is used to encrypt the message content. A new session key is generated and returned to the application if no key is provided. For each recipient, the session key is encrypted with the recipient's public key and stored in the EnvelopedData message. This means the public key algorithm must support data encryption. Currently, only RSA public keys support data encryption. In addition, if option_flag specifies that key encipherment is to be enforced, then the certificate key usage must allow key encipherment.

These encryption algorithms are supported. Strong encryption may not be available depending upon government export regulations.

- `x509_alg_rc2CbcPad` - 40-bit and 128-bit RC2 - Key lengths 5 and 16 - `{1.2.840.113549.3.2}
- `x509_alg_rc4` - 40-bit and 128-bit RC4 - Key lengths 5 and 16 - `{1.2.840.113549.3.4}
- `x509_alg_desCbcPad` - 56-bit DES - Key length 8 - `{1.3.14.3.2.7}
- `x509_alg_desEde3CbcPad` - 168-bit 3DES - Key length 24 - `{1.2.840.113549.3.7}

When executing in FIPS mode, encryption algorithms `x509_alg_rc2CbcPad`, `x509_alg_rc4` and `x509_alg_desCbcPad` are not supported.
gsk_make_enveloped_data_msg()

Creates a PKCS #7 EnvelopedData message from application data.

Format

```c
#include <gskcms.h>

int version,
pkcs_session_key * session_key,
pkcs_certificates * recipient_certificates,
gsk_buffer * data,
gsk_buffer * stream)
```

Parameters

**version**
Specifies the PKCS #7 EnvelopedData version number. Specify 0 to create an EnvelopedData message as described in PKCS #7 Version 1.5. Specify 1 to create an EnvelopedData message as described in PKCS #7 Version 1.6.

**session_key**
Specifies the session encryption key as follows:

- The `encryptionType` field specifies the encryption algorithm.
- The `encryptionKey.length` field specifies the encryption key length in bytes.
- The `encryptionKey.data` field specifies the address of the encryption key. A new key will be generated and returned in this parameter if the key address is NULL. If a new key is generated, the application should call the `gsk_free_buffer()` routine to release the key when it is no longer needed. Note that the `encryptionType` and `encryptionKey.length` fields must be set by the application even when a new session key is to be generated.

**recipient_certificates**
Specifies the certificates for the message recipients. There must be at least one recipient.

**data**
Specifies the application data for the EnvelopedData message.

**stream**
Returns the ASN.1 DER-encoded stream. The application should call the `gsk_free_buffer()` routine to release the stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_ALG_NOT_AVAILABLE]**
  The encryption algorithm is not available.

- **[CMSERR_ALG_NOT_SUPPORTED]**
  The encryption algorithm is not supported.

- **[CMSERR_BAD_KEY_SIZE]**
  The encryption key size is not supported.

- **[CMSERR_CONTENT_NOT_SUPPORTED]**
  The content type is not supported.

- **[CMSERR_INCORRECT_KEY_USAGE]**
  A recipient certificate does not allow key encipherment.
The gsk_make_enveloped_data_msg() routine creates a PKCS #7 (Cryptographic Message Syntax) EnvelopedData message and returns the ASN.1 DER-encoded ContentInfo sequence. The enveloped data content type will be Data. The gsk_read_enveloped_data_msg() routine can be used to extract the application data from the stream. No validity checking is performed on the recipient certificates. It is assumed that the application has already validated the recipient certificates.

Calling the gsk_make_enveloped_data_msg() routine is equivalent to calling the gsk_make_data_content() routine, the gsk_make_enveloped_data_content() routine, and the gsk_make_content_msg() routine.

The session key is used to encrypt the message content. A new session key is generated and returned to the application if no key is provided. For each recipient, the session key is encrypted with the recipient's public key and stored in the EnvelopedData message. This means the public key algorithm must support data encryption. Currently, only RSA public keys support data encryption. In addition, the certificate key usage must allow key encipherment.

These encryption algorithms are supported. Strong encryption may not be available depending upon government export regulations.

- x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - Key lengths 5 and 16 - {1.2.840.113549.3.2}
- x509_alg_rc4 - 40-bit and 128-bit RC4 - Key lengths 5 and 16 - {1.2.840.113549.3.4}
- x509_alg_desCbcPad - 56-bit DES - Key length 8 - {1.3.14.3.2.7}
- x509_alg_desEde3CbcPad - 168-bit 3DES - Key length 24 - {1.2.840.113549.3.7}

When executing in FIPS mode, encryption algorithms x509_alg_rc2CbcPad, x509_alg_rc4 and x509_alg_desCbcPad are not supported.
gsk_make_enveloped_data_msg_extended()

Creates a PKCS #7 EnvelopedData message from application data.

Format

```
#include <gskcms.h>

gsk_status gsk_make_enveloped_data_msg_extended (  
    gsk_process_option *option_flag,  
    int version,  
    pkcs_session_key *session_key,  
    pkcs_certificates *recipient_certificates,  
    gsk_buffer *data,  
    gsk_buffer *stream)
```

Parameters

**option_flag**

Specifies process options to customize process behavior:

- Enforce recipient certificate has key encryption capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate keyEncryption.

**version**

Specifies the PKCS #7 EnvelopedData version number. Specify 0 to create an EnvelopedData message as described in PKCS #7 Version 1.5. Specify 1 to create an EnvelopedData message as described in PKCS #7 Version 1.6.

**session_key**

Specifies the session encryption key as follows:

- The encryptionType field specifies the encryption algorithm.
- The encryptionKey.length field specifies the encryption key length in bytes.
- The encryptionKey.data field specifies the address of the encryption key. A new key will be generated and returned in this parameter if the key address is NULL. If a new key is generated, the application should call the gsk_free_buffer() routine to release the key when it is no longer needed. Note that the encryptionType and encryptionKey.length fields must be set by the application even when a new session key is to be generated.

**recipient_certificates**

Specifies the certificates for the message recipients. There must be at least one recipient.

**data**

Specifies the application data for the EnvelopedData message.

**stream**

Returns the ASN.1 DER-encoded stream. The application should call the gsk_free_buffer() routine to release the stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

[CMSERR_ALG_NOT_AVAILABLE]

The encryption algorithm is not available.

[CMSERR_ALG_NOT_SUPPORTED]

The encryption algorithm is not supported.

[CMSERR_BAD_KEY_SIZE]

The encryption key size is not supported.
The gsk_make_enveloped_data_msg_extended() routine creates a PKCS #7 (Cryptographic Message Syntax) EnvelopedData message and returns the ASN.1 DER-encoded ContentInfo sequence. Processing is equivalent to gsk_make_enveloped_data_msg(), except that the recipient certificate key usage need not assert key encipherment. The enveloped data content type will be Data. The gsk_read_enveloped_data_msg() routine or the gsk_read_enveloped_data_msg_extended() routine can be used to extract the application data from the stream. No validity checking is performed on the recipient certificates. It is assumed that the application has already validated the recipient certificates.

Calling the gsk_make_enveloped_data_msg_extended() routine is equivalent to calling the gsk_make_data_content() routine, the gsk_make_enveloped_data_content_extended() routine, and the gsk_make_content_msg() routine.

The session key is used to encrypt the message content. A new session key is generated and returned to the application if no key is provided. For each recipient, the session key is encrypted with the recipient’s public key and stored in the EnvelopedData message. This means the public key algorithm must support data encryption. Currently, only RSA public keys support data encryption. In addition, if option_flag specifies that key encipherment is to be enforced, then the certificate key usage must allow key encipherment.

These encryption algorithms are supported. Strong encryption may not be available depending upon government export regulations.

- x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - Key lengths 5 and 16 - \{1.2.840.113549.3.2\}
- x509_alg_rc4 - 40-bit and 128-bit RC4 - Key lengths 5 and 16 - \{1.2.840.113549.3.4\}
- x509_alg_desCbcPad - 56-bit DES - Key length 8 - \{1.3.14.3.2.7\}
- x509_alg_desEde3CbcPad - 168-bit 3DES - Key length 24 - \{1.2.840.113549.3.7\}

When executing in FIPS mode, encryption algorithms x509_alg_rc2CbcPad, x509_alg_rc4 and x509_alg_desCbcPad are not supported.
gsk_make_signed_data_content()

gsk_make_signed_data_content()

Creates PKCS #7 SignedData content information.

Format
#include <gskcms.h>

int

x509_algorithm_type

gsk_boolean

pkcs_cert_keys *

pkcs_certificates *

pkcs_content_info *

PKCS #7 SignedData content information.

Parameters

version
Specifies the PKCS #7 SignedData version number. Specify 0 to create SignedData content
information as described in PKCS #7 Version 1.4, specify 1 to create SignedData content information
as described in PKCS #7 Version 1.5, or specify 2 to create SignedData content information as
described in PKCS #7 Version 1.6.

digest_algorithm
Specifies the digest algorithm.

include_certificates
Specify TRUE if the signer and certification authority certificates are to be included in the SignedData
content information. Specify FALSE if the certificates are not to be included.

signer_certificates
Specifies the certificates and associated private keys for the message signers. There must be at least
one signer.

ca_certificates
Specifies the certification authority certificates. Zero or more certification authority certificates can be
included in the SignedData content information. This parameter is ignored if the include_certificates
parameter is set to FALSE. NULL can be specified for this parameter if no CA certificates are to be
included in the message.

content_data
Specifies the SignedData content. This must be one of the content information types defined in PKCS
#7.

content_info
Returns the SignedData content information. The application should call the gsk_free_content_info()
routine to release the content information when it is no longer needed.

Results
The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes
listed in the gskcms.h include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported.
** USAGE **

The `gsk_make_signed_data_content()` routine creates PKCS #7 (Cryptographic Message Syntax) SignedData content information. The data content type must be one of the types defined by PKCS #7. The `gsk_read_signed_data_content()` routine can be used to extract the content data from the SignedData content information. The key usage for the signer certificates must allow digital signature. No validity checking will be performed on the signer certificates. It is assumed that the application has already validated the signer certificates.

A signature is included for each signer provided by the `signer_certificates` parameter. The X.509 certificates used to sign the message will be included in the SignedData content information if the `include_certificates` parameter is set to TRUE. The message receiver will need to provide the signer certificates if the `include_certificates` parameter is set to FALSE.

You can optionally include certification authority certificates in the SignedData content information. These certificate can then be used by the message receiver to validate the signer certificates.

These digest algorithms are supported:

- **x509_alg_md2Digest**
  - MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

- **x509_alg_md5Digest**
  - MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

- **x509_alg_sha1Digest**
  - SHA-1 digest (RSA and DSA keys only) - {1.3.14.3.2.26}

- **x509_alg_sha224Digest**
  - SHA-224 digest (RSA keys only) - {2.16.840.1.101.3.4.2.4}

- **x509_alg_sha256Digest**
  - SHA-256 digest (RSA keys only) - {2.16.840.1.101.3.4.2.1}

- **x509_alg_sha384Digest**
  - SHA-384 digest (RSA keys only) - {2.16.840.1.101.3.4.2.2}

- **x509_alg_sha512Digest**
  - SHA-512 digest (RSA keys only) - {2.16.840.1.101.3.4.2.3}

When executing in FIPS mode, digest algorithms `x509_alg_md2Digest` and `x509_alg_md5Digest` are not supported.
**gsk_make_signed_data_content_extended()**

Creates PKCS #7 SignedData content information.

**Format**

```c
#include <gskcms.h>
gsk_status gsk_make_signed_data_content_extended (  
gsk_process_option option_flag,  
int version,  
x509_algorithm_type digest_algorithm,  
gsk_boolean include_certificates,  
pkcs_cert_keys * signer_certificates,  
pkcs_certificates * ca_certificates,  
pkcs_content_info * content_data,  
gsk_attributes_signers * attributes_signers,  
pkcs_content_info * content_info,)
```

**Parameters**

- **option_flag**  
  Specifies process options to customize process behavior.  
  - Enforce signing certificate has digital signing capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate digitalSignature.  
  - Don’t allow zero-length content data

- **version**  
  Specifies the PKCS #7 SignedData version number. Specify 0 to create SignedData content information as described in PKCS #7 Version 1.4, specify 1 to create SignedData content information as described in PKCS #7 Version 1.5, or specify 2 to create SignedData content information as described in PKCS #7 Version 1.6.

- **digest_algorithm**  
  Specifies the digest algorithm.

- **include_certificates**  
  Specify TRUE if the signer and certification authority certificates are to be included in the SignedData content information. Specify FALSE if the certificates are not to be included.

- **signer_certificates**  
  Specifies the certificates and associated private keys for the message signers. There must be at least one signer.

- **ca_certificates**  
  Specifies the certification authority certificates. Zero or more certification authority certificates can be included in the SignedData content information. This parameter is ignored if the include_certificates parameter is set to FALSE. NULL can be specified for this parameter if no CA certificates are to be included in the message.

- **content_data**  
  Specifies the SignedData content. This must be one of the content information types defined in PKCS #7.

- **attributes_signers**  
  Specifies the authenticated attributes per signer to be added to the message. Specify NULL for this parameter if there are no authenticated attributes to be included in the message. If specified, the set of authenticated attributes must NOT include content-type or message-digest authenticated attributes as these are automatically provided by `gsk_make_signed_data_content_extended()`. If the set of authenticated attributes includes signing-time, then this will override the signing-time attribute.
gsk_make_signed_data_content_extended()

generated by gsk_make_signed_data_content_extended(). The digestAlgorithm field within each gsk_attributes_signer structure is ignored - the digest algorithm is specified by the digest_algorithm parameter.

content_info
Returns the SignedData content information. The application should call the gsk_free_content_info() routine to release the content information when it is no longer needed.

Results
The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported.

[CMSERR_DIGEST_KEY_MISMATCH]
The digest algorithm is not supported for the private key type.

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_SIGNER_NOT_FOUND]
No signer certificate provided or the certificate is not valid.

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid.

[CMSERR_CONTENTTYPE_NOT_ALLOWED]
The content-type authenticated attribute is not allowed in attributes_signers.

[CMSERR_MESSAGEDIGEST_NOT_ALLOWED]
The message-digest authenticated attribute is not allowed in attributes_signers.

Usage
The gsk_make_signed_data_content_extended() routine creates PKCS #7 (Cryptographic Message Syntax) SignedData content information. The data content type must be one of the types defined by PKCS #7. Processing is similar to gsk_make_signed_data_content() except for the presence of the option_flag and authenticated_attributes parameters. The gsk_read_signed_data_content() routine or the gsk_read_signed_data_content_extended() routine can be used to extract the content data from the SignedData content information. The key usage for the signer certificates can be optionally specified as to whether digital signature must be allowed. No validity checking is performed on the signer certificates. It is assumed that the application has already validated the signer certificates.

A signature is included for each signer provided by the signer_certificates parameter. The X.509 certificates used to sign the message will be included in the SignedData content information if the include_certificates parameter is set to TRUE. The message receiver will need to provide the signer certificates if the include_certificates parameter is set to FALSE.

You can optionally include certification authority certificates in the SignedData content information. These certificates can then be used by the message receiver to validate the signer certificates.
gsk_make_signed_data_content_extended()

These digest algorithms are supported:

**x509_alg_md2Digest**
MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

**x509_alg_md5Digest**
MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

**x509_alg_sha1Digest**
SHA-1 digest (RSA and DSA keys only) - {1.3.14.3.2.26}

**x509_alg_sha224Digest**
SHA-224 digest (RSA keys only) - {2.16.840.1.101.3.4.2.4}

**x509_alg_sha256Digest**
SHA-256 digest (RSA keys only) - {2.16.840.1.101.3.4.2.1}

**x509_alg_sha384Digest**
SHA-384 digest (RSA keys only) - {2.16.840.1.101.3.4.2.2}

**x509_alg_sha512Digest**
SHA-512 digest (RSA keys only) - {2.16.840.1.101.3.4.2.3}

If authenticated attributes are provided via the `attributes_signers` parameter, then signing certificates for all signers represented within the gsk_attributes_signers structure must be provided via the `signer_certificates` parameter.

When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not supported.
Create a PKCS #7 SignedData message from application data.

Format

```c
#include <gskcms.h>
gsk_status gsk_make_signed_data_msg(
    int version,
    x509_algorithm_type digest_algorithm,
    gsk_boolean include_certificates,
    pkcs_cert_keys *signer_certificates,
    pkcs_certificates *ca_certificates,
    gsk_buffer *data,
    gsk_buffer *stream)
```

Parameters

- **version**
  Specifies the PKCS #7 SignedData version number. Specify 0 to create a SignedData message as described in PKCS #7 Version 1.4, specify 1 to create a SignedData message as described in PKCS #7 Version 1.5, or specify 2 to create a SignedData message as described in PKCS #7 Version 1.6.

- **digest_algorithm**
  Specifies the digest algorithm.

- **include_certificates**
  Specify TRUE if the signer and certification authority certificates are to be included in the SignedData message. Specify FALSE if the certificates are not to be included.

- **signer_certificates**
  Specifies the certificates and associated private keys for the message signers. There must be at least one signer.

- **ca_certificates**
  Specifies the certification authority certificates. Zero or more certification authority certificates can be included in the SignedData message. This parameter is ignored if the include_certificates parameter is set to FALSE. NULL can be specified for this parameter if no CA certificates are to be included in the message.

- **data**
  Specifies the application data for the SignedData message.

- **stream**
  Returns the ASN.1 DER-encoded stream. The application should call the `gsk_free_buffer()` routine to release the stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_ALG_NOT_SUPPORTED]**
  The digest algorithm is not supported.

- **[CMSERR_CONTENT_NOT_SUPPORTED]**
  The content type is not supported.

- **[CMSERR_DIGEST_KEY_MISMATCH]**
  The digest algorithm is not supported for the private key type.
gsk_make_signed_data_msg()

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_SIGNER_NOT_FOUND]
No signer certificate provided or the certificate is not valid.

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid.

Usage
The gsk_make_signed_data_msg() routine creates a PKCS #7 (Cryptographic Message Syntax) SignedData message and returns the ASN.1 DER-encoded ContentInfo sequence. The signed data content type will be Data. The gsk_read_signed_data_msg() routine can be used to extract the application data from the stream. The key usage for the signer certificates must allow digital signature. No validity checking will be performed on the signer certificates. It is assumed that the application has already validated the signer certificates.

Calling the gsk_make_signed_data_msg() routine is equivalent to calling the gsk_make_data_content() routine, the gsk_make_signed_data_content() routine, and the gsk_make_content_msg() routine.

A signature is included for each signer provided by the signer_certificates parameter. The X.509 certificates used to sign the message will be included in the SignedData message if the include_certificates parameter is set to TRUE. The message receiver will need to provide the signer certificates if the include_certificates parameter is set to FALSE.

You can optionally include certification authority certificates in the SignedData message. These certificates can then be used by the message receiver to validate the signer certificates.

These digest algorithms are supported:

x509_alg_md2Digest
  MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

x509_alg_md5Digest
  MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

x509_alg_sha1Digest
  SHA-1 digest (RSA and DSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest
  SHA-224 digest (RSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest
  SHA-256 digest (RSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest
  SHA-384 digest (RSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest
  SHA-512 digest (RSA keys only) - {2.16.840.1.101.3.4.2.3}

When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not supported.
gsk_make_signed_data_msg_extended()

Creates a PKCS #7 SignedData message from application data.

Format

```c
#include <gskcms.h>
gsk_status gsk_make_signed_data_msg_extended (gsk_process_option option_flag,
                                            int version,
                                            x509_algorithm_type digest_algorithm,
                                            gsk_boolean include_certificates,
                                            pkcs_cert_keys *signer_certificates,
                                            pkcs_certificates *ca_certificates,
                                            gsk_buffer *data,
                                            gsk_attributes_signers *attributes_signers,
                                            gsk_buffer *stream)
```

Parameters

**option_flag**
Specifies process options to customize process behavior.
- Enforce signing certificate has digital signing capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate digitalSignature.
- Don't allow zero-length content data

**version**
Specifies the PKCS #7 SignedData version number. Specify 0 to create a SignedData message as described in PKCS #7 Version 1.4, specify 1 to create a SignedData message as described in PKCS #7 Version 1.5, or specify 2 to create a SignedData message as described in PKCS #7 Version 1.6.

**digest_algorithm**
Specifies the digest algorithm.

**include_certificates**
Specify TRUE if the signer and certification authority certificates are to be included in the SignedData message. Specify FALSE if the certificates are not to be included.

**signer_certificates**
Specifies the certificates and associated private keys for the message signers. There must be at least one signer.

**ca_certificates**
Specifies the certification authority certificates. Zero or more certification authority certificates can be included in the SignedData message. This parameter is ignored if the include_certificates parameter is set to FALSE. NULL can be specified for this parameter if no CA certificates are to be included in the message.

**data**
Specifies the application data for the SignedData message.

**attributes_signers**
Specifies the authenticated attributes per signer to be added to the message. Specify NULL for this parameter if there are no authenticated attributes to be included in the message. If specified, then the set of authenticated attributes must NOT include content-type or message-digest authenticated attributes as these are automatically provided by `gsk_make_signed_data_msg_extended()`. If the set of authenticated attributes includes signing-time, then this will override the signing-time attribute generated by `gsk_make_signed_data_msg_extended()`. The `digest_algorithm` field within each `gsk_attributes_signer` structure is ignored - the digest algorithm is specified by the `digest_algorithm` parameter.
stream
Returns the ASN.1 DER-encoded stream. The application should call the gsk_free_buffer() routine to
release the stream when it is no longer needed.

Results
The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes
listed in the gskcms.h include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported.

[CMSERR_DIGEST_KEY_MISMATCH]
The digest algorithm is not supported for the private key type.

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_SIGNER_NOT_FOUND]
No signer certificate provided or the certificate is not valid.

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid.

[CMSERR_CONTENTTYPE_NOT_ALLOWED]
The content-type authenticated attribute is not allowed in attributes_signers.

[CMSERR_MESSAGEDIGEST_NOT_ALLOWED]
The message-digest authenticated attribute is not allowed in attributes_signers.

Usage
The gsk_make_signed_data_msg_extended() routine creates a PKCS #7 (Cryptographic Message
Syntax) SignedData message and returns the ASN.1 DER-encoded ContentInfo sequence. The signed
data content type will be Data. The gsk_read_signed_data_msg() or the
gsk_read_signed_data_msg_extended() routine can be used to extract the application data from the
stream. The key usage for the signer certificates can be optionally specified as to whether digital signature
must be allowed. No validity checking will be performed on the signer certificates. It is assumed that the
application has already validated the signer certificates.

Calling the gsk_make_signed_data_msg_extended() routine is equivalent to calling the
gsk_make_data_content() routine, the gsk_make_signed_data_content_extended() routine, and the
gsk_make_content_msg() routine.

A signature is included for each signer provided by the signer_certificates parameter. The X.509
certificates used to sign the message will be included in the SignedData message if the
include_certificates parameter is set to TRUE. The message receiver will need to provide the signer
certificates if the include_certificates parameter is set to FALSE.

You can optionally include certification authority certificates in the SignedData message. These certificates
can then be used by the message receiver to validate the signer certificates.

These digest algorithms are supported:
x509_alg_md2Digest
   MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

x509_alg_md5Digest
   MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

x509_alg_sha1Digest
   SHA-1 digest (RSA and DSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest
   SHA-224 digest (RSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest
   SHA-256 digest (RSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest
   SHA-384 digest (RSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest
   SHA-512 digest (RSA keys only) - {2.16.840.1.101.3.4.2.3}

If authenticated attributes are provided via the attributes_signers parameter, then signing certificates for all signers represented within the gsk_attributes_signers structure must be provided via the signer_certificates parameter.

When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not supported.
gsk_make_wrapped_content()

Format
#include <gskcms.h>

#include <gskcms.h>
gsk_status gsk_make_wrapped_content ( pkcs_content_info * content_info, pkcs_content_info * wrapped_content )

Parameters

current_info
    Specifies the content information to be wrapped.

wrapped_content
    Returns the wrapped content information. The application should call the gsk_free_content_info() routine to release the content information when it is no longer needed.

Results
The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage
The gsk_make_wrapped_content() routine wraps the supplied content information in an ASN.1 sequence and returns a new content information containing the wrapped data. The type of the wrapped content information is the same as the type of the original content information. The gsk_read_wrapped_content() routine can be used to extract the original content information.
gsk_mktime()

Converts year/month/day time value to number of seconds since the POSIX epoch

Format
#include <gskcms.h>

gsk_time gsk_mktime (gsk_timeval * ts)

Parameters

`ts` Specifies the time to be converted. The `tm_year`, `tm_mon`, `tm_mday`, `tm_hour`, `tm_min`, and `tm_sec` fields are used to generate the converted time.

Results

The return value is the number of seconds since January 1, 1970. Leap seconds are not included in the computation.

Usage

The `gsk_mktime()` routine converts the time specified in year/month/day format to the number of seconds since the POSIX epoch (January 1, 1970). The `gsk_mktime()` routine differs from the `mktime()` routine in that the time is UTC and is not adjusted for the local timezone or for daylight savings time.

The year value must be between 1970 and 2106 and is the actual year minus 1900, so `tm_year` must be between 70 and 206, `tm_mon` must be between 0 and 11, `tm_mday` must be between 1 and 31, `tm_hour` must be between 0 and 23, `tm_min` must be between 0 and 59, and `tm_sec` must be between 0 and 59.
gsk_name_compare()

Compares two X.509 names.

Format

```c
#include <gskcms.h>

gsk_boolean gsk_name_compare(
    x509_name * name1,
    x509_name * name2)
```

Parameters

- `name1`: Specifies the first name to be compared.
- `name2`: Specifies the second name to be compared.

Results

Returns TRUE if the two x.509 names are the same and FALSE if the two x.509 names are different.

Usage

The `gsk_name_compare()` routine compares two X.509 names and return TRUE if the names are the same and FALSE if they are not the same.

Two names are considered equal if they contain the same sequence of attribute types and attribute values. Attribute values are considered equal if they represent the same character string. If a relative distinguished name (RDN) contains multiple attributes, the attributes must be specified in ascending order based upon their ASN.1 DER encoding. Strings are always stored using UTF-8 encoding.

Printable strings (`gsk_string_printable`) are a special case. Multiple spaces are treated as a single space and the comparison is not case sensitive. Case-sensitive comparisons are used for all other string types.
gsk_name_to_dn()

Converts an X.509 name to a DN string.

Format

```c
#include <gskcms.h>

gsk_status gsk_name_to_dn (x509_name * name, char ** dn)
```

Parameters

- `name`: Specifies the X.509 name to be converted to a distinguished name string. The X.509 strings use UTF-8 encoding.
- `dn`: Returns the distinguished name in the local code page. The application should call the `gsk_free_string()` routine to release the string when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[ASN_CANT_CONVERT]**
  - The X.509 name is not a distinguished name.

- **[ASN_NO_MEMORY]**
  - Insufficient storage is available.

Usage

The `gsk_name_to_dn()` routine converts an X.509 name to a distinguished name (DN) string in accordance with RFC 2253 (UTF-8 String Representation of Distinguished Names). The DN string will consist of single-byte characters in the local code page. A double-byte character will be represented using the escaped UTF-8 encoding of the double-byte character in the UCS-2 or UCS-4 character set.

These DN attribute names are generated by the System SSL runtime. Unrecognized attribute types will be encoded using the numeric object identifier followed by the DER-encoded representation of the attribute value.

- **C** - Country
- **CN** - Common name
- **DC** - Domain component
- **DNQUALIFIER** - Distinguished name qualifier
- **EMAIL** - E-mail address
- **GENERATIONQUALIFIER** - Generation qualifier
- **GIVENNAME** - Given name
- **INITIALS** - Initials
- **L** - Locality
- **MAIL** - Mail RFC822 style address
- **NAME** - Name
- **O** - Organization name
- **OU** - Organizational unit name
- **PC** - Postal code
gsk_name_to_dn()

- SERIALNUMBER - Serial number
- SN - Surname
- ST - State or province
- STREET - Street
- T - Title
gsk_open_database()

Opens a key or request database.

**Format**

```c
#include <gskcms.h>
gsk_status gsk_open_database (
    const char * filename,
    const char * password,
    gsk_boolean update_mode,
    gsk_handle * db_handle,
    gskdb_database_type * db_type,
    int * num_records)
```

**Parameters**

- **filename**
  Specifies the database file name in the local code page. The length of the fully-qualified filename cannot exceed 251.

- **password**
  Specifies the database password in the local code page. The user will be prompted to enter the password if NULL is specified for this parameter.

- **update_mode**
  Specifies the file access mode. Specify TRUE if the database will be updated and FALSE if the database will not be updated. The application must have write access to the file if TRUE is specified.

- **db_handle**
  Returns the database handle. The application should call the `gsk_close_database()` routine when it no longer needs access to the database.

- **db_type**
  Returns the database type.

- **num_records**
  Returns the number of records in the database.

**Results**

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_ACCESS_DENIED]**
  The file permissions do not allow access.

- **[CMSERR_BAD_FILENAME]**
  The database file name is not valid.

- **[CMSERR_BAD_RNG_OUTPUT]**
  In FIPS mode, random bytes generation produced duplicate output.

- **[CMSERR_DB_CORRUPTED]**
  The database file is not valid.

- **[CMSERR_DB_FIPS_MODE_ONLY]**
  Key database can only be opened for update if running in FIPS mode.

- **[CMSERR_DB_LOCKED]**
  The database is open for update by another process.
gsk_open_database()

- [CMSERR_DB_NOT_FIPS] Key database is not a FIPS mode database.
- [CMSERR_FILE_NOT_FOUND] The database file is not found.
- [CMSERR_IO_CANCELED] The user canceled the password prompt.
- [CMSERR_NO_MEMORY] Insufficient storage is available.
- [CMSERR_OPEN_FAILED] Unable to open the database.

**Usage**

The `gsk_open_database()` routine will open a key or request database file for either read-only or read/write access. The database must already exist. The database integrity will be verified and the open will fail if the database has been incorrectly modified. Only one process at a time may open a database in update mode. The database may be accessed by multiple concurrent threads in the same process as long as the same database handle is used by all of the threads.

- A FIPS database file may only be opened for update while executing in FIPS mode. A FIPS database may be opened read-only while executing in non-FIPS mode. A non-FIPS database file cannot be opened for read or update while executing in FIPS mode.
gsk_open_database_using_stash_file()

Opens a key or request database using a stash file for the database password.

Format

```c
#include <gskcms.h>
gsk_status gsk_open_database_using_stash_file(
    const char * database_filename,
    const char * stash_filename,
    gsk_boolean update_mode,
    gsk_handle * db_handle,
    gskdb_database_type * db_type,
    int * num_records)
```

Parameters

database_filename
    Specifies the database file name in the local code page. The length of the fully-qualified filename
    cannot exceed 251.

stash_filename
    Specifies the stash file name in the local code page. The length of the fully-qualified filename cannot
    exceed 251. The stash file name always has an extension of "sth" and the supplied name will be
    changed if it does not have the correct extension.

update_mode
    Specifies the file access mode. Specify TRUE if the database will be updated and FALSE if the
    database will not be updated. The application must have write access to the file if TRUE is specified.

db_handle
    Returns the database handle. The application should call the gsk_close_database() routine when it
    no longer needs access to the database.

db_type
    Returns the database type.

num_records
    Returns the number of records in the database.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes
listed in the gskcms.h include file. These are some possible errors:

[CMSERR_ACCESS_DENIED]
    The file permissions do not allow access.

[CMSERR_BAD_FILENAME]
    The database file name is not valid.

[CMSERR_DB_CORRUPTED]
    The database file is not valid.

[CMSERR_DB_FIPS_MODE_ONLY]
    Key database can only be opened for update if running in FIPS mode.

[CMSERR_DB_LOCKED]
    The database is open for update by another process.

[CMSERR_NOT_FIPS]
    Key database is not a FIPS mode database.
gsk_open_database_using_stash_file()

[CMSERR_FILE_NOT_FOUND]
The database file is not found.

[CMSERR_IO_ERROR]
An input/output request failed.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_OPEN_FAILED]
Unable to open the database.

Usage
The gsk_open_database_using_stash_file() routine is the same as the gsk_open_database() routine except the database password is obtained from the password stash file instead of being specified as a call parameter. The key or request database can be opened for read-only access or for read/write access. The database must already exist. The database integrity will be verified and the open will fail if the database has been incorrectly modified. Only one process at a time may open a database in update mode. The database may be accessed by multiple concurrent threads in the same process as long as the same database handle is used by all of the threads.

A FIPS database file may only be opened for update while executing in FIPS mode. A FIPS database may be opened read-only while executing in non-FIPS mode. A non-FIPS database file cannot be opened for read or update while executing in FIPS mode.
gsk_open_directory()

Opens an LDAP directory.

Format

```c
#include <gskcms.h>
gsk_status gsk_open_directory (const char * server_name,
                                int server_port,
                                const char * user_name,
                                const char * user_password,
                                int crl_cache_timeout,
                                gsk_handle * directory_handle)
```

Parameters

- **server_name**
  - Specifies one or more blank-separated LDAP server host names. Each host name can contain an optional port number separated from the host name by a colon.

- **server_port**
  - Specifies the port assigned to the LDAP server. The default port will be used if zero is specified.

- **user_name**
  - Specifies the distinguished name to be used when binding to the LDAP server. An unauthenticated bind will be done if NULL is specified for this parameter.

- **user_password**
  - Specifies the password to be used when binding to the LDAP server. NULL may be specified for this parameter when NULL is also specified for the **user_name** parameter.

- **crl_cache_timeout**
  - Specifies the CRL cache timeout interval in hours. Specify 0 to disable CRL caching.

- **directory_handle**
  - Returns the directory handle. The application should call the gsk_close_directory() routine when it no longer needs access to the LDAP directory.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

- **[CMSERR_LDAP]**
  - Error reported by the LDAP client

- **[CMSERR_LDAP_NOT_AVAILABLE]**
  - LDAP server is not available.

- **[CMSERR_NO_MEMORY]**
  - Insufficient storage is available

Usage

The gsk_open_directory() routine will open an LDAP directory and return a directory handle.
gsk_open_keyring()

Opens a SAF digital certificate key ring or z/OS PKCS #11 token.

Format
#include <gskcms.h>
gsk_status gsk_open_keyring (
    const char * ring_name,
    gsk_handle * db_handle,
    int * num_records)

Parameters

ring_name
  Specifies the SAF key ring or z/OS PKCS #11 token name in the local code page. When using a key
  ring owned by the current user, specify the ring name as "name". When using a key ring owned by
  another user, specify the ring name as "userid/name". The maximum userid length is 8 and the
  maximum name length is 237. The z/OS PKCS #11 token name is specified as "TOKEN"/token-name.
  "TOKEN" indicates that the specified key ring is actually a token name.

db_handle
  Returns the database handle. The application should call the gsk_close_database() routine when it
  no longer needs access to the keyring.

num_records
  Returns the number of records in the key ring or token.

Results
The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes
listed in the gskcms.h include file. These are some possible errors:

[CMSERR_ACCESS_DENIED]
The access permissions do not allow access.

[CMSERR_BAD_FILENAME]
The key ring or token name is not valid.

[CMSERR_BAD_RING_OUTPUT]
In FIPS mode, random bytes generation produced duplicate output.

[CMSERR_FILE_NOT_FOUND]
The key ring or token does not exist

[CMSERR_IO_ERROR]
An error occurred while listing the key ring or token.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage
The gsk_open_keyring() routine will open a key ring maintained by the System Authorization Facility
(SAF) and construct a read-only key database. Only trusted certificates connected to the specified key ring
are included in the key database. The GSKDB_RECFLAG_DEFAULT flag will be set if the certificate is the
default certificate for the key ring or token.

The user must have READ access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when
using a SAF key ring owned by the user. The user must have UPDATE access to the
IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by another user. Note that certificate private keys are not available when using a SAF key ring owned by another user.

The application userid must have READ access to resource USER.token-name in the CRYPTOZ class in order for the certificates and their private keys, if present, to be read from a z/OS PKCS #11 token.
gsk_perform_kat()

Conducts a set of known answer tests for the System SSL algorithms validated by NIST. The caller must set FIPS mode (see "gsk_fips_state_set()" on page 205) prior to calling this function.

Format
#include <gskcms.h>
gsk_status gsk_perform_kat ()

Results
The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

[CMSERR_API_NOT_SUPPORTED]
The API is not supported in non-FIPS mode.

[CMSERR_KATPW_FAILED]
A known answer test has failed. This is a severe error and the application should terminate.

Usage
The gsk_perform_kat() routine can be used whenever an application, in order to meet security requirements, needs to check the correctness of cryptographic algorithms that are part of the product. The routine performs Known Answer Tests on the following cryptographic algorithms:

- AES 128-bit and AES 256-bit encryption and decryption
- Triple-DES encryption and decryption
- RSA signature generation/verification and encryption/decryption
- RSA encrypt and decrypt
- DSA signature generation and verification
- SHA Digest Algorithms: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 and HMAC-SHA-1
- Random number generation

If an error is encountered during testing, the gsk_perform_kat() routine will terminate and return the appropriate error code.

The gsk_perform_kat() routine will test software or hardware cryptographic algorithms depending on the value of the GSK_HW_CRYPTO environment variable at the time the CMS DLL (GSKCMS31 or GSKCMS64) is loaded.
gsk_query_crypto_level()

Returns the available cryptographic levels.

Format
#include <gskcms.h>

void gsk_query_crypto_level (  
    int * cms_version,  
    int * cms_release,  
    gsk_uint32 * crypto_level)

Parameters

- cms_version
  Returns the runtime version number.

- cms_release
  Returns the runtime release number.

- crypto_level
  Returns the available cryptographic levels.

Results

The gsk_query_crypto_level() routine returns the System SSL runtime version, release, and available cryptographic levels. The current System SSL runtime is Version 3 Release 21. The cryptographic level is a bit mask as follows:

[GSK_CRYPTO_64]
    Set if 64-bit encryption keys are supported.

[GSK_CRYPTO_128]
    Set if 128-bit encryption keys are supported.

[GSK_CRYPTO_168]
    Set if 168-bit encryption keys are supported.


gsk_query_database_label()

Determines if a database label exists

Format

```c
#include <gskcms.h>

gsk_status gsk_query_database_label( gsk_handle db_handle,
const char * label)
```

Parameters

- `db_handle`
  Specifies the database handle returned by the `gsk_create_database()` routine, the `gsk_open_database()` routine, or the `gsk_open_keyring()` routine.

- `label`
  Specifies the database label. The label is specified in the local code page.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- [CMSERR_BAD_HANDLE] The database handle is not valid.
- [CMSERR_BAD_LABEL] No label specified.
- [CMSERR_MULTIPLE_LABEL] Multiple certificates exist for label.
- [CMSERR_RECORD_NOT_FOUND] The label does not exist in the database.

Usage

The `gsk_query_database_label()` routine will check the database for the requested label.
**gsk_query_database_record_length()**

Queries the database record length.

**Format**

```c
#include <gskcms.h>
gsk_status gsk_query_database_record_length (gsk_handle db_handle, gsk_size *record_length)
```

**Parameters**

- `db_handle`
  - Specifies the database handle returned by the `gsk_create_database()` routine or the `gsk_open_database()` routine.

- `record_length`
  - Returns the current database record length. All records in the database have this length.

**Results**

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- [CMSERR_BAD_HANDLE]
  - The database handle is not valid.

**Usage**

The `gsk_query_database_record_length()` routine will return the record length for the database. All records in the database have the same length and a database entry cannot span records. The `gsk_change_database_record_length()` routine can be called to change the database record length.
gsk_rdtimetime()

Converts the number of seconds since the POSIX epoch to year/month/day.

Format

```
#include <gskcms.h>

gsk_timeval * gsk_rdtimetime (  
gsk_time    secs,  
gsk_timeval * ts)
```

Parameters

- `secs` Specifies the time value to be converted.
- `ts` Returns the converted time in the tm_year, tm_mon, tm_mday, tm_hour, tm_min, and tm_sec fields.

Usage

The `gsk_rdtimetime()` routine converts the number of seconds since the POSIX epoch (January 1, 1970) to year/month/day format. The year value is the actual year minus 1900 and the month value is the actual month minus 1 (that is, January is 0 and December is 11). The return value is the same as the second parameter (the address of the struct tm).
gsk_read_content_msg()

Processes a PKCS #7 message.

Format
#include <gskcms.h>

int gsk_status gsk_read_content_msg (gsk_buffer * stream, pkcs_content_info * content_info)

Parameters
stream
   Specifies the ASN.1 DER-encoded stream to be processed.
content_info
   Returns the content information for the message. The application should call the
gsk_free_content_info() routine to release the content information when it is no longer needed.

Results
The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes
listed in the gskcms.h include file. These are some possible errors:

[CMSERR_NO_MEMORY]
   Insufficient storage is available

Usage
The gsk_read_content_msg() routine processes a PKCS #7 (Cryptographic Message Syntax) content
information message and returns the content information. The message content type can be any of the
types defined by the PKCS #7 specification.
**gsk_read_data_content()**

Processes PKCS #7 Data content information.

**Format**

```c
#include <gskcms.h>

gsk_status gsk_read_data_content (pkcs_content_info * content_info,
                                   gsk_buffer * data)
```

**Parameters**

- `content_info`
  
  Specifies the content information to be processed.

- `data`
  
  Returns the application data. The application should call the `gsk_free_buffer()` routine to release the data when it is no longer needed.

**Results**

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_CONTENT_NOT_SUPPORTED]**
  
  The content type is not Data.

- **[CMSERR_NO_MEMORY]**
  
  Insufficient storage is available.

**Usage**

The `gsk_read_data_content()` routine processes PKCS #7 (Cryptographic Message Syntax) Data content information created by the `gsk_make_data_content()` routine and returns the application data.
gsk_read_data_msg()

Processes a PKCS #7 Data message.

Format
#include <gskcms.h>
gsk_status gsk_read_data_msg (  
gsk_buffer * stream,  
gsk_buffer * data)

Parameters
stream
    Specifies the ASN.1 DER-encoded stream to be processed.

data
    Returns the application data. The application should call the gsk_free_buffer() routine to release the data when it is no longer needed.

Results
The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

[CMSERRCONTENT_NOT_SUPPORTED]
The message content type is not Data.

[CMSERRNO_MEMORY]
Insufficient storage is available.

Usage
The gsk_read_data_msg() routine processes a PKCS #7 (Cryptographic Message Syntax) Data message created by the gsk_make_data_msg() routine and returns the application data. The message content type must be Data.

Calling the gsk_read_data_msg() routine is equivalent to calling the gsk_read_content_msg() routine followed by the gsk_read_data_content() routine.
gsk_read_encrypted_data_content()

Processes PKCS #7 EncryptedData content information.

**Format**

```c
#include <gskcms.h>

gsk_status gsk_read_encrypted_data_content (const char * password,
                                          pkcs_content_info * content_info,
                                          pkcs_content_info * content_data)
```

**Parameters**

- `password`
  Specifies the encryption password as a null-terminated string in the local code page. The user will be prompted to enter the password if NULL is specified for this parameter.

- `content_info`
  Specifies the content information to be processed

- `content_data`
  Returns the decrypted content data. The application should call the `gsk_free_content_info()` routine to release the content information when it is no longer needed.

**Results**

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_ALG_NOT_AVAILABLE]**
  Encryption algorithm is not available.

- **[CMSERR_ALG_NOT_SUPPORTED]**
  Encryption algorithm is not supported.

- **[CMSERR_API_NOT_SUPPORTED]**
  The API is not supported.

- **[CMSERR_CONTENT_NOT_SUPPORTED]**
  The message content type is not EncryptedData or the content of the EncryptedData message is not supported.

- **[CMSERR_NO_CONTENT_DATA]**
  The encrypted data length is zero.

- **[CMSERR_NO_MEMORY]**
  Insufficient storage is available.

**Usage**

The `gsk_read_encrypted_data_content()` routine processes PKCS #7 (Cryptographic Message Syntax) EncryptedData content information created by the `gsk_make_encrypted_data_content()` routine and returns the decrypted content data.

- `gsk_read_encrypted_data_content()` is not supported when executing in FIPS mode and will return **CMSERR_API_NOT_SUPPORTED**.

The decryption key is derived from the password as described in PKCS #5 (Password-based Encryption) and PKCS #12 (Personal Information Exchange). The selected algorithm determines how the key is derived from the password.
These password-based encryption algorithms are supported. The strong encryption algorithms may not be available depending upon government export regulations.

- x509_alg_pbeWithMd2AndDesCbc - 56-bit DES encryption with MD2 digest - {1.2.840.113549.1.5.1}
- x509_alg_pbeWithMd5AndDesCbc - 56-bit DES encryption with MD5 digest - {1.2.840.113549.1.5.3}
- x509_alg_pbeWithSha1AndDesCbc - 56-bit DES encryption with SHA-1 digest - {1.2.840.113549.1.5.10}
- x509_alg_pbeWithMd2AndRc2Cbc - 64-bit RC2 encryption with MD2 digest - {1.2.840.113549.1.5.4}
- x509_alg_pbeWithMd5AndRc2Cbc - 64-bit RC2 encryption with MD5 digest - {1.2.840.113549.1.5.6}
- x509_alg_pbeWithSha1AndRc2Cbc - 64-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.5.11}
- x509_alg_pbeWithSha1And40BitRc2Cbc - 40-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.6}
- x509_alg_pbeWithSha1And128BitRc2Cbc - 128-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.5}
- x509_alg_pbeWithSha1And40BitRc4 - 40-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.2}
- x509_alg_pbeWithSha1And128BitRc4 - 128-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.1}
- x509_alg_pbeWithSha1And3DesCbc - 168-bit 3DES encryption with SHA-1 digest - {1.2.840.113549.1.12.1.3}
gsk_read_encrypted_data_msg()

Processes a PKCS #7 EncryptedData message.

Format

```c
#include <gskcms.h>
gsk_status gsk_read_encrypted_data_msg (const char *password, gsk_buffer *stream, gsk_buffer *data)
```

Parameters

`password`
Specifies the encryption password as a null-terminated string in the local code page. The user will be prompted to enter the password if NULL is specified for this parameter.

`stream`
Specifies the ASN.1 DER-encoded stream to be processed.

`data`
Returns the decrypted content of the EncryptedData message. The application should call the `gsk_free_buffer()` routine to release the data when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_ALG_NOT_AVAILABLE]**
  Encryption algorithm is not available.

- **[CMSERR_ALG_NOT_SUPPORTED]**
  Encryption algorithm is not supported.

- **[CMSERR_API_NOT_SUPPORTED]**
  The API is not supported.

- **[CMSERR_CONTENT_NOT_SUPPORTED]**
  The message content type is not EncryptedData or the content of the EncryptedData message is not Data.

- **[CMSERR_NO_CONTENT_DATA]**
  The encrypted data length is zero.

- **[CMSERR_NO_MEMORY]**
  Insufficient storage is available.

Usage

The `gsk_read_encrypted_data_msg()` routine processes a PKCS #7 (Cryptographic Message Syntax) EncryptedData message created by the `gsk_make_encrypted_data_msg()` routine and returns the decrypted message content. The encrypted data content type must be Data.

Calling the `gsk_read_encrypted_data_msg()` routine is equivalent to calling the `gsk_read_content_msg()` routine, the `gsk_read_encrypted_data_content()` routine, and the `gsk_read_data_content()` routine.
The decryption key is derived from the password as described in PKCS #5 (Password-based Encryption) and PKCS #12 (Personal Information Exchange). The selected algorithm determines how the key is derived from the password.

These password-based encryption algorithms are supported. The strong encryption algorithms may not be available depending upon government export regulations.

- **x509_alg_pbeWithMd2AndDesCbc** - 56-bit DES encryption with MD2 digest - {1.2.840.113549.1.5.1}
- **x509_alg_pbeWithMd5AndDesCbc** - 56-bit DES encryption with MD5 digest - {1.2.840.113549.1.5.3}
- **x509_alg_pbeWithSha1AndDesCbc** - 56-bit DES encryption with SHA-1 digest - {1.2.840.113549.1.5.10}
- **x509_alg_pbeWithMd2AndRc2Cbc** - 64-bit RC2 encryption with MD2 digest - {1.2.840.113549.1.5.4}
- **x509_alg_pbeWithMd5AndRc2Cbc** - 64-bit RC2 encryption with MD5 digest - {1.2.840.113549.1.5.6}
- **x509_alg_pbeWithSha1AndRc2Cbc** - 64-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.5.11}
- **x509_alg_pbeWithSha1And40BitRc2Cbc** - 40-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.6}
- **x509_alg_pbeWithSha1And128BitRc2Cbc** - 128-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.5}
- **x509_alg_pbeWithSha1And40BitRc4** - 40-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.2}
- **x509_alg_pbeWithSha1And128BitRc4** - 128-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.1}
- **x509_alg_pbeWithSha1And3DesCbc** - 168-bit 3DES encryption with SHA-1 digest - {1.2.840.113549.1.12.1.3}
gsk_read_enveloped_data_content()

Processes PKCS #7 EnvelopedData content information.

Format
#include <gskcms.h>
gsk_status gsk_read_enveloped_data_content (pkcs_cert_keys *recipient_keys, pkcs_content_info *content_info, x509_algorithm_type *encryption_algorithm, gsk_size *key_size, pkcs_content_info *content_data)

Parameters
recipient_keys
   Specifies one or more certificates and associated private keys.

content_info
   Specifies the content information to be processed.

encryption_algorithm
   Returns the encryption algorithm used to encrypt the message content.

key_size
   Returns the encryption key size in bytes.

content_data
   Returns the EnvelopedData content data. The application should call the gsk_free_content_info() routine to release the content information when it is no longer needed.

Results
The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

[CMSERR_ALG_NOT_AVAILABLE]
The encryption algorithm is not available.

[CMSERR_ALG_NOT_SUPPORTED]
The encryption algorithm is not supported.

[CMSERR_BAD_KEY_SIZE]
The encryption key size is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The message content type is not EnvelopedData or the content of the EnvelopedData message is not supported.

[CMSERR_INCORRECT_KEY_USAGE]
The recipient certificate does not allow key encipherment.

[CMSERR_KEY_MISMATCH]
A recipient private key does not support data decryption.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.
gsk_read_enveloped_data_content()

[CMSERR_RECIPIENT_NOT_FOUND]
No matching recipient certificate provided.

Usage
The gsk_read_enveloped_data_content() routine processes PKCS #7 (Cryptographic Message Syntax) EnvelopedData content information created by the gsk_make_enveloped_data_content() routine.

The recipient_keys parameter supplies one or more recipient certificates and associated private keys. The gsk_read_enveloped_data_content() routine will search for a certificate matching one of the message recipients. The private key will be used to decrypt the session key and the session key will then be used to decrypt the enveloped data. The certificate key usage must allow key encipherment.

No certificate validation is performed by the gsk_read_enveloped_data_content() routine. It is assumed that the application has already validated the recipient certificates.

These encryption algorithms are supported. Strong encryption may not be available depending upon government export regulations.
- x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - {1.2.840.113549.3.2}
- x509_alg_rc4 - 40-bit and 128-bit RC4 - {1.2.840.113549.3.4}
- x509_alg_desCbcPad - 56-bit DES - {1.3.14.3.2.7}
- x509_alg_desEde3CbcPad - 168-bit 3DES - {1.2.840.113549.3.7}

When executing in FIPS mode, encryption algorithms x509_alg_rc2CbcPad, x509_alg_rc4 and x509_alg_desCbcPad are not supported.
gsk_read_enveloped_data_content_extended()

Processes PKCS #7 EnvelopedData content information.

Format

```c
#include <gskcms.h>
gsk_status gsk_read_enveloped_data_content_extended (  
gsk_process_option option_flag,  
pkcs_cert_keys * recipient_keys,  
pkcs_content_info * content_info,  
x509_algorithm_type * encryption_algorithm,  
gsk_size * key_size,  
pkcs_content_info * content_data)
```

Parameters

- **option_flag**
  Specifies process options to customize process behavior.
  - Enforce recipient certificate has key encipherment capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate keyEncipherment.
  - Enforce key parity when using DES or 3DES session keys.

- **recipient_keys**
  Specifies one or more certificates and associated private keys.

- **content_info**
  Specifies the content information to be processed.

- **encryption_algorithm**
  Returns the encryption algorithm used to encrypt the message content.

- **key_size**
  Returns the encryption key size in bytes.

- **content_data**
  Returns the EnvelopedData content data. The application should call the `gsk_free_content_info()` routine to release the content information when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_ALG_NOT_AVAILABLE]**
  The encryption algorithm is not available.

- **[CMSERR_ALG_NOT_SUPPORTED]**
  The encryption algorithm is not supported.

- **[CMSERR_BAD_KEY_SIZE]**
  The encryption key size is not supported.

- **[CMSERR_CONTENT_NOT_SUPPORTED]**
  The message content type is not EnvelopedData or the content of the EnvelopedData message is not supported.

- **[CMSERR_INCORRECT_KEY_USAGE]**
  The recipient certificate does not allow key encipherment.

- **[CMSERR_KEY_MISMATCH]**
  A recipient private key does not support data decryption.
gsk_read_enveloped_data_content_extended()

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECIPIENT_NOT_FOUND]
No matching recipient certificate provided.

Usage
The gsk_read_enveloped_data_content_extended() routine processes PKCS #7 (Cryptographic Message Syntax) EnvelopedData content information created by the gsk_make_enveloped_data_content() routine or the gsk_make_enveloped_data_content_extended() routine. Processing is equivalent to gsk_read_enveloped_data_content(), except that the recipient certificate key usage need not assert key encipherment.

The recipient_keys parameter supplies one or more recipient certificates and associated private keys. The gsk_read_enveloped_data_content_extended() routine will search for a certificate matching one of the message recipients. The private key will be used to decrypt the session key and the session key will then be used to decrypt the enveloped data. In addition, if option_flag specifies that key encipherment is to be enforced, then the certificate key usage must allow key encipherment and session keys need not be odd parity.

No certificate validation is performed by the gsk_read_enveloped_data_content_extended() routine. It is assumed that the application has already validated the recipient certificates.

These encryption algorithms are supported. Strong encryption may not be available depending upon government export regulations.

- x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - {1.2.840.113549.3.2}
- x509_alg_rc4 - 40-bit and 128-bit RC4 - {1.2.840.113549.3.4}
- x509_alg_desCbcPad - 56-bit DES - {1.3.14.3.2.7}
- x509_alg_desEde3CbcPad - 168-bit 3DES - {1.2.840.113549.3.7}

When executing in FIPS mode, encryption algorithms x509_alg_rc2CbcPad, x509_alg_rc4 and x509_alg_desCbcPad are not supported.
gsk_read_enveloped_data_msg()

Processes a PKCS #7 EnvelopedData message.

Format

```c
#include <gskcms.h>
gsk_status gsk_read_enveloped_data_msg (pkcs_cert_keys *recipient_keys, gsk_buffer *stream, x509_algorithm_type *encryption_algorithm, gsk_size *key_size, gsk_buffer *data)
```

Parameters

- **recipient_keys**
  - Specifies one or more certificates and associated private keys.

- **stream**
  - Specifies the ASN.1 DER-encoded stream to be processed.

- **encryption_algorithm**
  - Returns the encryption algorithm used to encrypt the message content.

- **key_size**
  - Returns the encryption key size in bytes.

- **data**
  - Returns the content of the EnvelopedData message. The application should call the `gsk_free_buffer()` routine to release the data when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_ALG_NOT_SUPPORTED]**
  - Encryption algorithm is not supported.

- **[CMSERR_BAD_ENCODING]**
  - The message content type is not EnvelopedData or the message content is not Data.

- **[CMSERR_BAD_KEY_SIZE]**
  - The encryption key size is not supported.

- **[CMSERR_CONTENT_NOT_SUPPORTED]**
  - The message content type is not EnvelopedData or the content of the EnvelopedData message is not Data.

- **[CMSERR_INCORRECT_KEY_USAGE]**
  - The recipient certificate does not allow key encipherment.

- **[CMSERR_KEY_MISMATCH]**
  - A recipient private key does not support data decryption.

- **[CMSERR_NO_CONTENT_DATA]**
  - The content data length is zero.

- **[CMSERR_NO_MEMORY]**
  - Insufficient storage is available.
gsk_read_enveloped_data_msg()

[CMSERR_RECIPIENT_NOT_FOUND]
No matching recipient certificate provided.

Usage
The gsk_read_enveloped_data_msg() routine processes a PKCS #7 (Cryptographic Message Syntax) EnvelopedData message created by the gsk_make_enveloped_data_msg() routine and returns the message content. The enveloped data content type must be Data.

Calling the gsk_read_enveloped_data_msg() routine is equivalent to calling the gsk_read_content_msg() routine, the gsk_read_enveloped_data_content() routine, and the gsk_read_data_content() routine.

The recipient_keys parameter supplies one or more recipient certificates and associated private keys. The gsk_read_enveloped_data_msg() routine will search for a certificate matching one of the message recipients. The private key will be used to decrypt the session key and the session key will then be used to decrypt the enveloped data. The certificate key usage must allow key encryption.

No certificate validation is performed by the gsk_read_enveloped_data_msg() routine. It is assumed that the application has already validated the recipient certificates.

These encryption algorithms are supported. Strong encryption may not be available depending upon government export regulations.
- x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - {1.2.840.113549.3.2}
- x509_alg_rc4 - 40-bit and 128-bit RC4 - {1.2.840.113549.3.4}
- x509_alg_desCbcPad - 56-bit DES - {1.3.14.3.2.7}
- x509_alg_desEde3CbcPad - 168-bit 3DES - {1.2.840.113549.3.7}

When executing in FIPS mode, encryption algorithms x509_alg_rc2CbcPad, x509_alg_rc4 and x509_alg_desCbcPad are not supported.
gsk_read_enveloped_data_msg_extended()

Processes a PKCS #7 EnvelopedData message.

Format
#include <gskcms.h>
gsk_status gsk_read_enveloped_data_msg_extended (  
gsk_process_option option_flag,  
pkcs_cert_keys * recipient_keys,  
gsk_buffer * stream,  
x509_algorithm_type * encryption_algorithm,  
gsk_size * key_size,  
gsk_buffer * data)

Parameters

option_flag
  Specifies process options to customize process behavior.
  • Enforce recipient certificate has key encipherment capabilities. That is, the purpose of the certificate
    key as reflected by the key usage extension must indicate keyEncipherment.
  • Enforce key parity when using DES or 3DES session keys.

recipient_keys
  Specifies one or more certificates and associated private keys.

stream
  Specifies the ASN.1 DER-encoded stream to be processed.

enCRYPTION_algorithm
  Returns the encryption algorithm used to encrypt the message content.

key_size
  Returns the encryption key size in bytes.

data
  Returns the content of the EnvelopedData message. The application should call the gsk_free_buffer()  
routine to release the data when it is no longer needed.

Results
The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes  
listed in the gskcms.h include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
  Encryption algorithm is not supported.

[CMSERR_BAD_ENCODING]
  The message content type is not EnvelopedData or the message content is not Data.

[CMSERR_BAD_KEY_SIZE]
  The encryption key size is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
  The message content type is not EnvelopedData or the content of the EnvelopedData message is
  not Data.

[CMSERR_INCORRECT_KEY_USAGE]
  The recipient certificate does not allow key encipherment.

[CMSERR_KEY_MISMATCH]
  A recipient private key does not support data decryption.
gsk_read_enveloped_data_msg_extended() 

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECIPIENT_NOT_FOUND]
No matching recipient certificate provided.

Usage
The gsk_read_enveloped_data_msg_extended() routine processes a PKCS #7 (Cryptographic Message Syntax) EnvelopedData message created by the gsk_make_enveloped_data_content() routine or the gsk_make_enveloped_data_msg_extended() routine and returns the message content. Processing is equivalent to gsk_read_enveloped_data_content(), except that the recipient certificate key usage need not assert key encipherment and session keys need not be odd parity. The enveloped data content type must be Data.

Calling the gsk_read_enveloped_data_msg_extended() routine is equivalent to calling the gsk_read_content_msg() routine, the gsk_read_enveloped_data_content_extended() routine, and the gsk_read_data_content() routine.

The recipient_keys parameter supplies one or more recipient certificates and associated private keys. The gsk_read_enveloped_data_msg_extended() routine will search for a certificate matching one of the message recipients. The private key will be used to decrypt the session key and the session key will then be used to decrypt the enveloped data. If option_flag specifies that key encipherment is to be enforced, then the certificate key usage must allow key encipherment.

No certificate validation is performed by the gsk_read_enveloped_data_msg_extended() routine. It is assumed that the application has already validated the recipient certificates.

These encryption algorithms are supported. Strong encryption may not be available depending upon government export regulations.

- x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - {1.2.840.113549.3.2}
- x509_alg_rc4 - 40-bit and 128-bit RC4 - {1.2.840.113549.3.4}
- x509_alg_desCbcPad - 56-bit DES - {1.3.14.3.2.7}
- x509_alg_desEde3CbcPad - 168-bit 3DES - {1.2.840.113549.3.7}

When executing in FIPS mode, encryption algorithms x509_alg_rc2CbcPad, x509_alg_rc4 and x509_alg_desCbcPad are not supported.
gsk_read_signed_data_content()

Processes PKCS #7 SignedData content information.

Format

```c
#include <gskcms.h>
gsk_status gsk_read_signed_data_content (  
    pkcs_certificates * local_certificates,  
    pkcs_content_info * content_info,  
    gsk_boolean * used_local,  
    pkcs_certificates * msg_certificates,  
    pkcs_certificates * signer_certificates,  
    pkcs_content_info * content_data)
```

Parameters

*local_certificates*
Specifies zero or more X.509 certificates to use when verifying the message signatures. NULL can be specified for this parameter if no local certificates are provided.

*content_info*
Specifies the content information to be processed.

*used_local*
This parameter will be set to TRUE if the signatures were verified using just the certificates supplied by the local_certificates parameter. This parameter will be set to FALSE if any of the signatures were verified using certificates contained within the message.

*msg_certificates*
Returns the X.509 certificates contained within the message. The application should call the `gsk_free_certificates()` routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the message certificates are not needed.

*signer_certificates*
Returns the certificates used to sign the message. The application should call the `gsk_free_certificates()` routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the signer certificates are not needed.

*content_data*
Returns the SignedData content data. The application should call the `gsk_free_content_info()` routine to release the data when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

**[CMSERR_ALG_NOT_SUPPORTED]**
The digest algorithm is not supported.

**[CMSERR_BAD_SIGNATURE]**
Signature is not correct.

**[CMSERR_CONTENT_NOT_SUPPORTED]**
The content type is not SignedData.

**[CMSERR_DIGEST_KEY_MISMATCH]**
The digest algorithm is not supported for the private key type.

**[CMSERR_INCORRECT_KEY_USAGE]**
A signer certificate does not allow digital signature.
Usage

The `gsk_read_signed_data_content()` routine processes PKCS #7 (Cryptographic Message Syntax) SignedData message created by the `gsk_make_signed_data_content()` routine and returns the content data.

The `local_certificates` parameter can supply the signer certificates used to verify the message signatures. If a certificate is not found for a message signer, the `gsk_read_signed_data_content()` routine will attempt to locate the signer certificate in the SignedData message. An error will be returned if the signer certificate cannot be found or if the certificate key usage does not allow digital signature.

No certificate validation is performed by the `gsk_read_signed_data_content()` routine. It is assumed that the application has already validated the local certificates. The certificates contained in the SignedData message will be returned in the `msg_certificates` parameter and the `used_local` parameter will be set to `FALSE` if any of these certificates were used to verify the message signatures. It is the responsibility of the application to validate the message certificates (for example, by calling the `gsk_validate_certificate()` routine for each of the signer certificates).

These digest algorithms are supported:

- **x509_alg_md2Digest**
  MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

- **x509_alg_md5Digest**
  MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

- **x509_alg_sha1Digest**
  SHA-1 digest (RSA and DSA keys only) - {1.3.14.3.2.26}

- **x509_alg_sha224Digest**
  SHA-224 digest (RSA keys only) - {2.16.840.1.101.3.4.2.4}

- **x509_alg_sha256Digest**
  SHA-256 digest (RSA keys only) - {2.16.840.1.101.3.4.2.1}

- **x509_alg_sha384Digest**
  SHA-384 digest (RSA keys only) - {2.16.840.1.101.3.4.2.2}

- **x509_alg_sha512Digest**
  SHA-512 digest (RSA keys only) - {2.16.840.1.101.3.4.2.3}

When executing in FIPS mode, digest algorithms `x509_alg_md2Digest` and `x509_alg_md5Digest` are not supported.
gsk_read_signed_data_content_extended()

Processes PKCS #7 SignedData content information.

Format
#include <gskcms.h>
gsk_status gsk_read_signed_data_content_extended (  
gsk_process_option option_flag  
    pkcs_certificates * local_certificates,  
    pkcs_content_info * content_info,  
    gsk_boolean * used_local,  
    pkcs_certificates * msg_certificates,  
    pkcs_certificates * signer_certificates,  
    gsk_attributes_signers * attributes_signers,  
    pkcs_content_info * content_data)

Parameters

option_flag
Specifies process options to customize process behavior.
- Enforce signing certificate has digital signing capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate digitalSignature.
- Don’t allow zero-length content data.

local_certificates
Specifies zero or more X.509 certificates to use when verifying the message signatures. NULL can be specified for this parameter if no local certificates are provided.

content_info
Specifies the content information to be processed.

used_local
This parameter will be set to TRUE if the signatures were verified using just the certificates supplied by the local_certificates parameter. This parameter will be set to FALSE if any of the signatures were verified using certificates contained within the message.

msg_certificates
Returns the X.509 certificates contained within the message. The application should call the gsk_free_certificates() routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the message certificates are not needed.

signer_certificates
Returns the certificates used to sign the message. The application should call the gsk_free_certificates() routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the signer certificates are not needed.

attributes_signers
Returns the authenticated attributes per signer contained within the message. The application should call the gsk_free_attributes_signers() routine to release the gsk_attributes_signers structure when it is no longer needed. Specify NULL for this parameter if the authenticated attributes per signer are not needed. The set of authenticated attributes returned, omits the content-type and message-digest authenticated attributes as these authenticated attributes must always be present, if any authenticated attributes are present, and are automatically verified by gsk_read_signed_data_content_extended(). The digestAlgorithm field within each gsk_attributes_signer structure returns the digest algorithm originally used for the signer.

content_data
Returns the SignedData content data. The application should call the gsk_free_content_info() routine to release the data when it is no longer needed.
Results
The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_BAD_SIGNATURE]
Signature is not correct.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not SignedData.

[CMSERR_DIGEST_KEY_MISMATCH]
The digest algorithm is not supported for the private key type.

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_SIGNER_NOT_FOUND]
Signer certificate not found.

Usage
The `gsk_read_signed_data_content_extended()` routine processes PKCS #7 (Cryptographic Message Syntax) SignedData message created by the `gsk_make_signed_data_content_extended()` routine and returns the content data and authenticated attributes per signed (if present).

Processing is equivalent to `gsk_read_signed_data_content()`, with these differences:

- The signing certificate key usage need not assert digital signing capabilities depending on `option_flag`.
- Zero length content is acceptable depending on `option_flag`.
- Authenticated attributes and the digest algorithm used to create the signed data per signer, if present, are returned.

The `local_certificates` parameter can supply the signer certificates used to verify the message signatures. If a certificate is not found for a message signer, the `gsk_read_signed_data_content_extended()` routine will attempt to locate the signer certificate in the SignedData message. An error will be returned if the signer certificate cannot be found. An error may optionally be returned if the certificate key usage does not allow digital signature.

No certificate validation is performed by the `gsk_read_signed_data_content_extended()` routine. It is assumed that the application has already validated the local certificates. The certificates contained in the SignedData message will be returned in the `msg_certificates` parameter and the `used_local` parameter will be set to FALSE if any of these certificates were used to verify the message signatures. It is the responsibility of the application to validate the message certificates (for example, by calling the `gsk_validate_certificate()` routine for each of the signer certificates).

These digest algorithms are supported:

- **x509_alg_md2Digest**
  - MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

- **x509_alg_md5Digest**
  - MD5 digest (RSA keys only) - {1.2.840.113549.2.5}
gsk_read_signed_data_content_extended()

**x509_alg_sha1Digest**
SHA-1 digest (RSA and DSA keys only) - {1.3.14.3.2.26}

**x509_alg_sha224Digest**
SHA-224 digest (RSA keys only) - {2.16.840.1.101.3.4.2.4}

**x509_alg_sha256Digest**
SHA-256 digest (RSA keys only) - {2.16.840.1.101.3.4.2.1}

**x509_alg_sha384Digest**
SHA-384 digest (RSA keys only) - {2.16.840.1.101.3.4.2.2}

**x509_alg_sha512Digest**
SHA-512 digest (RSA keys only) - {2.16.840.1.101.3.4.2.3}

If authenticated attributes are returned via the `attributes_signers` parameter, then it is recommended that signing certificates for all signers represented within the gsk_attributes_signers structure should be requested via the `signer_certificates` parameter.

* When executing in FIPS mode, digest algorithms `x509_alg_md2Digest` and `x509_alg_md5Digest` are not supported.
gsk_read_signed_data_msg()

Processes a PKCS #7 SignedData message.

Format

#include <gskcms.h>

gsk_status gsk_read_signed_data_msg (
  pkcs_certificates * local_certificates,
  gsk_buffer * stream,
  gsk_boolean * used_local,
  pkcs_certificates * msg_certificates,
  pkcs_certificates * signer_certificates,
  gsk_buffer * data)

Parameters

local_certificates
  Specifies zero or more X.509 certificates to use when verifying the message signatures. NULL can be
  specified for this parameter if no local certificates are provided.

stream
  Specifies the ASN.1 DER-encoded stream to be processed.

used_local
  This parameter will be set to TRUE if the signatures were verified using just the certificates supplied
  by the local_certificates parameter. This parameter will be set to FALSE if any of the signatures were
  verified using certificates contained within the message.

msg_certificates
  Returns the X.509 certificates contained within the message. The application should call the
  gsk_free_certificates() routine to release the certificates when they are no longer needed. Specify
  NULL for this parameter if the message certificates are not needed.

signer_certificates
  Returns the certificates used to sign the message. The application should call the
  gsk_free_certificates() routine to release the certificates when they are no longer needed. Specify
  NULL for this parameter if the signer certificates are not needed.

data
  Returns the content of the SignedData message. The application should call the gsk_free_buffer()
  routine to release the data when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes
listed in the gskcms.h include file. These are some possible errors:

[ASN_NO_MEMORY]
  Insufficient storage is available.

[ASN_SELECTION_OUT_OF_RANGE]
  Certificate type or version number is not valid.

[CMSERR_ALG_NOT_SUPPORTED]
  The digest algorithm is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
  The message content type is not SignedData or the content of the SignedData message is not
  Data.
gsk_read_signed_data_msg()

**[CMSERR_BAD_SIGNATURE]**
Signature is not correct.

**[CMSERR_DIGEST_KEY_MISMATCH]**
The digest algorithm is not supported for the private key type.

**[CMSERR_INCORRECT_KEY_USAGE]**
A signer certificate does not allow digital signature.

**[CMSERR_NO_CONTENT_DATA]**
The content data length is zero.

**[CMSERR_NO_MEMORY]**
Insufficient storage is available.

**[CMSERR.Signer_NOT_FOUND]**
Signer certificate not found.

**Usage**
The `gsk_read_signed_data_msg()` routine processes a PKCS #7 (Cryptographic Message Syntax) SignedData message created by the `gsk_make_signed_data_msg()` routine and returns the message content. The signed data content type must be Data.

Calling the `gsk_read_signed_data_msg()` routine is equivalent to calling the `gsk_read_content_msg()` routine, the `gsk_read_signed_data_content()` routine, and the `gsk_read_data_content()` routine.

The `local_certificates` parameter can supply the signer certificates used to verify the message signatures. If a certificate is not found for a message signer, the `gsk_read_signed_data_msg()` routine will attempt to locate the signer certificate in the SignedData message. An error will be returned if the signer certificate cannot be found or if the certificate key usage does not allow digital signature.

No certificate validation is performed by the `gsk_read_signed_data_msg()` routine. It is assumed that the application has already validated the local certificates. The certificates contained in the SignedData message will be returned in the `msg_certificates` parameter and the `used_local` parameter will be set to FALSE if any of these certificates were used to verify the message signatures. It is the responsibility of the application to validate the message certificates (for example, by calling the `gsk_validate_certificate()` routine for each of the signer certificates).

These digest algorithms are supported:

- **x509_alg_md2Digest**
  MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

- **x509_alg_md5Digest**
  MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

- **x509_alg_sha1Digest**
  SHA-1 digest (RSA and DSA keys only) - {1.3.14.3.2.26}

- **x509_alg_sha224Digest**
  SHA-224 digest (RSA keys only) - {2.16.840.1.101.3.4.2.4}

- **x509_alg_sha256Digest**
  SHA-256 digest (RSA keys only) - {2.16.840.1.101.3.4.2.1}

- **x509_alg_sha384Digest**
  SHA-384 digest (RSA keys only) - {2.16.840.1.101.3.4.2.2}

- **x509_alg_sha512Digest**
  SHA-512 digest (RSA keys only) - {2.16.840.1.101.3.4.2.3}
When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not supported.
gsk_read_signed_data_msg_extended()

Processes a PKCS #7 SignedData message.

**Format**

```c
#include <gskcms.h>
gsk_status gsk_read_signed_data_msg_extended (
    gsk_process_option
    pkcs_certificates *
    gsk_buffer *
    gsk_boolean *
    pkcs_certificates *
    pkcs_certificates *
    gsk_attributes_signers *
    gsk_buffer *
    option_flag
    local_certificates,
    stream,
    used_local,
    msg_certificates,
    signer_certificates,
    attributes_signers,
    data)
```

**Parameters**

*option_flag*

Specifies process options to customize process behavior.

- Enforce signing certificate has digital signing capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate digitalSignature.
- Don't allow zero-length content data.

*local_certificates*

Specifies zero or more X.509 certificates to use when verifying the message signatures. NULL can be specified for this parameter if no local certificates are provided.

*stream*

Specifies the ASN.1 DER-encoded stream to be processed.

*used_local*

This parameter will be set to TRUE if the signatures were verified using just the certificates supplied by the local_certificates parameter. This parameter will be set to FALSE if any of the signatures were verified using certificates contained within the message.

*msg_certificates*

Returns the X.509 certificates contained within the message. The application should call the `gsk_free_certificates()` routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the message certificates are not needed.

*signer_certificates*

Returns the certificates used to sign the message. The application should call the `gsk_free_certificates()` routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the signer certificates are not needed.

*attributes_signers*

Returns the authenticated attributes per signer contained within the message. The application should call the `gsk_free_attributes_signers()` routine to release the gsk_attributes_signers structure when it is no longer needed. Specify NULL for this parameter if the authenticated attributes per signer are not needed. The set of authenticated attributes returned, omits the content-type and message-digest authenticated attributes as these authenticated attributes must always be present, if any authenticated attributes are present, and are automatically verified by `gsk_read_signed_data_msg_extended()`.

The `digestAlgorithm` field within each gsk_attributes_signer structure returns the digest algorithm originally used for the signer.

*data*

Returns the content of the SignedData message. The application should call the `gsk_free_buffer()` routine to release the data when it is no longer needed.
gsk_read_signed_data_msg_extended()

Results
The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes
listed in the gskcms.h include file. These are some possible errors:

[ASN_NO_MEMORY]
Insufficient storage is available.

[ASN_SELECTION_OUT_OF_RANGE]
Certificate type or version number is not valid.

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The message content type is not SignedData or the content of the SignedData message is not
Data.

[CMSERR_BAD_SIGNATURE]
Signature is not correct.

[CMSERR_DIGEST_KEY_MISMATCH]
The digest algorithm is not supported for the private key type.

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_SIGNER_NOT_FOUND]
Signer certificate not found.

Usage
The gsk_read_signed_data_msg_extended() routine processes a PKCS #7 (Cryptographic Message
Syntax) SignedData message created by the gsk_make_signed_data_msg_extended() routine and
returns the message content and all authenticated attributes (if present). The signed data content type
must be Data.

Processing is equivalent to gsk_read_signed_data_msg(), with these differences:
• The signing certificate key usage need not assert digital signing capabilities depending on option_flag.
• Zero length content is acceptable depending on option_flag.
• Authenticated attributes and the digest algorithm used to create the signed data per signer, if present,
  are returned.

Calling the gsk_read_signed_data_msg_extended() routine is equivalent to calling the
gsk_read_content_msg() routine, the gsk_read_signed_data_content_extended() routine, and the
gsk_read_data_content() routine.

The local_certificates parameter can supply the signer certificates used to verify the message signatures. If
a certificate is not found for a message signer, the gsk_read_signed_data_msg_extended() routine will
attempt to locate the signer certificate in the SignedData message. An error will be returned if the signer
certificate cannot be found. An error may optionally be returned if the certificate key usage does not allow
digital signature.

No certificate validation is performed by the gsk_read_signed_data_msg_extended() routine. It is
assumed that the application has already validated the local certificates. The certificates contained in the
SignedData message will be returned in the *msg_certificates* parameter and the *used_local* parameter will be set to FALSE if any of these certificates were used to verify the message signatures. It is the responsibility of the application to validate the message certificates (for example, by calling the *gsk_validate_certificate()* routine for each of the signer certificates).

These digest algorithms are supported:

- **x509_alg_md2Digest**
  MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

- **x509_alg_md5Digest**
  MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

- **x509_alg_sha1Digest**
  SHA-1 digest (RSA and DSA keys only) - {1.3.14.3.2.26}

- **x509_alg_sha224Digest**
  SHA-224 digest (RSA keys only) - {2.16.840.1.101.3.4.2.4}

- **x509_alg_sha256Digest**
  SHA-256 digest (RSA keys only) - {2.16.840.1.101.3.4.2.1}

- **x509_alg_sha384Digest**
  SHA-384 digest (RSA keys only) - {2.16.840.1.101.3.4.2.2}

- **x509_alg_sha512Digest**
  SHA-512 digest (RSA keys only) - {2.16.840.1.101.3.4.2.3}

If authenticated attributes are returned via the *attributes_signers* parameter, then it is recommended that signing certificates for all signers represented within the gsk_attributes_signers structure should be requested via the *signer_certificates* parameter.

When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not supported.
**gsk_read_wrapped_content()**

Processes wrapped content information.

**Format**

```c
#include <gskcms.h>

gsk_status gsk_read_wrapped_content(
    pkcs_content_info * wrapped_content,
    pkcs_content_info * content_info)
```

**Parameters**

- **wrapped_content**
  Specifies the wrapped content information.

- **content_info**
  Returns the content information. The application should call the `gsk_free_content_info()` routine to release the content information when it is no longer needed.

**Results**

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_CONTENT_NOT_SUPPORTED]**
  The content type is not supported.

- **[CMSERR_NO_MEMORY]**
  Insufficient storage is available.

**Usage**

The `gsk_read_wrapped_content()` routine processes an ASN.1 sequence containing encoded content information and returns the unwrapped content information.
gsk_receive_certificate()

Receives one or more certificates.

Format

```c
#include <gskcms.h>

gsk_status gsk_receive_certificate (  
gsk_buffer * stream,  
pkcs_certificates * certificates)
```

Parameters

**stream**

Specifies the byte stream of the encoded certificate.

**certificate**

Returns the decoded certificates. The application should call the `gsk_free_certificates()` routine to release the certificates when they are no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_BAD_BASE64_ENCODING]**
  - The Base64 encoding of the import file is not correct.

- **[CMSERR_BAD_ENCODING]**
  - The import file format is not recognized.

- **[CMSERR_NO_MEMORY]**
  - Insufficient storage is available.

Usage

The `gsk_receive_certificate()` routine receives one or more X.509 certificates and returns the decoded certificates to the caller.

The supplied stream can represent either the ASN.1 DER encoding for the certificate or the Cryptographic Message Syntax (PKCS #7) encoding for the certificate. This can be either the binary value or the Base64 encoding of the binary value. A Base64 encoded stream must be in the local code page and must include the encoding header and footer lines.

A Base64 DER-encoded sequence must start with the encoding header `'-----BEGIN CERTIFICATE-----'` and end with the encoding footer `'-----END CERTIFICATE-----'`. A Base 64 PKCS #7 signed data message must start with the encoding header `'-----BEGIN CERTIFICATE-----'` and end with the encoding footer `'-----END CERTIFICATE-----'` or must start with the encoding header `'-----BEGIN PKCS #7 SIGNED DATA-----'` and end with the encoding footer `'-----END PKCS #7 SIGNED DATA-----'`.

A DER-encoded certificate stream contains a single X.509 certificate while a PKCS #7 message stream contains one or more certificates. All of the certificates in a PKCS #7 message will be returned to the application for processing.
gsk_replace_record()

Replaces a record in a key or request database.

Format
#include <gskcms.h>

gsk_status gsk_replace_record (  
gsk_handle  
  db_handle,  
gskdb_record *  
  record)

Parameters

db_handle
  Specifies the database handle returned by the gsk_create_database() routine or the gsk_open_database() routine.

record
  Specifies the database record.

Results
The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

- [CMSERR_ALG_NOT_SUPPORTED]
  The signature algorithm is not supported.

- [CMSERR_BACKUP_EXISTS]
  The backup file already exists.

- [CMSERR_BAD_HANDLE]
  The database handle is not valid.

- [CMSERR_BAD_KEY_SIZE]
  The key size is not valid.

- [CMSERR_BAD_LABEL]
  The record label is not valid.

- [CMSERR_BAD_RNG_OUTPUT]
  In FIPS mode, random bytes generation produced duplicate output.

- [CMSERR_DEFAULT_KEY_CHANGED]
  The default key cannot be changed.

- [CMSERR_INCORRECT_DBTYPE]
  The record type is not supported for the database type.

- [CMSERR_IO_ERROR]
  Unable to write record.

- [CMSERR_LABEL_NOT_UNIQUE]
  The record label is not unique.

- [CMSERR_NO_MEMORY]
  Insufficient storage is available.

- [CMSERR_NO_PRIVATE_KEY]
  No private key is provided for a record type that requires a private key.

- [CMSERR_PUBLIC_KEY_CHANGED]
  The subject public key cannot be changed.
gsk_replace_record()

[CMSERR_RECORD_NOT_FOUND]
Record is not found.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_RECTYPE_NOT_VALID]
The record type is not valid.

[CMSERR_SUBJECT_CHANGED]
The subject name cannot be changed.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage
The gsk_replace_record() routine replaces a record in a key or request database. The database must be open for update in order to replace records. The unique record identifier identifies the record to be replaced. Unused and reserved fields in the gskdb_record structure must be initialized to zero. If the record has a private key, the encrypted private key will be generated from the private key supplied in the database record.

The recordType field identifies the database record type as follows:

gskdb_rectype_certificate
The record contains an X.509 certificate.

gskdb_rectype_certKey
The record contains an X.509 certificate and private key.

gskdb_rectype_keyPairTerm
The record contains a PKCS #10 certification request and private key.

The recordFlags field is a bit field with these values:

GSKDB_RECFLAG_TRUSTED
The certificate is trusted.

GSKDB_RECFLAG_DEFAULT
This is the default key

The record label is used as a friendly name for the database entry and is in the local code page. It can be set to any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be set to an empty string.

If the record contains a certificate, the certificate will be validated and the record will not be replaced in the database if the validation check fails. If executing in FIPS mode, only FIPS-approved algorithms and key sizes are supported.

With the exception of the record label, all character strings are specified using UTF-8.

The record type, subject name, and subject public key cannot be changed when replacing a record. In addition, the GSKDB_RECFLAG_DEFAULT flag cannot be changed when replacing a record (call the gsk_set_default_key() routine to change the default record for the database).

The database file is updated as part of the gsk_replace_record() processing. A temporary database file is created using the same name as the database file with "new" appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.
gsk_set_default_key()

Sets the default key.

Format

```c
#include <gskcms.h>
gsk_status gsk_set_default_key (gsk_handle db_handle, gsk_int32 record_id);
```

Parameters

- `db_handle` Specifies the database handle returned by the `gsk_create_database()` routine or the `gsk_open_database()` routine.
- `record_id` Specifies the unique record identifier of the new default key.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- `[CMSERR_BAD_HANDLE]` The database handle is not valid.
- `[CMSERR_INCORRECT_DBTYPE]` The database type does not support a default key.
- `[CMSERR_IO_ERROR]` Unable to write record.
- `[CMSERR_NO_MEMORY]` Insufficient storage is available.
- `[CMSERR_NO_PRIVATE_KEY]` The database record does not contain a private key.
- `[CMSERR_RECORD_NOT_FOUND]` Record is not found.
- `[CMSERR_UPDATE_NOT_ALLOWED]` Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The `gsk_set_default_key()` routine sets the default key for a key database. If the key database already has a default key, the record for the old default key is updated to remove the GSKDB_RECFLAG_DEFAULT flag. The record for the new default key is then updated to add the GSKDB_RECFLAG_DEFAULT flag. The database must be open for update in order to set the default key. An error will be returned if the specified database record does not contain a private key.

The database file is updated as part of the `gsk_set_default_key()` processing. A temporary database file is created using the same name as the database file with "_.new" appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.
gsk_set_directory_enum()

Sets an enumerated value for an LDAP directory.

Format
#include <gskcms.h>

gsk_status gsk_set_directory_enum (gsk_handle directory_handle,
                                   GSKCMS_DIRECTORY_ENUM_ID enum_id,
                                   GSKCMS_DIRECTORY_ENUM_VALUE enum_value)

Parameters

directory_handle
  Specifies an LDAP directory handle returned by gsk_open_directory().

enum_id
  Specifies the directory enumeration identifier.

enum_value
  Specifies the directory enumeration value.

Results
The function return value will be 0 (GSK_OK) if no error is detected. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

[CMSERR_ATTRIBUTE_INVALID_ID]
  The enumeration identifier is not valid or cannot be used with the specified handle.

[CMSERR_ATTRIBUTE_INVALID_ENUMERATION]
  The enumeration value is not valid or cannot be used with the specified enumeration ID.

[CMSERR_BAD_HANDLE]
  The handle is not valid.

Usage
The gsk_set_directory_enum() routine sets the enumerated value for an LDAP directory vector. The LDAP directory must have a valid LDAP handle as initialized using gsk_open_directory()

These enumeration identifiers are supported:

GSKCMS_CRL_SECURITY_LEVEL
  Specifies the level of security to be used when contacting an LDAP server in order to check for revoked certificates in a Certificate Revocation List (CRL). CRLs located will be cached according to the GSK_CRL_CACHE_TIMEOUT setting of the SSL environment. To enforce contact with the LDAP server for each CRL check, CRL caching must be disabled. See “gsk_attribute_set_numeric_value()” on page 62 and Appendix A, “Environment Variables,” on page 481 for additional information on the GSK_CRL_CACHE_TIMEOUT setting.

Three levels of security are available:
  • GSKCMS_CRL_SECURITY_LEVEL_LOW - Certificate validation will not fail if the LDAP server cannot be contacted.
  • GSKCMS_CRL_SECURITY_LEVEL_MEDIUM - Certificate validation requires the LDAP server to be contactable, but does not require a CRL to be defined. This is the default setting.
  • GSKCMS_CRL_SECURITY_LEVEL_HIGH - Certificate validation requires the LDAP server to be contactable, and a CRL to be defined.
gsk_sign_certificate()

Signs an X.509 certificate.

Format

```c
#include <gskcms.h>

gsk_status gsk_sign_certificate (  
    x509_certificate *  
    certificate,  
    pkcs_private_key_info *  
    private_key)
```

Parameters

- **certificate**
  - Specifies the X.509 certificate.

- **private_key**
  - Specifies the private key.

Results

The return status will be zero if the signature is successfully generated. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_ALG_NOT_SUPPORTED]**
  - The signature algorithm is not supported.

- **[CMSERR_BAD_KEY_SIZE]**
  - The key size is not valid.

- **[CMSERR_KEY_MISMATCH]**
  - The supplied key does not match the signature algorithm.

- **[CMSERR_NO_MEMORY]**
  - Insufficient storage is available.

Usage

The `gsk_sign_certificate()` routine will sign an X.509 certificate using the supplied private key. The private key can be an RSA key or a DSA key. If executing in FIPS mode, the minimum key size is 1024 bits. The private key can be an ASN.1-encoded value contained in the privateKey field or an ICSF key label contained in the keyToken field. In either case, the key type must be specified by the privateKeyAlgorithm field.

The signature algorithm is obtained from the signature field of the x509_tbs_certificate structure contained within the x509_certificate structure. The generated signature will be placed in the signatureAlgorithm and signatureValue fields of the x509_certificate structure.

The following signature algorithms are supported:

- **x509_alg_md2WithRsaEncryption**
  - RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

- **x509_alg_md5WithRsaEncryption**
  - RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

- **x509_alg_sha1WithRsaEncryption**
  - RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

- **x509_alg_sha224WithRsaEncryption**
  - RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}
gsk_sign_certificate()

x509_alg_sha256WithRsaEncryption
  RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
  RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
  RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
  Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.
**gsk_sign_crl()**

Signs an X.509 certificate revocation list.

**Format**

```c
#include <gskcms.h>

gsk_status gsk_sign_crl (x509_crl *crl, pkcs_private_key_info *private_key)
```

**Parameters**

- **crl** Specifies the X.509 certificate revocation list.
- **private_key** Specifies the private key.

**Results**

The return status will be zero if the signature is successfully generated. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_ALG_NOT_SUPPORTED]**
  The signature algorithm is not supported.
- **[CMSERR_BAD_KEY_SIZE]**
  The key size is not valid.
- **[CMSERR_KEY_MISMATCH]**
  The supplied key does not match the signature algorithm.
- **[CMSERR_NO_MEMORY]**
  Insufficient storage is available.

**Usage**

The `gsk_sign_crl()` routine will sign an X.509 certificate revocation list using the supplied private key. The private key can be an RSA key or a DSA key. If executing in FIPS mode, the minimum key size is 1024 bits. The private key can be an ASN.1-encoded value contained in the privateKey field or an ICSF key label contained in the keyToken field. In either case, the key type must be specified by the privateKeyAlgorithm field.

The signature algorithm is obtained from the signature field of the x509_tbs_crl structure contained within the x509_crl structure. The generated signature will be placed in the signatureAlgorithm and signatureValue fields of the x509_crl structure.

The following signature algorithms are supported:

- **x509_alg_md2WithRsaEncryption**
  RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}
- **x509_alg_md5WithRsaEncryption**
  RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}
- **x509_alg_sha1WithRsaEncryption**
  RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}
- **x509_alg_sha224WithRsaEncryption**
  RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}
gsk_sign_crl()

x509_alg_sha256WithRsaEncryption
    RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
    RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
    RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
    Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

* When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and
  x509_alg_md5WithRsaEncryption are not supported.
gsk_sign_data()

Signs a data stream.

Format

```c
#include <gskcms.h>

gsk_status gsk_sign_data (x509_algorithm_type sign_algorithm,
                        pkcs_private_key_info * private_key,
                        gsk_boolean is_digest,
                        gsk_buffer * data,
                        gsk_buffer * signature)
```

Parameters

- **sign_algorithm**
  Specifies the signature algorithm.

- **private_key**
  Specifies the private key.

- **is_digest**
  Specify TRUE if the data stream digest has been computed or FALSE if the data stream digest needs to be computed.

- **data**
  Specifies either the data stream digest (is_digest is TRUE) or the data stream (is_digest is FALSE).

- **signature**
  Returns the generated signature. The caller should release the signature buffer when it is no longer needed by calling the `gsk_free_buffer()` routine.

Results

The return status will be zero if the signature is successfully generated. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- **[CMSERR_ALG_NOT_SUPPORTED]**
  The signature algorithm is not supported.

- **[CMSERR_BAD_DIGEST_SIZE]**
  The digest size is not correct.

- **[CMSERR_BAD_KEY_SIZE]**
  The key size is not valid.

- **[CMSERR_BAD_RNG_OUTPUT]**
  In FIPS mode, random bytes generation produced duplicate output.

- **[CMSERR_KEY_MISMATCH]**
  The supplied key does not match the signature algorithm.

- **[CMSERR_NO_MEMORY]**
  Insufficient storage is available.

Usage

The `gsk_sign_data()` routine will generate the signature for a data stream using the supplied private key. The private key can be an RSA key or a DSA key. If executing in FIPS mode, the minimum key size is 1024 bits. The private key can be an ASN.1-encoded value contained in the privateKey field or an ICSF key label contained in the keyToken field. In either case, the key type must be specified by the `privateKeyAlgorithm` field.
gsk_sign_data()

The application can either provide the message digest or have the `gsk_sign_data()` routine compute the message digest.

When the application provides the message digest, the digest length must be correct for the specified signature algorithm. Digest lengths: MD2 and MD5 are 16 bytes; SHA-1 is 20 bytes; SHA-224 is 28 bytes; SHA-256 is 32 bytes; SHA-384 is 48 bytes and SHA-512 is 64 bytes. The supplied digest will be used as-is without any further processing (specifically, for an RSA encryption key, the digest will not be encoded as an ASN.1 DigestInfo sequence before generating the signature).

The following signature algorithms are supported:

- `x509_alg_md2WithRsaEncryption`
  
  RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

- `x509_alg_md5WithRsaEncryption`
  
  RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

- `x509_alg_sha1WithRsaEncryption`
  
  RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

- `x509_alg_sha224WithRsaEncryption`
  
  RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

- `x509_alg_sha256WithRsaEncryption`
  
  RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

- `x509_alg_sha384WithRsaEncryption`
  
  RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

- `x509_alg_sha512WithRsaEncryption`
  
  RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

- `x509_alg_dsaWithSha1`
  
  Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

- `x509_alg_md5WithRsaEncryption`
  
  RSA encryption with combined MD5 and SHA-1 digests

When executing in FIPS mode, signature algorithms `x509_alg_md2WithRSAEncryption` and `x509_alg_md5WithRSAEncryption` are not supported.
gsk_validate_certificate()

Validates an X.509 certificate.

This function is deprecated. Use gsk_validate_certificate_mode() instead.

Format

```
#include <gskcms.h>

gsk_status gsk_validate_certificate (  
gskdb_data_sources * data_sources,  
x509_certificate * subject_certificate,  
gsk_boolean accept_root,  
gsk_int32 * issuer_record_id)
```

Parameters

data_sources
  Specifies the data sources for CA certificates and revocation lists. The data sources are searched in  
  the order they occur in the data source array, so trusted sources should be included before untrusted  
  sources and local sources should be included before remote sources.

subject_certificate
  Specifies the certificate to be validated.

accept_root
  Specify TRUE if a self-signed root certificate is to be accepted without checking the data sources. 
  Specify FALSE if a self-signed root certificate must be found in one of the trusted data sources in 
  order to be accepted.

issuer_record_id
  Returns the record identifier for the issuer certificate used to validate the certificate. The record 
  identifier will be 0 if the issuer certificate is found in a non-database source. Specify NULL for this 
  parameter if the issuer record identifier is not needed.

Results

The return status will be zero if the validation is successful. Otherwise, it will be one of the return codes 
listed in the gskcms.h include file. These are some possible errors:

- [CMSERR_ALG_NOT_SUPPORTED]
  The signature algorithm is not supported.

- [CMSERR_BAD_HANDLE]
  The database handle is not valid.

- [CMSERR_BAD_KEY_SIZE]
  The key size is not valid.

- [CMSERR_BAD_ISSUER_NAME]
  The certificate issuer name is not valid.

- [CMSERR_BAD_SIGNATURE]
  The signature is not correct.

- [CMSERR_CERT_CHAIN_NOT_TRUST]
  The certification chain is not trusted

- [CMSERR_CERTIFICATE_REVOKED]
  The certificate is revoked.

- [CMSERR_EXPIRED]
  The certificate is expired.
gsk_validate_certificate()

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_USAGE]
The issuer certificate does not allow signing certificates

[CMSERR_ISSUER_NOT_CA]
The certificate issuer is not a certification authority.

[CMSERR_ISSUER_NOT_FOUND]
The issuer certificate is not found in one of the data sources.

[CMSERR_NAME_CONSTRAINTS_VIOLATED]
The certificate name is not consistent with the name constraints.

[CMSERR_NAME_NOT_SUPPORTED]
The AuthorityKeyIdentifier extension name is not a directory name.

[CMSERR_NOT_YET_VALID]
The certificate is not yet valid.

[CMSERR_PATH_TOO_LONG]
The certification chain exceeds the maximum allowed by the CA.

[CMSERR_SELF_SIGNED_NOT_FOUND]
A self-signed certificate is not found in a trusted data source

Usage
The gsk_validate_certificate() routine validates an X.509 certificate by performing these checks on the subject certificate:

- The certificate subject name must be either a non-empty distinguished name or an empty distinguished name with a SubjectAltName certificate extension
- An empty subject name is not allowed for a CA certificate
- The certificate issuer name must not be an empty distinguished name
- The CertificatePolicy extension, if present, must not be a critical extension
- The current time must not be earlier than the start of the certificate validity period
- The current time must not be later than the end of the certificate validity period
- The issuer certificate must be a valid CA certificate
- The certificate signature must be correct
- The certificate must not be revoked
- The certification chain must lead to a certificate obtained from a trusted data source
- No certificate in the certification chain can be revoked or expired.

If executing in FIPS mode, only FIPS-approved algorithms and key sizes are supported (see Chapter 4, “System SSL and FIPS 140-2,” on page 13 for more details).

The gsk_validate_certificate() routine will obtain any necessary CA certificates from the supplied data sources. The CA certificate will be validated as described if it is obtained from an untrusted data source. In addition, these checks will be performed on CA certificates when validating the certification chain:

- The BasicConstraints extension, if present, must have the CA indicator set and the path length constraint must not be violated by subordinate certificates in the certification chain
- The NameConstraints extension, if present, must not be violated by the subject certificate
A root certificate is a self-signed certificate and its signature is verified using the public key in the certificate. If accept_root is FALSE, the root certificate must be found in a trusted data source in order to be accepted. If accept_root is TRUE, the self-signed certificate is accepted as long as the signature is correct.

An intermediate certificate or an end-entity certificate is a certificate signed by another entity. Its signature is verified using the public key in the issuer’s certificate. The issuer certificate must be found in one of the supplied data sources. When intermediate CA certificates are used, the certificate chain is validated until an issuer is reached whose certificate is in one of the trusted data sources.

The data sources must contain at least one LDAP directory source or CRL source in order to check for revoked certificates. The CRL distribution point name (or the certificate issuer name if the certificate does not have a CrlDistributionPoints extension) is used as the distinguished name of the LDAP directory entry containing the certificate revocation list (CRL). The CRL distribution point name and CRL issuer name must be X.500 directory names. The BasicConstraints certificate extension determines whether the CA revocation list or the user revocation list is used. An error will be returned if a CRL obtained from an untrusted source cannot be validated.

Security levels for connecting to LDAP directories are based on the GSKCMS_CRL_SECURITY_LEVEL setting. When using the CMS APIs, the GSKCMS_CRL_SECURITY_LEVEL setting can be specified using the `gsk_set_directory_enum()` routine. Security levels can be set to LOW, MEDIUM or HIGH. See "gsk_attribute_set_enum()" on page 59 and Appendix A, "Environment Variables," on page 481 for further information on CRL security level settings.

These data sources are supported:

- `gskdb_source_key_database` - The source is a key database. The handle must be a database handle returned by the `gsk_create_database()` routine, the `gsk_open_database()` routine, or the `gsk_open_keyring()` routine. This is a trusted data source.
- `gskdb_source_directory` - The source is an LDAP directory. The handle must be the directory handle returned by the `gsk_open_directory()` routine. This is an untrusted data source. Any certificate or revocation list obtained from this source will be validated before being accepted. Refer to the `gsk_get_directory_certificates()` and `gsk_get_directory_crls()` routines for more information concerning the use of LDAP directory entries.
- `gskdb_source_trusted_certs` - The source is an array of certificates. This is a trusted data source.
- `gskdb_source_untrusted_certs` - The source is an array of certificates. This is an untrusted data source. Any certificate used from this list will be validated before being accepted.
- `gskdb_source_trusted_crls` - The source is an array of certificate revocation lists. This is a trusted data source.
- `gskdb_source_untrusted_crls` - The source is an array of certificate revocation lists. This is an untrusted data source. Any CRL used from this list will be validated before being accepted.
- `gskdb_source_cert_callback` - The source is the address of a callback routine which will receive control when an issuer certificate is needed. This is a trusted data source. The subject name is passed as an input parameter and the certCallback routine returns an array of one or more certificates with that subject name. The `gsk_validate_certificate()` routine will call the freeCallback routine to release the certificates. The return status should be 0 if no errors are detected. Otherwise it should be one of the error code listed in the gskcms.h include file. The return status should be 0 and the certificate count should be 0 if there are no certificates matching the supplied subject name.
- `gskdb_source_crl_callback` - The source is the address of a callback routine which will receive control when a certificate needs to be checked to see if it has been revoked. The return value should be 0 if the certificate is not revoked. Otherwise it should be one of the error codes defined in the gskcms.h include file.
gsk_validate_certificate_mode()

Validates an X.509 certificate.

Format
#include <gskcms.h>
gsk_status gsk_validate_certificate_mode (
    gskdb_data_sources * data_sources,
x509_certificate * subject_certificate,
gsk_boolean accept_root,
gsk_int32 * issuer_record_id,
GSKCMS_CERT_VALIDATION_MODE validation_mode,
gsk_uint32 arg_count,
...)

Parameters

data_sources
    Specifies the data sources for CA certificates and revocation lists. The data sources are searched in
    the order they occur in the data source array, so trusted sources should be included before untrusted
    sources and local sources should be included before remote sources.

subject_certificate
    Specifies the certificate to be validated.

accept_root
    Specify TRUE if a self-signed root certificate is to be accepted without checking the data sources.
    Specify FALSE if a self-signed root certificate must be found in one of the trusted data sources in
    order to be accepted.

issuer_record_id
    Returns the record identifier for the issuer certificate used to validate the certificate. The record
    identifier will be 0 if the issuer certificate is found in a non-database source. Specify NULL for this
    parameter if the issuer record identifier is not needed.

validation_mode
    Specifies certificate validation mode to customize the policy used for certificate validation.

arg_count
    Specifies the number of optional parameters following the arg_count parameter. Currently no optional
    parameters are supported and arg_count must be set to 0.

Results
The return status will be zero if the validation is successful. Otherwise, it will be one of the return codes
listed in the gskcms.h include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not supported.

[CMSERR_BAD_ARG_COUNT]
Variable argument count is not valid.

[CMSERR_BAD_CRL]
Certificate revocation list cannot be found.

[CMSERR_BAD_EXT_DATA]
Certificate extension data is incorrect.

[CMSERR_BAD_HANDLE]
The database handle is not valid.
gsk_validate_certificate_mode()

- **CMSERR_BAD_KEY_SIZE**: The key size is not valid.
- **CMSERR_BAD_ISSUER_NAME**: The certificate issuer name is not valid.
- **CMSERR_BAD_SIGNATURE**: The signature is not correct.
- **CMSERR_BAD_SUBJECT_NAME**: Subject name is not valid.
- **CMSERR_BAD_VALIDATION_OPTION**: Validation option is not valid.
- **CMSERR_CERT_CHAIN_NOT_TRUSTED**: The certification chain is not trusted.
- **CMSERR_CERTIFICATE_REVOKED**: The certificate is revoked.
- **CMSERR_CRITICAL_EXT_INCORRECT**: Certificate extension has an incorrect critical indicator.
- **CMSERR_DISTRIBUTION_POINTS**: Cannot match CRL distribution points.
- **CMSERR_DUPLICATE_EXTENSION**: Supplied extensions contain a duplicate extension.
- **CMSERR_EXPIRED**: The certificate is expired.
- **CMSERR_EXT_NOT_SUPPORTED**: Certificate extension is not supported.
- **CMSERR_INCORRECT_DBTYPE**: The database type does not support certificates.
- **CMSERR_INCORRECT_KEY_USAGE**: The issuer certificate does not allow signing certificates.
- **CMSERR_ISSUER_NOT_CA**: The certificate issuer is not a certification authority.
- **CMSERR_ISSUER_NOT_FOUND**: The issuer certificate is not found in one of the data sources.
- **CMSERR_LDAP_NOT_AVAILABLE**: LDAP is not available.
- **CMSERR_NAME_CONSTRAINTS_VIOLATED**: The certificate name is not consistent with the name constraints.
- **CMSERR_NAME_NOT_SUPPORTED**: The AuthorityIdentifier extension name is not a directory name.
- **CMSERR_NO_ACCEPTABLE_POLICIES**: Acceptable policy intersection cannot be found.
- **CMSERR_NOT_YET_VALID**: The certificate is not yet valid.
- **CMSERR_PATH_TOO_LONG**: The certification chain exceeds the maximum allowed by the CA.
The `gsk_validate_certificate_mode()` routine validates an X.509 certificate according to the standards defined in RFC2459, Internet X.509 Public Key Infrastructure - Certificate and CRL Profile, or RFC3280, Internet X.509 Public Key Infrastructure - Certificate and Certificate Revocation List (CRL) Profile. Any necessary CA or issuer certificates will be obtained from the supplied data sources. The CA certificate will also be validated according to the previously mentioned Internet standards.

The `validation_mode` parameter determines the Internet standard that the certificate and certificate chain will be validated against. The following validation modes are supported:
- **GSKCMS_CERT_VALIDATION_MODE_2459** – validate the certificate against RFC2459 only.
- **GSKCMS_CERT_VALIDATION_MODE_3280** – validate the certificate against RFC3280 only.
- **GSKCMS_CERT_VALIDATION_MODE_ANY** – attempt to validate the certificate against RFC2459 initially. If that fails, validate against RFC3280.

A root certificate is a self-signed certificate and its signature is verified using the public key in the certificate. If `accept_root` is `FALSE`, the root certificate must be found in a trusted data source in order to be accepted. If `accept_root` is `TRUE`, the self-signed certificate is accepted as long as the signature is correct.

An intermediate certificate or an end-entity certificate is a certificate signed by another entity. Its signature is verified using the public key in the issuer's certificate. The issuer certificate must be found in one of the supplied data sources. When intermediate CA certificates are used, the certificate chain is validated until an issuer is reached whose certificate is in one of the trusted data sources.

The data sources must contain at least one LDAP directory source or CRL source in order to check for revoked certificates. The CRL distribution point name (or the certificate issuer name if the certificate does not have a CrlDistributionPoints extension) is used as the distinguished name of the LDAP directory entry containing the certificate revocation list (CRL). The CRL distribution point name and CRL issuer name must be X.500 directory names. The BasicConstraints certificate extension determines whether the CA revocation list or the user revocation list is used. An error will be returned if a CRL obtained from an untrusted source cannot be validated.

Security levels for connecting to LDAP directories are based on the GSKCMS_CRL_SECURITY_LEVEL setting. When using the CMS APIs, the GSKCMS_CRL_SECURITY_LEVEL setting can be specified using the `gsk_set_directory_enum()` routine. Security levels can be set to LOW, MEDIUM or HIGH. See "gsk_attribute_set_enum()" on page 59, "gsk_set_directory_enum()" on page 322 and Appendix A, "Environment Variables," on page 481 for further information on CRL security level settings.

These data sources are supported:
- `gskdb_source_key_database` - The source is a key database. The handle must be a database handle returned by the `gsk_create_database()` routine, the `gsk_open_database()` routine, or the `gsk_open_keyring()` routine. This is a trusted data source.
- `gskdb_source_directory` - The source is an LDAP directory. The handle must be the directory handle returned by the `gsk_open_directory()` routine. This is an untrusted data source. Any certificate or revocation list obtained from this source will be validated before being accepted. Refer to the `gsk_get_directory_certificates()` and `gsk_get_directory_crls()` routines for more information concerning the use of LDAP directory entries.
- `gskdb_source_trusted_certs` - The source is an array of certificates. This is a trusted data source.
gsk_validate_certificate_mode()

- `gskdb_source_untrusted_certs` - The source is an array of certificates. This is an untrusted data source. Any certificate used from this list will be validated before being accepted.
- `gskdb_source_trusted_crls` - The source is an array of certificate revocation lists. This is a trusted data source.
- `gskdb_source_untrusted_crls` - The source is an array of certificate revocation lists. This is an untrusted data source. Any CRL used from this list will be validated before being accepted.
- `gskdb_source_cert_callback` - The source is the address of a callback routine which will receive control when an issuer certificate is needed. This is a trusted data source. The subject name is passed as an input parameter and the certCallback routine returns an array of one or more certificates with that subject name. The `gsk_validate_certificate_mode()` routine will call the freeCallback routine to release the certificates. The return status should be 0 if no errors are detected. Otherwise it should be one of the error code listed in the `gskcms.h` include file. The return status should be 0 and the certificate count should be 0 if there are no certificates matching the supplied subject name.
- `gskdb_source_crl_callback` - The source is the address of a callback routine which will receive control when a certificate needs to be checked to see if it has been revoked. The return value should be 0 if the certificate is not revoked. Otherwise it should be one of the error codes defined in the `gskcms.h` include file.
gsk_validate_hostname()

Validates a host certificate against the supplied hostname.

Format

```c
#include <gskcms.h>
gsk_status gsk_validate_hostname (
    x509_certificate *       host_certificate,
    const char *             host_name,
    GSKCMS_VALIDATE_HOSTNAME val_option)
```

Parameters

- **host_certificate**
  Specifies the host certificate to be validated.

- **host_name**
  Specifies the fully-qualified host name in the local code page.

- **val_option**
  Specifies validation option to customize the order of the validation process.

Results

The function return value will be 0 (GSK_OK) if the validation is successful. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

- [CMSERR_HOST_NOT_VALID]
  The certificate is not valid for the specified host name.

- [CMSERR_BAD_VALIDATION_OPTION]
  Validation option is not valid.

Usage

The `gsk_validate_hostname()` routine validates the certificate against the specified host name. For successful validation the certificate must contain the specified host name as either the common name (CN) element of the subject name or as a DNS entry for the subject alternate name as indicated by the validation option.

The `val_option` parameter will determine the composition and order of the validation process. A value of:

- GSKCMS_VALIDATE_HOSTNAME_CN will validate the host name against the common name (CN) of the certificate first and then against the DNS entry for the subject alternate name extension if no match is found in the CN.
- GSKCMS_VALIDATE_HOSTNAME_CN_ONLY will validate the host name against the common name (CN) of the certificate only.
- GSKCMS_VALIDATE_HOSTNAME_DNS will validate the host name against the DNS entry in the subject alternate name extension first and, only if that is not present, validate the host name against the common name.
- GSKCMS_VALIDATE_HOSTNAME_DNS_ONLY will validate the host name against the DNS entry in the subject alternate name extension only.

The host name in the certificate can be a fully-qualified name (e.g., 'dcesec4.endicott.ibm.com'), a domain suffix (e.g., '.endicott.ibm.com') or a wildcard name beginning with an asterisk (e.g., '*.endicott.ibm.com'). A case-sensitive comparison is performed between the supplied host name and the host name in the certificate. A fully-qualified name must be the same as the supplied host name. A domain suffix matches any host name with the same suffix but does not match the suffix itself. For example, '*.endicott.ibm.com'
matches 'ldap.dcesec4.endicott.ibm.com' and 'dcesec4.endicott.ibm.com' but does not match
'endicott.ibm.com'. A wildcard name matches any name ending with the characters that follow the asterisk.
A trailing period in a host name will be ignored (e.g., 'dcesec4.endicott.ibm.com.' is the same as
'dcesec4.endicott.ibm.com').

I No other certificate validation is performed. The gsk_validate_certificate_mode() routine should be called
if the certificate itself needs to be validated.
gsk_validate_server()

Format

```c
#include <gskcms.h>

gsk_status gsk_validate_server (  
  x509_certificate * server_certificate,  
  const char * host_name)
```

Parameters

- `server_certificate`
  Specifies the server certificate to be validated.

- `host_name`
  Specifies the fully-qualified server host name in the local code page.

Results

The return status will be zero if the validation is successful. Otherwise, it will be one of the return codes listed in the `gskcms.h` include file. These are some possible errors:

- [CMSERR_HOST_NOT_VALID](#)
  The server certificate is not valid for the specified host name.

Usage

The `gsk_validate_server()` routine validates a server certificate by verifying the host name associated with the server. The server certificate must contain the specified host name as either the common name (CN) element of the subject name or as a DNS entry for the subject alternate name. For other combinations of hostname verification options use `gsk_validate_hostname()`.

The host name in the server certificate can be a fully-qualified name (for example, `'dcesec4.endicott.ibm.com'`), a domain suffix (for example, `'.endicott.ibm.com'`) or a wildcard name beginning with an asterisk (for example, `'*endicott.ibm.com'`). A case-insensitive comparison is performed between the supplied host name and the host name in the server certificate. A fully-qualified name must be the same as the supplied host name. A domain suffix matches any host name with the same suffix but does not match the suffix itself. For example, `'*endicott.ibm.com'` matches `'ldap.dcesec4.endicott.ibm.com'` and `'dcesec4.endicott.ibm.com'` but does not match `'endicott.ibm.com'`. A wildcard name matches any name ending with the characters that follow the asterisk. A trailing period in a host name will be ignored (for example, `'dcesec4.endicott.ibm.com.'` is the same as `dcesec4.endicott.ibm.com`).

No other certificate validation is performed. The `gsk_validate_certificate_mode()` routine should be called if the certificate itself needs to be validated.
gsk_verify_certificate_signature()

Verifies the signature for an X.509 certificate.

Format
#include <gskcms.h>

gsk_status gsk_verify_certificate_signature ( x509_certificate * certificate, x509_public_key_info * key )

Parameters

certificate
  Specifies the decoded certificate returned by the gsk_decode_certificate() routine.

key
  Specifies the public key for the Certification Authority that signed the certificate.

Results
The return status will be zero if the signature is correct. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
  The signature algorithm is not supported.

[CMSERR_BAD_KEY_SIZE]
  The key size is not valid.

[CMSERR_BAD_SIGNATURE]
  The signature is not correct.

[CMSERR_KEY_MISMATCH]
  The supplied key does not match the signature algorithm.

Usage
The gsk_verify_certificate_signature() routine validates an X.509 certificate by computing its signature and then comparing the result to the signature contained in the certificate.

The following signature algorithms are supported:

x509_alg_md2WithRsaEncryption
  RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
  RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
  RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
  RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
  RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
  RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
  RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}
gsk_verify_certificate_signature()

x509_alg_dsaWithSha1
  Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_md5Sha1WithRsaEncryption
  RSA encryption with combined MD5 and SHA-1 digests

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and
x509_alg_md5WithRsaEncryption are not supported.
gsk_verify_crl_signature()

Verifies the signature for an X.509 certificate revocation list.

Format

```c
#include <gskcms.h>
gsk_status gsk_verify_crl_signature(
    x509_crl * crl,
    x509_public_key_info * key)
```

Parameters

crl  Specifies the decoded certificate revocation list returned by the gsk_decode_crl() routine.

key  Specifies the public key for the Certification Authority that signed the certificate revocation list.

Results

The return status will be zero if the signature is correct. Otherwise, it will be one of the return codes listed in the gskcms.h include file. These are some possible errors:

- [CMSERR_ALG_NOT_SUPPORTED] The signature algorithm is not supported.
- [CMSERR_BAD_KEY_SIZE] The key size is not valid.
- [CMSERR_BAD_SIGNATURE] The signature is not correct.
- [CMSERR_KEY_MISMATCH] The supplied key does not match the signature algorithm.

Usage

The gsk_verify_crl_signature() routine validates an X.509 certificate revocation list (CRL) by computing its signature and then comparing the result to the signature contained in the CRL.

The following signature algorithms are supported:

- **x509_alg_md2WithRsaEncryption**
  - RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}
- **x509_alg_md5WithRsaEncryption**
  - RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}
- **x509_alg_sha1WithRsaEncryption**
  - RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}
- **x509_alg_sha224WithRsaEncryption**
  - RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}
- **x509_alg_sha256WithRsaEncryption**
  - RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}
- **x509_alg_sha384WithRsaEncryption**
  - RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}
- **x509_alg_sha512WithRsaEncryption**
  - RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}
gsk_verify_crl_signature()

x509_alg_dsaWithSha1
   Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_md5Sha1WithRSAEncryption
   RSA encryption with combined MD5 and SHA-1 digests

- When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRSAEncryption are not supported.
gsk_verify_data_signature()

Verifies the signature for a data stream.

Format

```
#include <gskcms.h>

gsk_status gsk_verify_data_signature (
    x509_algorithm_type
    x509_public_key_info *
    gsk_boolean
    gsk_buffer *
    gsk_buffer *

Parameters

sign_algorithm
    Specifies the signature algorithm.

key
    Specifies the public key.

is_digest
    Specify TRUE if the data stream digest has been computed or FALSE if the data stream digest needs
to be computed.

data
    Specifies either the data stream digest (is_digest is TRUE) or the data stream (is_digest is FALSE).

signature
    Specifies the data stream signature.

Results
The return status will be zero if the signature is correct. Otherwise, it will be one of the return codes listed
in the gskcms.h include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not supported.

[CMSERR_BAD_DIGEST_SIZE]
The digest size is not correct.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_SIGNATURE]
The signature is not correct.

[CMSERR_KEY_MISMATCH]
The supplied key does not match the signature algorithm.

Usage
The gsk_verify_data_signature() routine validates the signature for a data stream. The public key can be
an RSA key or a DSA key.

The application can either provide the message digest or have the gsk_verify_signed_data() routine
compute the message digest.

When the application provides the message digest, the digest length must be correct for the specified
signature algorithm. Digest lengths: MD2 and MD5 are 16 bytes; SHA-1 is 20 bytes; SHA-224 is 28 bytes;
gsk_verify_data_signature()

SHA-256 is 32 bytes; SHA-384 is 48 bytes and SHA-512 is 64 bytes. The supplied digest will be used as-is without any further processing (specifically, for an RSA encryption key, the digest will not be encoded as an ASN.1 DigestInfo sequence before comparing it with the digest in the signature).

The following signature algorithms are supported:

- **x509_alg_md2WithRsaEncryption**
  - RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}
- **x509_alg_md5WithRsaEncryption**
  - RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}
- **x509_alg_sha1WithRsaEncryption**
  - RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}
- **x509_alg_sha224WithRsaEncryption**
  - RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}
- **x509_alg_sha256WithRsaEncryption**
  - RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}
- **x509_alg_sha384WithRsaEncryption**
  - RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}
- **x509_alg_sha512WithRsaEncryption**
  - RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}
- **x509_alg_dsaWithSha1**
  - Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}
- **x509_alg_md5Sha1WithRsaEncryption**
  - RSA encryption with combined MD5 and SHA-1 digests

The x509_alg_md5Sha1WithRsaEncryption algorithm is a special algorithm used by the SSL protocol. The data signature consists of the MD5 digest over the data followed by the SHA-1 digest over the data for a total digest length of 36 bytes. The digest is encrypted as-is without any further processing.

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.
Chapter 9. Deprecated Secure Sockets Layer APIs

These set of application program interfaces, or APIs, have been superseded by the APIs defined in Chapter 7, “API Reference,” on page 35.

- `gsk_free_memory()` (see page 346)
- `gsk_get_cipher_info()` (see page 347)
- `gsk_get_dn_by_label()` (see page 348)
- `gsk_initialize()` (see page 349)
- `gsk_secure_soc_close()` (see page 353)
- `gsk_secure_soc_init()` (see page 354)
- `gsk_secure_soc_read()` (see page 361)
- `gsk_secure_soc_reset()` (see page 363)
- `gsk_secure_soc_write()` (see page 364)
- `gsk_srb_initialize()` (see page 366)
- `GSKSRBRD()` (see page 367)
- `GSKSRBWT()` (see page 368)
- `gsk_uninitialize()` (see page 369)
- `gsk_user_set()` (see page 370)

Although use of the deprecated set of APIs in this topic is still supported in z/OS Version 1 Release 11, it is strongly recommended that new applications be developed using the set of APIs defined in Chapter 7, “API Reference.”

The deprecated APIs are not being explicitly updated to allow exploitation of new functionality being added to System SSL. If an application wants to exploit new functionality being added, for example TLS V1.1 protocol, the application must be coded to the SSL APIs in Chapter 7, “API Reference.”

In addition, it is strongly recommended that existing applications be modified to make use of the set of APIs defined in Chapter 7, “API Reference,” on page 35. Those modified applications should only use the new APIs, and not a mix of the new APIs and these deprecated APIs. Information about migrating your existing application programs to use the new API set can be found in Chapter 6, “Migrating from Deprecated SSL Interfaces,” on page 33.

IBM may remove support of APIs contained within this topic in a future release.
gsk_free_memory()

Releases storage allocated by the SSL runtime.

Format

```c
#include <gskssl.h>

void gsk_free_memory(
    void * address,
    void * reserved)
```

Parameters

- `address`
  Specifies the address of the storage to be released.

- `reserved`
  Reserved for future use. Specify NULL for this parameter.

Usage

The `gsk_free_memory()` routine releases storage allocated by the SSL runtime.

Related Topics

- `gsk_get_dn_by_label()`
**gsk_get_cipher_info()**

Returns the supported cipher specifications.

**Format**

```c
#include <gskssl.h>

gsk_status gsk_get_cipher_info(
    int level,
    gsk_sec_level *sec_level,
    void *rsvd)
```

**Parameters**

- **level**
  Specifies GSK_LOW_SECURITY to return just the export cipher specifications or GSK_HIGH_SECURITY to return the domestic cipher specifications as well as the export cipher specifications.

- **sec_level**
  Returns the cipher specifications.

- **rsvd**
  Reserved for future use. Specify NULL for this parameter.

**Results**

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. This is a possible error:

**[GSK_BAD_PARAMETER]**

The level value is not valid or a NULL address is specified for sec_level.

**Usage**

The **gsk_get_cipher_info()** routine returns the available cipher specifications. Both domestic and export ciphers will be included if GSK_HIGH_SECURITY is specified while only export ciphers will be included if GSK_LOW_SECURITY is specified. The **gsk_get_cipher_info()** routine can be called at any time and does not require the **gsk_initialize()** routine to be called first.

The SSL V2 cipher specifications returned for GSK_HIGH_SECURITY are "713642" while the SSL V3 cipher specifications are "05043563738392F303132330A1613100D0915120F0C0306020100" if not in FIPS mode, and "3563738392F303132330A1613100D" in FIPS mode. If the Security Level 3 FMID is not installed, the SSL V2 cipher specifications are "642", the SSL V3 cipher specifications are "0915120F0C0306020100" and FIPS mode is not supported.

The SSL V2 cipher specifications returned for GSK_LOW SECURITY are "642" while the SSL V3 cipher specifications are "0915120F0C0306020100" in non-FIPS mode and "" in FIPS mode.

**Related Topics**

- **gsk_secure_soc_init()**
- **gsk_initialize()**
gsk_get_dn_by_label()

Gets the distinguished name for a certificate.

Format

```
#include <gskssl.h>

char * gsk_get_dn_by_label(const char * label)
```

Parameters

- **label**
  - Specifies the key label.

Usage

The `gsk_get_dn_by_label()` routine returns the distinguished name for the certificate associated with the key label. The `gsk_initialize()` routine must be called before the `gsk_get_dn_by_label()` routine can be called. The application should release the returned name when it is no longer needed by calling the `gsk_free_memory()` routine. The return value will be NULL if an error occurred while accessing the key database or when using z/OS PKCS #11 token and multiple certificates exist for the specified label.

Related Topics

- `gsk_initialize()`
- `gsk_secure_soc_init()`
- `gsk_free_memory()`
gsk_initialize()

Initializes the System SSL runtime environment.

Format

```c
#include <gskssl.h>
gsk_status gsk_initialize(
gsk_init_data * init_data)
```

Parameters

`init_data`

Specifies the data used to initialize the SSL runtime environment.

Results

The function return value will be 0 (GSK_OK) if no error is detected. Otherwise, it will be one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

[GSK_ERR_INIT_PARM_NOT_VALID]

An initialization parameter is not valid.

[GSK_ERROR_BAD_MALLOC]

Insufficient storage is available.

[GSK_ERROR_CRYPTO]

Cryptographic error detected.

[GSK_ERROR_LDAP]

Unable to initialize the LDAP client.

[GSK_ERROR_MULTIPLE_LABEL]

Multiple certificates exist for label.

[GSK_ERROR_MULTIPLE_DEFAULT]

Multiple keys are marked as the default.

[GSK_ERROR_PERMISSION_DENIED]

Not authorized to access the key database, key ring or token.

[GSK_INIT_SEC_TYPE_NOT_VALID]

The security type is not valid.

[GSK_INIT_V2_TIMEOUT_NOT_VALID]

The SSL V2 timeout is not valid.

[GSK_INIT_V3_TIMEOUT_NOT_VALID]

The SSL V3 timeout is not valid.

[GSK_KEYFILE_BAD_FORMAT]

Key database or key ring format is not valid.

[GSK_KEYFILE_BAD_PASSWORD]

Key database password is not correct.

[GSK_KEYFILE_IO_ERROR]

Unable to read the key database, key ring or token.

[GSK_KEYFILE_NO_CERTIFICATES]

The key database, key ring or token does not contain any certificates.
gsk_initialize()

[GSK_KEYFILE_OPEN_FAILED]
Unable to open the key database, key ring or token.

[GSK_KEYFILE_PW_EXPIRED]
Key database password is expired.

Usage
The gsk_initialize() routine initializes the System SSL runtime environment for the current process. The gsk_uninitialize() routine should be called to release the SSL environment when it is no longer needed. Multiple calls to gsk_initialize() will cause the existing environment to be released before creating the new environment.

Environment variables are processed along with the gsk_initialize data structures. Information passed in the key database, key ring or token is read as part of the environment initialization. Upon successful completion of gsk_initialize(), the application is ready to begin creating and using secure socket connections.

The gsk_init_data structure contains these fields:

sec_types
Specifies one of these null-terminated character strings:

- "SSLV2" or "SSL20" to use the SSL V2 protocol
- "SSLV3" or "SSL30" to use the SSL V3 protocol
- "TLSV1" or "TLS10" to use the TLS V1.0 protocol
- "SSLV2_OFF" to allow either TLS V1.0 or SSL V3 to be used
- "ALL" to use any supported protocol

When "SSLV2_OFF" is specified the SSL client/server will attempt first to use the TLS V1.0 protocol, before falling back to the most secure protocol supported by its SSL partner, excluding the SSL V2 protocol.

When "ALL" is specified for an SSL client, the client will attempt first to use the TLS V1.0 protocol and will fall-back to the most secure protocol that the server will support, excluding the SSL V2 protocol (the client must explicitly request the SSL V2 protocol if it wants to use this protocol).

When "ALL" is specified for an SSL server, the server will accept any of the supported protocols.

When running in FIPS mode, the minimum requirement is TLS V1.0 protocol. If only the SSL V2 or the SSL V3 protocol is enabled, then a FIPS mode SSL connection is not possible.

keyring
Specifies the name of the key database, SAF key ring or z/OS PKCS #11 token as a null-terminated character string. When both the password and stash file name are NULL, a SAF key ring or PKCS #11 token is used.

The SAF key ring name is specified as "userid/keyring". The current userid is used if the userid is omitted. The user must have READ access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by the user. The user must have UPDATE access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by another user. Note that certificate private keys are not available when using a SAF key ring owned by another user.

The z/OS PKCS #11 token name is specified as *TOKEN*/token-name. *TOKEN* indicates that the specified key ring is actually a token name. The application userid must have READ access to resource USER.token-name in the CRYPTOZ class in order for the certificate and their private keys, if present, to be read.
keyring_pw
Specifies the password for the key database as a null-terminated character string. Specify NULL to indicate no password is provided.

describe keyring_stash
Specifies the name of the password stash file as a null-terminated character string. Specify NULL to indicate no stash file is provided. The password stash file is used if the keyring_pw value is NULL.

V2_session_timeout
Specifies the SSL V2 session cache timeout value in seconds. The valid range is 0 to 100. A short SSL handshake is performed when a cached session exists since the session parameters have already been negotiated between the client and the server.

V3_session_timeout
Specifies the SSL V3 session cache timeout value in seconds. The valid range is 0 to 86400. A short SSL handshake is performed when a cached session exists since the session parameters have already been negotiated between the client and the server.

LDAP_server
Specifies one or more blank-separated LDAP server host names as a null-terminated character string. Each host name can contain an optional port number separated from the host name by a colon. The LDAP server is used for certificate validation. The LDAP server is used only when LDAP_CA_roots is set to GSK_CA_ROOTS_LOCAL_AND_X500 and auth_type is not set to GSK_CLIENT_AUTH_LOCAL or GSK_CLIENT_AUTH_PASSTHRU.

LDAP_port
Specifies the LDAP server port. The default LDAP port will be used if 0 is specified.

LDAP_user
Specifies the distinguished name to use when connecting to the LDAP server and is a null-terminated character string. An anonymous bind will be done if NULL is specified for this field.

LDAP_password
Specifies the password to use when connecting to the LDAP server and is a null-terminated character string. This field is ignored if NULL is specified for LDAP_user.

LDAP_CA_roots
Specifies the location of CA certificates and certificate revocation lists used to validate certificates. When GSK_CA_ROOTS_LOCAL_ONLY is specified, the CA certificates and certificate revocation lists are obtained from the local database. When GSK_CA_ROOTS_LOCAL_AND_X500 is specified, the CA certificates and certificate revocation lists are obtained from the LDAP server if they are not found in the local database. Even when an LDAP server is used, root CA certificates must be found in the local database since the LDAP server is not a trusted data source.

auth_type
Specifies the client authentication type. This field is ignored unless LDAP_CA_roots is set to GSK_CA_ROOTS_LOCAL_AND_X500. The client certificate is not validated when GSK_CLIENT_AUTH_PASSTHRU is specified. The client certificate is validated using just the local database when GSK_CLIENT_AUTH_LOCAL is specified. CA certificates and certificate revocation lists not found in the local database will be obtained from the LDAP server when GSK_CLIENT_AUTH_STRONG or GSK_CLIENT_AUTH_STRONG_OVER_SSL is specified (the local database must still contain the root CA certificates). There is no difference between GSK_CLIENT_AUTH_STRONG and GSK_CLIENT_AUTH_STRONG_OVER_SSL.

Environment variables

Environment variables are processed along with the information passed in the gsk_init_data structure during environment initialization. Also during environment initialization, the key database, key ring or token is read.
gsk_initialize()

The gsk_initialize() routine supports these environment variables:

**GSKV2CACHESIZE**

  Specifies the number of entries in the SSL V2 session cache with a range of 0 to 32000. The value specified by the GSK_V2_SIDCACHE_SIZE environment variable will be used if the GSKV2CACHESIZE variable is not defined. The default value is 256 if neither environment variable is defined.

**GSKV3CACHESIZE**

  Specifies the number of entries in the SSL V3 session cache with a range of 0 to 64000. The value specified by the GSK_V3_SIDCACHE_SIZE environment variable will be used if the GSKV3CACHESIZE variable is not defined. The default value is 512 if neither environment variable is defined. The SSL V3 session cache is used for both the SSL V3 and TLS V1.0 protocols.

The environment variables that will be overridden with information passed in the gsk_init_data structure are:

- GSK_KEYRING_FILE
- GSK_KEYRING_PW
- GSK_KEYRING_STASH
- GSK_LDAP_SERVER
- GSK_LDAP_PASSWORD
- GSK_LDAP_PORT
- GSK_LDAP_USER
- GSK_PROTOCOL_SSLV2
- GSK_PROTOCOL_SSLV3
- GSK_PROTOCOL_TLSV1
- GSK_V2_SESSION_TIMEOUT
- GSK_V3_SESSION_TIMEOUT

**Related Topics**

gsk_secure_soc_init()

gsk_secure_soc_read()

gsk_secure_soc_write()

gsk_secure_soc_close()

gsk_uninitialize()
gsk_secure_soc_close

Closes a secure socket connection.

Format

```c
#include <gskssl.h>

void gsk_secure_soc_close(gsk_soc_data * handle)
```

Parameters

`handle`
Specifies the connection handle returned by the `gsk_secure_soc_init()` routine.

Usage

The `gsk_secure_soc_close()` routine closes a secure connection created by the `gsk_secure_soc_init()` routine. The socket itself is not closed (the application is responsible for closing the socket). The connection can no longer be used for secure communications after calling the `gsk_secure_soc_close()` routine.

Related Topics

gsk_initialize()

gsk_secure_soc_init()

gsk_secure_soc_read()

gsk_secure_soc_write()
gsk_secure_soc_init()

Initializes a secure socket connection.

Format

```
#include <gskssl.h>

gsk_soc_data * gsk_secure_soc_init(gsk_soc_init_data * init_data);
```

Parameters

`init_data`

Specifies the socket connection initialization data.

Results

The function return value will be `0` (GSK_OK) if no error is detected. Otherwise, it will be one of the return codes listed in the gskssl.h include file. These are some possible errors:

- [GSK_ERR_INIT_PARM_NOT_VALID]
  A connection initialization parameter is not valid.

- [GSK_ERROR_BAD_CERT]
  A certificate is not valid.

- [GSK_ERROR_BAD_DATE]
  A certificate is not valid yet or is expired.

- [GSK_ERROR_BAD_MAC]
  Message verification failed.

- [GSK_ERROR_BAD_MALLOC]
  Insufficient storage is available.

- [GSK_ERROR_BAD_MESSAGE]
  Incorrectly-formatted message received from peer application.

- [GSK_ERROR_BAD_PEER]
  Peer application has violated the SSL protocol.

- [GSK_ERROR_BAD_STATE]
  The SSL environment has not been initialized.

- [GSK_ERROR_CRYPTO]
  Cryptographic error detected.

- [GSK_ERROR_INCOMPATIBLE_KEY]
  The certificate key is not compatible with the negotiated cipher suite.

- [GSK_ERROR_IO]
  I/O error communicating with peer application.

- [GSK_ERROR_LDAP]
  An LDAP error is detected.

- [GSK_ERROR_LDAP_NOT_AVAILABLE]
  The LDAP server is not available.

- [GSK_ERROR_NO_CIPHERS]
  No cipher specifications.
Usage

The `gsk_secure_soc_init()` routine initializes a secure socket connection. The `gsk_initialize()` routine must be called before any secure socket connections can be initialized. After the connection has been initialized, it can be used for secure data transmission using the `gsk_secure_soc_read()` and `gsk_secure_soc_write()` routines. The `gsk_secure_soc_close()` routine should be called to close the connection when it is no longer needed. The `gsk_secure_soc_close()` routine should not be called if an error is returned by the `gsk_secure_soc_init()` routine.

Before calling the `gsk_secure_soc_init()` routine, the application must create a connected socket. For a client, this means calling the `socket()` and `connect()` routines. For a server, this means calling the `socket()`, `listen()`, and `accept()` routines. However, SSL does not require the use of TCP/IP for the communications layer. The socket descriptor can be any integer value that is meaningful to the application. The application must provide its own socket routines if it is not using TCP/IP.

An SSL handshake is performed as part of the processing of the `gsk_secure_soc_init()` routine. This establishes the server identity and optionally the client identity. It also negotiates the cryptographic parameters to be used for the connection.

- The server certificate can use either RSA or DSA as the public/private key algorithm. In FIPS mode, the
- RSA or DSA key size must be at least 1024 bits. An RSA certificate can be used with an RSA, fixed Diffie-Hellman or ephemeral Diffie-Hellman key exchange. A DSA certificate can be used with either a fixed
or ephemeral Diffie-Hellman key exchange. In FIPS mode, the Diffie-Hellman key size must be at least 2048 bits. If the server's certificate contains a key usage extension during the SSL handshake, it must allow key usage as follows:

- RSA certificates using export restricted ciphers (40-bit RC4 encryption and 40-bit RC2 encryption) with a public key size greater than 512 bits must allow digital signature. If operating in FIPS mode, export restricted ciphers cannot be selected.
- RSA or DSA certificates using fixed Diffie-Hellman key exchange must allow key agreement.
- Other RSA certificates must allow key encipherment.
- DSA certificates using ephemeral Diffie-Hellman key exchange must allow digital signature.

System SSL does not honor Verisign Global Server ID certificates. When specified, System SSL will utilize these certificates as any other certificate when determining the encryption cipher to be used for the SSL session.

The client certificate must support digital signatures. This means the certificate key usage extension (if any) must allow digital signature. The key algorithm can be either the RSA encryption algorithm or the Digital Signature Standard algorithm (DSA).

The SSL server always provides its certificate to the SSL client as part of the handshake. Depending upon the server handshake type, the server may ask the client to provide its certificate. The key label stored in the connection is used to retrieve the certificate from the key database, key ring or token. The default key will be used if no label is set. The key record must contain both an X.509 certificate and a private key.

These SSL V2 cipher specifications are supported in non-FIPS mode only:

- "1" = 128-bit RC4 encryption with MD5 message authentication (128-bit secret key)
- "2" = 128-bit RC4 export encryption with MD5 message authentication (40-bit secret key)
- "3" = 128-bit RC2 encryption with MD5 message authentication (128-bit secret key)
- "4" = 128-bit RC2 export encryption with MD5 message authentication (40-bit secret key)
- "6" = 56-bit DES encryption with MD5 message authentication (56-bit secret key)
- "7" = 168-bit Triple DES encryption with MD5 message authentication (168-bit secret key)

These SSL V3 cipher specifications are supported in non-FIPS mode only:

- "00" = No encryption or message authentication and RSA key exchange
- "01" = No encryption with MD5 message authentication and RSA key exchange
- "02" = No encryption with SHA-1 message authentication and RSA key exchange
- "03" = 40-bit RC4 encryption with MD5 message authentication and RSA key exchange
- "04" = 128-bit RC4 encryption with MD5 message authentication and RSA key exchange
- "05" = 128-bit RC4 encryption with SHA-1 message authentication and RSA key exchange
- "06" = 40-bit RC2 encryption with MD5 message authentication and RSA key exchange
- "09" = 56-bit DES encryption with SHA-1 message authentication and RSA key exchange
- "0C" = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate
- "0F" = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "12" = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate
- "15" = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate

These SSL V3 cipher specifications are supported in FIPS mode and non-FIPS mode:
The client sends a list of ciphers it supports during the SSL handshake. The server application uses this list, and the defined ciphers supported by the server, to determine the cipher to be used during the SSL handshake. This selection is done by looking through the server's cipher list for a match in the client's list. The first matching cipher is used.

Environment variables

Environment variables are processed along with the information passed in the gsk_init_data structure during environment initialization. Also during environment initialization, the key database, key ring or token is read.

The environment variables that will be overridden by non-NULL values in the gsk_soc_init_data structure are:

- GSK_KEY_LABEL
- GSK_V2_CIPHER_SPECS
- GSK_V3_CIPHER_SPECS

The gsk_soc_init_data structure contains these fields:

- **fd** Specifies the socket descriptor for the secure connection. The socket must remain open until after the `gsk_secure_soc_close()` routine has been called to close the secure connection.

- **hs_type** Specifies the desired handshake type as follows:
gsk_secure_soc_init()

**GSK_AS_CLIENT**
Performs a client SSL handshake

**GSK_AS_CLIENT_NO_AUTH**
Performs a client SSL handshake but do not provide a client certificate to the SSL server

**GSK_AS_SERVER**
Performs a server SSL handshake

**GSK_AS_SERVER_WITH_CLIENT_AUTH**
Performs a server SSL handshake with client authentication

**DName**
Specifies either the distinguished name or the key label of the local certificate. Specify NULL to use the default key for the key database, key ring or token.

**sec_type**
Returns the selected security protocol as "SSLV2", "SSLV3", or "TLSV1". This is a static string and must not be modified or freed by the application.

**cipher_specs**
Specifies the SSL V2 cipher specifications as a null-terminated string consisting of 1 or more 1-character values. Specify NULL to use the default cipher specifications ("713642" if Security Level 3 FMID encryption is enabled and "642" otherwise). Valid cipher specifications that are not supported due to the installed cryptographic level will be skipped when the connection is initialized.

The SSL V2 protocol can only be used when executing in non-FIPS mode.

**v3cipher_specs**
Specifies the SSL V3 cipher specifications as a null-terminated string consisting of 1 or more 2-character values. Specify NULL to use the default cipher specifications ("050435363738392F303132330A1613100D0915120F0C0306020100" if Security Level 3 FMID is installed and in non-FIPS mode, "35363738392F303132330A1613100D" if Security Level 3 FMID is installed and in FIPS mode, and "0915120F0C0306020100" otherwise). The SSL V3 cipher specifications are used for both the SSL V3 and TLS V1.0 protocols. Valid cipher specifications that are not supported due to the installed cryptographic level will be skipped when the connection is initialized. The SSL V3 protocol can only be used when executing in non-FIPS mode.

**skread**
Specifies the address of the read routine used during the SSL handshake. See "gsk_attribute_set_callback()" on page 55 for additional information on the I/O callback routines.

**skwrite**
Specifies the address of the write routine used during the SSL handshake. See "gsk_attribute_set_callback()" on page 55 for additional information on the I/O callback routines.

**cipherSelected**
Returns the selected cipher for the SSL V2 protocol as a 3-byte binary value:
- 0x010080 = 128-bit RC4 encryption with MD5 message authentication
- 0x020080 = 128-bit RC4 export encryption with MD5 message authentication
- 0x030080 = 128-bit RC2 encryption with MD5 message authentication
- 0x040080 = 128-bit RC2 export encryption with MD5 message authentication
- 0x060040 = 56-bit DES encryption with MD5 message authentication
- 0x0700c0 = 168-bit Triple DES encryption with MD5 message authentication

**v3cipherSelected**
Returns the selected cipher for the SSL V3 or TLS V1.0 protocol as a 2-byte character value with no string delimiter:
- "00" = No encryption or message authentication
- "01" = No encryption with MD5 message authentication and RSA key exchange
- "02" = No encryption with SHA-1 message authentication and RSA key exchange
gsk_secure_soc_init()

- "03" = 40-bit RC4 encryption with MD5 message authentication and RSA key exchange
- "04" = 128-bit RC4 encryption with MD5 message authentication and RSA key exchange
- "05" = 128-bit RC4 encryption with SHA-1 message authentication and RSA key exchange
- "06" = 40-bit RC2 encryption with MD5 message authentication and RSA key exchange
- "09" = 56-bit DES encryption with SHA-1 message authentication and RSA key exchange
- "0A" = 168-bit Triple DES encryption with SHA-1 message authentication and RSA key exchange
- "0C" = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate
- "0D" = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate
- "0F" = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "10" = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "12" = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate
- "13" = 168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate
- "15" = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
- "16" = 168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
- "2F" = 128-bit AES encryption with SHA-1 message authentication and RSA key exchange
- "30" = 128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate
- "31" = 128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "32" = 128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate
- "33" = 128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
- "35" = 256-bit AES encryption with SHA-1 message authentication and RSA key exchange
- "36" = 256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate
- "37" = 256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "38" = 256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate
- "39" = 256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate

failureReasonCode
Returns the gsk_secure_soc_init() error code.

cert_info
Returns peer certificate information. The application must not modify or free this information.

gsk_data
This field is ignored. The key database information is set when gsk_initialize() is called.
gsk_secure_soc_init()

Related Topics

gsk_initialize()

gsk_secure_soc_write()

gsk_secure_soc_read()

gsk_secure_soc_close()

gsk_get_dn_by_label()

gsk_get_cipher_info()

gsk_secure_soc_reset()
gsk_secure_soc_read()

Reads data using a secure socket connection.

Format

```c
#include <gskssl.h>

int gsk_secure_soc_read(gsk_soc_data * soc_handle,
                        void * buffer,
                        int size)
```

Parameters

- `soc_handle` Specifies the connection handle returned by the `gsk_secure_soc_init()` routine.
- `buffer` Specifies the buffer to receive the data read from the secure socket connection. The maximum amount of data returned by `gsk_secure_soc_read()` is 16384 (16K) bytes. If the SSL V2 protocol is used, then the maximum length is 16384 minus the length of the SSL protocol headers.
- `size` Specifies the size of the supplied buffer.

Results

The function return value will be the number of bytes read if no error is detected. Otherwise, it will be a negative value representing one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

- `[GSK_ERROR_BAD_BUFFER_SIZE]` The buffer address or buffer size is not valid.
- `[GSK_ERROR_BAD_MALLOC]` Insufficient storage is available.
- `[GSK_ERROR_BAD_MESSAGE]` Incorrectly-formatted message received from peer application.
- `[GSK_ERROR_BAD_PEER]` Peer application has violated the SSL protocol.
- `[GSK_ERROR_BAD_SSL_HANDLE]` The connection handle is not valid.
- `[GSK_ERROR_CONNECTION_ACTIVE]` A read request is already active for the connection.
- `[GSK_ERROR_CRYPTO]` Cryptographic error detected.
- `[GSK_ERROR_IO]` I/O error communicating with peer application.
- `[GSK_ERROR_SOCKET_CLOSED]` Socket connection closed by peer application.
- `[GSK_ERROR_WOULD_BLOCK]` A complete SSL record is not available.
gsk_secure_soc_read()

[GSK_ERROR_WOULD_BLOCK_WRITE]
An SSL handshake is in progress but data cannot be written to the socket.

Usage
The gsk_secure_soc_read() routine reads data from a secure socket connection and returns it in the application buffer. SSL is a record-based protocol and a single call will never return more than a single SSL record. The maximum amount of data returned by gsk_secure_soc_read() is 16384 (16K) bytes. If the SSL V2 protocol is used, then the maximum length is 16384 minus the length of the SSL protocol headers. The application can read an entire SSL record in a single call by supplying a buffer large enough to contain the record. Otherwise, multiple calls will be required to retrieve the entire SSL record.

SSL supports multiple threads but only one thread at a time can call the gsk_secure_soc_read() routine for a given connection handle. Multiple concurrent threads can call gsk_secure_soc_read() as long as each thread has its own connection handle.

SSL supports sockets in blocking mode and in non-blocking mode. When a socket is in non-blocking mode and a complete SSL record is not available, gsk_secure_soc_read() will return with GSK_ERROR_WOULD_BLOCK. No data will be returned in the application buffer when GSK_ERROR_WOULD_BLOCK is returned. The application should call gsk_secure_soc_read() again when there is data available to be read from the socket.

The peer application can initiate an SSL handshake sequence after the connection is established. If this is done and the socket is in non-blocking mode, it is possible for gsk_secure_soc_read() to return with GSK_ERROR_WOULD_BLOCK_WRITE. This indicates that an SSL handshake is in progress and the application should call gsk_secure_soc_read() again when data can be written to the socket. No data will be returned in the application buffer when GSK_ERROR_WOULD_BLOCK_WRITE is returned.

The application should not read data directly from the socket since this can cause SSL protocol errors if the application inadvertently reads part of an SSL record. If the application must read data from the socket, it is responsible for synchronizing this activity with the peer application so that no SSL records are sent while the application is performing its own read operations.

Related Topics

gsk_initialize()

gsk_secure_soc_init()

gsk_secure_soc_write()

gsk_secure_soc_close()
gsk_secure_soc_reset()

Resets the session keys for a secure connection.

Format

```c
#include <gskssl.h>

gsk_status gsk_secure_soc_reset(gsk_soc_data * soc_handle)
```

Parameters

- `soc_handle`
  - Specifies the connection handle returned by the `gsk_secure_soc_init()` routine.

Results

The function return value will be 0 (GSK_OK) if no error is detected. Otherwise, it will be one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

- **[GSK_ERROR_BAD_MALLOC]**
  - Insufficient storage is available.

- **[GSK_ERROR_BAD_SSL_HANDLE]**
  - The connection handle is not valid.

- **[GSK_ERROR_CONNECTION_CLOSED]**
  - The connection was closed by the peer application.

- **[GSK_ERROR_IO]**
  - I/O error communicating with peer application.

- **[GSK_ERROR_NOT_SSLV3]**
  - The session is not using the SSL V3 or TLS V1.0 protocol.

- **[GSK_ERROR_SOCKET_CLOSED]**
  - Socket connection closed by peer application.

Usage

The `gsk_secure_soc_reset()` routine generates new session keys for the connection. A full SSL handshake will be performed if the session has expired. Otherwise a short SSL handshake will be performed. The `gsk_secure_soc_reset()` routine can be called only for a session using the SSL V3 or TLS V1.0 protocol. The `gsk_secure_soc_reset()` routine initiates the SSL handshake but does not wait for it to complete. Any pending handshake messages will be processed when the `gsk_secure_soc_read()` routine is called to process incoming data.

Related Topics

- `gsk_secure_soc_init()`
gsk_secure_soc_write()

Writes data using a secure socket connection.

Format

```c
#include <gskssl.h>

int gsk_secure_soc_write(
    gsk_soc_data * soc_handle,
    void * buffer,
    int length)
```

Parameters

- `soc_handle`
  Specifies the connection handle returned by the `gsk_secure_soc_init()` routine.

- `buffer`
  Specifies the buffer containing the data to write to the secure socket connection.

- `length`
  Specifies the amount to write.

Results

The function return value will be the number of bytes written if no error is detected. Otherwise, it will be a negative value representing one of the return codes listed in the `gskssl.h` include file. These are some possible errors:

- `[GSK_ERROR_BAD_BUFFER_SIZE]`
  The buffer address or buffer size is not valid.

- `[GSK_ERROR_BAD_MALLOC]`
  Insufficient storage is available.

- `[GSK_ERROR_BAD_SSL_HANDLE]`
  The connection handle is not valid.

- `[GSK_ERROR_CONNECTION_ACTIVE]`
  A write request is already active for the connection.

- `[GSK_ERROR_CONNECTION_CLOSED]`
  A close notification alert has been sent for the connection.

- `[GSK_ERROR_CRYPTO]`
  Cryptographic error detected.

- `[GSK_ERROR_IO]`
  I/O error communicating with peer application.

- `[GSK_ERROR_SOCKET_CLOSED]`
  Socket connection closed by peer application.

- `[GSK_ERROR_WOULD_BLOCK]`
  The SSL record cannot be written to the socket due to an EWOULDBLOCK condition.

Usage

The `gsk_secure_soc_write()` routine writes data to a secure socket connection. SSL is a record-based protocol with a maximum record length of 16384 bytes. If the SSL V2 protocol is used, then the maximum length is 16384 minus the length of the SSL protocol headers. Application data larger than the size of an SSL record will be sent using multiple records.
SSL supports multiple threads but only one thread at a time can call the `gsk_secure_soc_write()` routine for a given connection handle. Multiple concurrent threads can call `gsk_secure_soc_write()` as long as each thread has its own connection handle.

SSL supports sockets in blocking mode and in non-blocking mode. When a socket is in non-blocking mode and the SSL record cannot be written to the socket, `gsk_secure_soc_write()` will return with GSK_ERROR_WOULD_BLOCK. The application must call `gsk_secure_soc_write()` again when the socket is ready to accept more data, specifying the same buffer address and buffer size as the original request. A new write request must not be initiated until the pending write request has been completed as indicated by a return value of 0.

The application should not write data directly to the socket since this can cause SSL protocol errors if the application inadvertently intermixes its data with SSL protocol data. If the application must write data to the socket, it is responsible for synchronizing this activity with the peer application so that application data is not intermixed with SSL data.

**Related Topics**

- `gsk_initialize()`
- `gsk_secure_soc_init()`
- `gsk_secure_soc_read()`
- `gsk_secure_soc_close()`
**gsk_srb_initialize()**

Initializes SRB support.

**Format**

```c
#include <gskssl.h>

gsk_status gsk_srb_initialize (int num_tasks)
```

**Parameters**

- `num_tasks`
  
  Specifies the maximum number of service tasks and must be greater than 0.

**Results**

The function return value will be 0 (GSK_OK) if no error is detected. Otherwise, it will be one of the return codes listed in the gskssl.h include file. These are some possible errors:

- **[GSK_ERR_INIT_PARM_NOT_VALID]**
  
  The number of tasks parameter is not valid.

- **[GSK_ERROR_BAD_STATE]**
  
  The SSL environment is not initialized.

- **[GSK_SRB_INIT_ESTAEX]**
  
  Unable to establish ESTAE exit.

- **[GSK_SRB_INIT_NOT_APF]**
  
  The application is not APF authorized.

- **[GSK_SRB_INIT_THREAD_CREATE]**
  
  Unable to create a thread.

**Usage**

The **gsk_srb_initialize()** routine will initialize the SRB (Service Request Block) support. The application must be APF-authorized in order to use SRB mode. The **gsk_srb_initialize()** routine must be called after the **gsk_initialize()** routine and before any calls to the GKSRSRBD and GKSRSRWWT routines.

The SRB support provided by System SSL is a mode converter which allows an SSL read or write operation to be initiated in SRB mode but processed in TASK mode. This is necessary because SRB mode is not supported by many of the functions invoked by System SSL while processing a read or write request.

The **gsk_srb_initialize()** routine creates a monitor thread and the first service thread. Additional threads are created as needed up to the maximum number of threads specified by the `num_tasks` parameter. The threads run in FIPS mode if FIPS mode was set by a call to **gsk_fips_state_set()**. These threads will be destroyed and SRB mode support will be terminated when the **gsk_uninitialize()** routine is called.

Refer to the z/OS Authorized Assembler Services Guide for more information about service request blocks.

**Related Topics**

- GKSRSRBD
- GKSRSRWWT
GSKSRBRD

Reads from a secure connection in SRB mode.

Format

LOAD  EP=GSKSRBRD
LR    15,0
CALL  (15), (SOCHNDLE, BUFPR, BUFSIZE, RSNCODE)

Parameters

SOCHNDLE
Specifies a 4-byte word containing the gsk_soc_data address returned by the gsk_secure_soc_init() routine.

BUFPR
Specifies a 4-byte word containing the address of the data buffer.

BUFSIZE
Specifies a 4-byte word containing the length of the data buffer.

RSNCODE
Specifies a 4-byte word which will contain the reason code if an error is detected. In most cases, this will be the errno value at the completion of the read request.

Results
The return value will be the number of bytes read if no error is detected. Otherwise, it will be a negative value representing one of the return codes listed in the gskssl.h include file. Refer to the description of the gsk_secure_soc_read() routine for more information.

Usage
The GSKSRBRD routine is called to read from a secure connection in SRB mode. The gsk_srb_initialize() routine must have been called previously to initialize the SRB support. All of the parameters must be in the application storage key and must reside in the primary address space. The GSKSRBRD routine will pass the read request to one of the SRB service tasks. The service task will then call the gsk_secure_soc_read() routine. The GSKSRBRD routine will not return until the gsk_secure_soc_read() routine has completed.

Related Topics
GSKSRBWT

gsk_initialize()

gsk_srb_initialize()

gsk_secure_soc_init()

gsk_secure_soc_write()

gsk_secure_soc_close()
GSKSRBWT

GSKSRBWT

Writes to a secure connection in SRB mode.

Format

```
LOAD   EP=GSKSRBRD
LR     15,0
CALL   (15), (SOCHNDLE, BUFPTR, BUFSIZE, RSNCODE)
```

Parameters

**SOCHNDLE**

Specifies a 4-byte word containing the gsk_soc_data address returned by the \texttt{gsk\_secure\_soc\_init()} routine.

**BUFPTR**

Specifies a 4-byte word containing the address of the data buffer.

**BUFSIZE**

Specifies a 4-byte word containing the length of the data buffer.

**RSNCODE**

Specifies a 4-byte word which will contain the reason code if an error is detected. In most cases, this will be the \texttt{errno} value at the completion of the read request.

Results

The return value will be the number of bytes written if no error is detected. Otherwise, it will be a negative value representing one of the return codes listed in the \texttt{gskssl.h} include file. Refer to the description of the \texttt{gsk\_secure\_soc\_write()} routine for more information.

Usage

The GSKSRBWT routine is called to write to a secure connection in SRB mode. The \texttt{gsk\_srb\_initialize()} routine must have been called previously to initialize the SRB support. All of the parameters must be in the application storage key and must reside in the primary address space. The GSKSRBWT routine will pass the write request to one of the SRB service tasks. The service task will then call the \texttt{gsk\_secure\_soc\_write()} routine. The GSKSRBWT routine will not return until the \texttt{gsk\_secure\_soc\_write()} routine has completed.

Related Topics

GSKSRBRD

gsk\_initialize()

gsk\_srb\_initialize()

gsk\_secure\_soc\_init()

gsk\_secure\_soc\_write()

gsk\_secure\_soc\_close()
gsk_uninitialize()

Terminates the SSL environment.

**Format**

```c
#include <gskssl.h>

gsk_status gsk_uninitialize ( void )
```

**Parameters**

There are no parameters.

**Results**

The function return value will be 0 (GSK_OK) if no error is detected. Otherwise, it will be one of the return codes listed in the gskssl.h include file. This is a possible error:

[GSK_ERROR_CLOSE_FAILED]

An error occurred while closing the environment.

**Usage**

The `gsk_uninitialize()` routine will close the SSL environment created by the `gsk_initialize()` routine. New SSL connections cannot be initiated after calling the `gsk_uninitialize()` routine until the `gsk_initialize()` routine is called to initialize a new SSL environment. All resources allocated for the environment will be released unless there are active SSL connections still using the environment. If there are active connections, the environment will not be actually closed until the last connection has been closed.

**Related Topics**

gsk_initialize()

gsk_secure_soc_init()
**gsk_user_set()**

Sets an application callback.

**Format**

```c
#include <gskssl.h>

gsk_status gsk_user_set(
    gsk_user_set_fid set_id,
    void * set_data,
    void * reserved)
```

**Parameters**

- **set_id**
  Specifies the set function identifier.

- **set_data**
  Specifies the address of the set data.

- **reserved**
  Specify NULL for this parameter.

**Results**

The function return value will be 0 (GSK_OK) if no error is detected. Otherwise, it will be one of the return codes listed in the gskssl.h include file. These are some possible errors:

- **[GSK_BAD_PARAMETER]**
  A parameter is not valid.

- **[GSK_ERROR_BAD_STATE]**
  The SSL environment has not been initialized.

**Usage**

The `gsk_user_set()` routine will set or reset an application callback. The `gsk_initialize()` routine must be called before the `gsk_user_set()` routine can be called.

These set function identifiers are supported:

- **[GSK_SET_SIDCACHE_CALLBACK]**
  This function sets the session identifier cache callback. The set data is the address of the `gsk_sidcache_callback` structure. The application session identifier cache will be used only for SSL servers (the internal cache is always used for SSL clients). This sets the session identifier cache for existing connections as well as new connections created by the `gsk_secure_soc_init()` routine.

  The routine specified by the `Get` entry is called to retrieve an entry from the session identifier cache. The `session_id` parameter is the session identifier, the `session_id_length` parameter is the length of the session identifier, and the `ssl_version` parameter is the SSL protocol version number (GSK_SSLVERSION_V2 or GSK_SSLVERSION_V3). The function return value is the address of the session data buffer or NULL if an error is detected. The `FreeDataBuffer` routine will be called to release the session data buffer when it is no longer needed by the SSL runtime.

  ```c
  gsk_data_buffer * Get (
      const unsigned char * session_id,
      unsigned int session_id_length,
      gsk_sslversion ssl_version)
  ```

  The routine specified by the `Put` entry is called to store an entry in the session identifier cache. The `ssl_session_data` parameter is the session data, the `session_id` parameter is the session
identifier, the session_id_length parameter is the length of the session identifier, and the
ssl_version parameter is the SSL protocol version number (GSK_SSLVERSION_V2 or
GSK_SSLVERSION_V3). The function return value is ignored and can be a NULL address. The
callback routine must make its own copy of the session data since the SSL structure will be
released when the connection is closed.

```c
void Delete (  
    const unsigned char * session_id,
    unsigned int session_id_length,
    gsk_sslversion ssl_version)
```

The routine specified by the Delete entry is called to remove an entry from the session identifier
cache. The session_id parameter is the session identifier, the session_id_length parameter is the
length of the session identifier, and the ssl_version parameter is the SSL protocol version number
(GSK_SSLVERSION_V2 or GSK_SSLVERSION_V3).

```c
void FreeDataBuffer (  
    gsk_data_buffer * ssl_session_data)
```

The routine specified by the FreeDataBuffer entry is called to release the data buffer returned by
the Get routine.

```c
[gSK_RESET_SIDCACHE_CALLBACK]
This function resets the session identifier cache callback. The internal session identifier cache will
be used instead of an application session identifier cache. This resets the session identifier cache
for existing connections as well as new connections created by the gsk_secure_soc_init() routine.

[gSK_SET_GETPEER_CALLBACK]
This function sets the peer identification callback. The peer identification callback returns the 32-bit
network identifier for the remote partner. The fd parameter is the socket descriptor specified when
the connection was initialized. The peer identification routine will be called for new connections
created by gsk_secure_soc_init() but will not be called for existing connections.

```c
unsigned long io_getpeerid (  
    int fd)
```

[gSK_RESET_GETPEER_CALLBACK]
This function resets the peer identification callback. The internal peer identification routine will be
used instead of the application routine. This applies to new connections created by
gsk_secure_soc_init() and does not affect existing connections.

**Related Topics**
- gsk_initialize()
- gsk_secure_soc_init()
gsk_user_set()
Chapter 10. Certificate/Key Management

This topic discusses the use of the z/OS shell-based gskkyman command to manage private keys, certificates and tokens. In addition, detailed examples using the gskkyman command are shown in this topic (see "gskkyman Command Line Mode Examples" on page 431).

Introduction

SSL connections make use of public/private key mechanisms for authenticating each side of the SSL session and agreeing on bulk encryption keys to be used for the SSL session. To use public/private key mechanisms (termed PKI), public/private key pairs must be generated. In addition, X.509 certificates (which contain public keys) may need to be created, or certificates must be requested, received, and managed.

System SSL supports these two methods for managing PKI private keys and certificates:

- A z/OS shell-based program called gskkyman. gskkyman creates, fills in and manages either a z/OS file or z/OS PKCS #11 token that contains PKI private keys, certificate requests, and certificates. The z/OS file is called a key database and, by convention, has a file extension of .kdb.
- The z/OS Security Server (RACF) RACDCERT command. RACDCERT installs and maintains PKI private keys and certificates in RACF. Refer to the z/OS Security Server RACF Command Language Reference for details on the RACDCERT command. RACF supports multiple PKI private keys and certificates to be managed as a group. These groups are called key rings or z/OS PKCS #11 tokens.
- RACF key rings or z/OS PKCS #11 tokens are the preferred method for managing PKI private keys and certificates for System SSL.

The System SSL application uses the GSK_KEYRING_FILE parameter of the gsk_attribute_set_buffer() API or the GSK_KEYRING_FILE environment variable to specify the locations of the PKI private keys and certificates to System SSL. If you are using a z/OS key database, the key database file name is passed in this parameter. If you are using a RACF key ring or z/OS PKCS #11 token, the name of the key ring or token is passed in this parameter.

**gskkyman Overview**

gskkyman is a z/OS shell-based program that creates, fills in, and manages a z/OS file or z/OS PKCS #11 token that contains PKI private keys, certificate requests, and certificates. The z/OS file is called a key database and, by convention, has a file extension of .kdb. There is also an .rdb file that is a counterpart to the .kdb file.

The interface to gskkyman, while command-line based, is an interactive dialog between you (the user) and the program. At each step, the interactive gskkyman program prompts you with one or more lines of output and expects a numeric choice to be supplied as input at the prompt. Once a choice has been made, the gskkyman program prompts you for the individual pieces of information needed to fulfill the request. You are prompted for each piece of information. Many times there is a default choice that is listed between parentheses at the end of the command prompt. If the default choice is acceptable, press Enter to select the default. If a choice other than the default is desired, enter the value at the prompt and press Enter. If a value is entered that is outside of the acceptable range of inputs, you will be re-prompted for the information.

**Note:** For a description of command line mode functions and options, see "gskkyman Command Line Mode Syntax" on page 427.
Setting Up the Environment to Run gskkyman

gskkyman uses the DLLs that are installed with System SSL and must have access to these at run-time. gskkyman must also have access to the message catalogs. The /bin directory includes a symbolic link to gskkyman, therefore, if your PATH environment variable contains this directory, you will find gskkyman. If your PATH environment variable does not contain this directory, add /usr/lpp/gskssl/bin to your PATH using:

```bash
PATH=$PATH:/usr/lpp/gskssl/bin
```

These DLLs include symbolic links to the message catalogs for gskkyman. If they do not include these links, add /usr/lpp/gskssl/lib/nls/msg to your NLSPATH using this command:

```bash
export NLSPATH=$NLSPATH:/usr/lpp/gskssl/lib/nls/msg
```

This setting assumes that your environment has the LANG environment variable set to En_US.IBM-1047 (or Ja_JP.IBM-939 for JCPT3BJ installations) include symbolic links to the message catalogs for gskkyman. If they do not include these links, add /usr/lpp/gskssl/lib/nls/msg to your NLSPATH using this command:

```bash
export NLSPATH=$NLSPATH:/usr/lpp/gskssl/lib/nls/msg
```

This setting assumes that your environment has the LANG environment variable set to En_US.IBM-1047 (or Ja_JP.IBM-939 for JCPT3BJ installations that expect Japanese messages and prompts). If LANG is not set properly, set the NLSPATH environment variable using this command:

```bash
export NLSPATH=$NLSPATH:/usr/lpp/gskssl/lib/nls/msg
```

The DLLs for System SSL are installed into a partitioned dataset (PDSE) in HLQ.SIEALNKE. These DLLs are not installed in SYS1.LPALIB by default. If System SSL is to execute in FIPS mode, the DLLs in the HLQ.SIEALNKE dataset cannot be put into the LPA.

If the System SSL DLLs have not been put into either the dynamic LPA or system link list, you must set the STEPLIB environment variable to find the DLLs. For example:

```bash
export STEPLIB=$STEPLIB:<HLQ>.SIEALNKE
```

Key Database Files

Key database files are password protected because they contain the private keys that are associated with some of the certificates that are contained in the key database. Private keys, as their name implies, should be protected because their value is used in verifying the authenticity of requests made during PKI operations.

It is recommended that key database files be set with these string file permissions:

- rw------- (600) (read-write for only the owner of the key database)

The owner of the key database should be the user who will be managing the key database. The program using System SSL (and the key database) must have at least read permission to the key database file at run-time. If the program is a server program that runs under a different user ID than the administrator of the key database file, it is recommended that a group be setup to control access to the key database file. In this case, it is recommended that you set the permissions on the key database file to:

- rw-r--- (640) (read-write for owner and read-only for group)

The owner of the key database file is set to the administrator user ID and the group owner of the key database file is set to the group that contains the server that will be using the key database file.

A key database that is created as a FIPS mode database, can only be updated by gskkyman or by using the CMS APIs executing in FIPS mode. Such a database, however, may be opened as read-only when executing in non-FIPS mode. Key databases created while in non-FIPS mode cannot be opened when executing in FIPS mode.
z/OS PKCS #11 Tokens

z/OS PKCS #11 tokens are managed and protected by ICSF. ICSF uses the CRYPTOZ SAF class to determine if the issuer of gskkyman is permitted to perform the operation against a z/OS PKCS #11 token. This class has two resources:

- USER.token-name
- SO.token-name

Table 7 on page 375 illustrates the SAF access levels required to perform certain functions. The 3 SAF levels in order of increasing accessibility are READ, UPDATE, and CONTROL. The higher levels each retain all the permissions of the previous level as well as gaining additional capability. For more information see the Token Access Levels table under Overview of z/OS support for PKCS #11 in z/OS Cryptographic Services ICSF Writing PKCS #11 Applications.

Table 7. SAF Access Levels

<table>
<thead>
<tr>
<th>USER.token-name CRYPTOZ resource:</th>
<th>SAF Access Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create/delete/modify CA certificate and private key</td>
<td>Control</td>
</tr>
<tr>
<td>Create/delete/modify user certificate and private key</td>
<td>Update</td>
</tr>
<tr>
<td>Read certificate and private key</td>
<td>Read</td>
</tr>
<tr>
<td>Set default key</td>
<td>Update</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SO.token-name CRYPTOZ resource:</th>
<th>SAF Access Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create or delete token</td>
<td>Update</td>
</tr>
<tr>
<td>Read/create/delete/modify certificate (but not the private key)</td>
<td>Read</td>
</tr>
<tr>
<td>Read/create/delete/modify private key</td>
<td>Control</td>
</tr>
<tr>
<td>Set default key</td>
<td>Read</td>
</tr>
</tbody>
</table>

gskkyman Interactive Mode Descriptions

Interactive mode is entered when the gskkyman command is entered without any parameters. A series of menus will be presented to allow you to select the database functions to be performed. Leading and trailing blanks will be removed from data entries but imbedded blanks will be retained. Blanks will not be removed from passwords.

Database Menu

This is the top-level menu and is displayed when the gskkyman command starts:
Create new database

This option will create a new key database and the associated request database. You will be prompted to enter the key database name, the database password, the password expiration interval, and the database record length and choose either a FIPS or non-FIPS database (see “Key Database Files” on page 374 for a discussion of FIPS mode databases).

The fully-qualified key database name must be between 2 and 251 characters and should either have no extension or an extension of '.kdb' (the maximum database name is 247 characters if the name does not end with an extension of 1-3 characters to allow for the addition of an extension when creating the request database or the password stash file). The key database name may not end with '.rdb' or '.sth' as these extensions are reserved for the request database and the password stash file.

The database password must be between 1 and 128 characters. A password exceeding 128 characters will be truncated to 128 characters.

The password expiration interval must be between 0 and 9999 days (a value of 0 indicates the password does not expire).

The record length must be large enough to contain the largest certificate to be stored in the database and must be between 2500 and 65536.

Two files will be created: the key database and the request database. The request database has an extension of '.rdb'. The file access permissions will be set so only the owner has access to the files.

Open database

This option will open an existing database. You will be prompted to enter the key database name and the database password.

The fully-qualified key database name must be between 2 and 251 characters and should either have no extension or an extension of '.kdb' (the maximum database name is 247 characters if the name does not end with an extension of 1-3 characters to allow for the addition of an extension.
when accessing the request database or the password stash file). The key database name may not end with ".rdb" or ".sth" as these extensions are reserved for the request database and the password stash file.

Change database password

This option will change the database password. You can change the password at any time but you must change it once it has expired in order to access the database once more. You will be prompted to enter the key database name, the current database password, the new database password, and the new password expiration interval.

The fully-qualified key database name must be between 2 and 251 characters and should either have no extension or an extension of ".kdb" (the maximum database name is 247 characters if the name does not end with an extension of 1-3 characters to allow for the addition of an extension when accessing the request database or the password stash file). The key database name may not end with ".rdb" or ".sth" as these extensions are reserved for the request database and the password stash file.

The new database password must be between 1 and 128 characters.

The password expiration interval must be between 0 and 9999 days (a value of 0 indicates the password does not expire).

Change database record length

This option will change the database record length. All database records have the same length and database entries cannot span records. You can increase the record length if you find it is too small to store a new certificate. You can decrease the record length to reduce the database size if the original record length is too large. You cannot reduce the record length to a value smaller than the largest certificate currently in the database. You will be prompted to enter the key database name, the database password, and the new record length.

The fully-qualified key database name must be between 2 and 251 characters and should either have no extension or an extension of ".kdb" (the maximum database name is 247 characters if the name does not end with an extension of 1-3 characters to allow for the addition of an extension when accessing the request database or the password stash file). The key database name may not end with ".rdb" or ".sth" as these extensions are reserved for the request database and the password stash file.

The new record length must be between 2500 and 65536.

Delete database

This option will delete the key database, the associated request database, and the database password stash file. You will be prompted to enter the key database name.

The fully-qualified key database name must be between 2 and 251 characters and should either have no extension or an extension of ".kdb" (the maximum database name is 247 characters if the name does not end with an extension of 1-3 characters to allow for the addition of an extension when accessing the request database or the password stash file). The key database name may not end with ".rdb" or ".sth" as these extensions are reserved for the request database and the password stash file.

Create key parameter file

This option will create a file containing a set of key generation parameters. Key generation parameters are used when generating Digital Signature Standard (DSS) and Diffie-Hellman (DH) keys. The parameters will be stored in the specified file as an ASN.1-encoded sequence in Base64 format. This file can then be used when creating a signed certificate. The same key generation parameters can be used to generate multiple public/private key pairs. Using the same key generation parameters significantly reduces the time required to generate a public/private key pair. In addition, the Diffie-Hellman key agreement method requires both sides to use the same group parameters in order to compute the key exchange value. Refer to FIPS 186-2 (Digital
Signature Standard) and RFC 2631 (Diffie-Hellman Key Agreement Method) for more information on the key generation parameters. The key parameter generation process can take from 1 to 10 minutes depending upon key size, processor speed and system load.

**Display certificate file (Binary or Base64 ASN.1 DER)**
This option displays information about an X.509 certificate file. You will be prompted to enter the certificate filename. The fully-qualified certificate filename must be between 2 and 251 characters. The specified file must contain either a binary ASN.1 DER-encoded certificate or the Base64-encoding of a binary ASN.1 stream. A Base64-encoded certificate must be in the local code page.

**Note:** Information retrieved for z/OS PKCS #11 tokens is not cached. Each time a menu is displayed, the information is retrieved from the ICSF TKDS (token key dataspace). This is also true when displaying the list of available z/OS PKCS #11 tokens. On return from displaying a subordinate menu, the current list of tokens is retrieved and the menu refreshed.

**Create new token**
This option will create a new token. You will be prompted to enter the token name. The name must be a unique non-empty string and consist of characters that are alphanumeric, national (@ -x5B, # -x7B, $ -x7C) and period (x4B). The name is specified in the local code page. The first character must be alphabetic or national. Lowercase letters are permitted but will be folded to uppercase.

Once the token is created the Database Menu is displayed.

**Delete token**
This option will delete the key token. You will be prompted to enter the token name. If the token exists, the user is prompted again to re-enter the full token name as confirmation prior to deletion of the specified token.

**Note:** If name consists of lowercase characters it will be uppercased when processed.

**Manage token**
This option manages the token. You will be prompted to enter the token name. The token that matches the entered name is then used in the Token Management Menu that is subsequently displayed.

**Note:** If name consists of lowercase characters it will be uppercased when processed.

**Manage token from list of tokens**
This option displays a list of existing tokens by name from which an entry can be chosen for use in the Token Management Menu that is subsequently displayed.

**Note:** If name consists of lowercase characters it will be uppercased when processed.

**Key/Token Management**
The Key/Token Management menus allow for the creation/deletion/management of certificates within a key database file or z/OS PKCS #11 token. Once the key database or token is created, the management of the certificates within the repository is very similar. This is illustrated throughout this topic by the key database menu, which is always on the left, and token menu, which is always on the right, being displayed side by side in the figures.
Key Management Menu/Token Management Menu

The **Key Management Menu** is displayed once the key database has been created or opened. The key database and the associated request database are opened for update and remain open until you return to the **Database Menu**.

The Token Management Menu is displayed once a z/OS PKCS #11 token has been opened.

![Key Management Menu](image1)

![Token Management Menu](image2)

#### Manage Keys and Certificates

This option manages certificates with private keys. A list of key labels is displayed. Pressing the ENTER key without making a selection will display the next set of labels. Selecting one of the label numbers will display this menu:

---

*Figure 3. Key Management Menu/Token Management Menu*

---

**Manage Keys and Certificates**

This option manages certificates with private keys. A list of key labels is displayed. Pressing the ENTER key without making a selection will display the next set of labels. Selecting one of the label numbers will display this menu:
Show certificate information
This option displays information about the X.509 certificate associated with the private key.

Show key information
This option displays information about the private key.

Set key as default
This option makes the current key the default key for the database.

Set certificate trust status
This option sets or resets the trusted status for the X.509 certificate. A certificate cannot be used for authentication unless it is trusted.

Note: All z/OS PKCS #11 token certificates are automatically created with the status set to trusted. Changing of the trust status is not supported for z/OS PKCS #11 token certificates.

Copy certificate and key to another database/token
This option copies the certificate and key to another token or a database. An error is returned if the certificate is already in the token/database or if the label is not unique. A certificate and key may only be copied from a FIPS mode database to another FIPS mode database. A certificate and key may not be copied from a non-FIPS mode database or a PKCS #11 token to a FIPS mode database.

Export certificate to a file
This option exports just the X.509 certificate to a file. The supported export formats are ASN.1 Distinguished Encoding Rules (DER) and PKCS #7 (Cryptographic Message Syntax)

Export certificate and key to a file
This option exports the X.509 certificate and its private key to a file. The private key is encrypted when it is written to the file. The password you select will be needed when you import the file. The supported export formats for a key database file are PKCS #12 Version 1 (obsoleted) and PKCS #12 Version 3. For z/OS PKCS #11 tokens and FIPS mode databases, the export format supported is PKCS #12 Version 3. The strong encryption option uses Triple DES to encrypt the private key while the export encryption option uses 40-bit RC2. Strong encryption is the only supported option when exporting from a FIPS database. The export file will contain the requested certificate and its certification chain.
**Delete certificate and key**
The certificate and its associated private key are deleted.

**Change label**
This option will change the label for the database record.

**Create a signed certificate and key**
This option will create a new certificate and associated public/private key pair. The new certificate will be signed using the certificate in the current record and then stored in either the key database file or z/OS PKCS #11 token.

DSS and DH based certificates are only supported in key database files. The key generation parameters must be compatible with the requested key type and key size.

Keys are in the same domain if they have the same set of key generation parameters. Refer to FIPS 186-2 (Digital Signature Standard) and RFC 2631 (Diffie-Hellman Key Agreement Method) for more information on the key generation parameters. The subject name and one or more subject alternate names can be specified for the new certificate.

The subject name is always an X.500 directory name while a subject alternate name can be an X.500 directory name, a domain name, an e-mail address, an IP address, or a uniform resource identifier. An X.500 directory name consists of common name, organization, and country attributes with optional organizational unit, city/locality, and state/province attributes. A domain name is one or more tokens separated by periods. An e-mail address consists of a user name and a domain name separated by '@'. An IP address is an IPv4 address (nnn.nnn.nnn.nnn) or an IPv6 address (nnnn:nnnn:nnnn:nnnn:nnnn:nnnn:nnnn:nnnn). A uniform resource identifier consists of a scheme name, a domain name, and a scheme-specific portion.

The signature algorithm used when signing the certificate is derived from the key algorithm of the signing certificate. The digest type used when signing the certificate will match the digest type used in the signature algorithm of the signing certificate. If the signing certificate’s key algorithm is DSA, or the digest type used by the signing certificate is not a SHA-based digest, then the digest type used will be SHA-1. Possible signature algorithms are:

- x509_alg_sha1WithRsaEncryption
- x509_alg_sha224WithRsaEncryption
- x509_alg_sha256WithRsaEncryption
- x509_alg_sha384WithRsaEncryption
- x509_alg_sha512WithRsaEncryption
- x509_alg_dsaWithSha1

**Create a certificate renewal request**
This option will create a certification request using the subject name and public/private key pair from an existing certificate. The certificate request will be exported to a file in Base64 format. This file can then be sent to a certification authority for processing. The certificate returned by the certification authority can then be processed using option 5 (Receive requested certificate or a renewal certificate) on the **Key Management Menu** or **Token Management Menu**. The new certificate will replace the existing certificate.

**Manage Certificates**
This option manages certificates without private keys. A list of key labels is displayed. Pressing the ENTER key without making a selection will display the next set of labels. Selecting one of the label numbers will display this menu:
### Certificate Menu

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Show certificate information</td>
</tr>
<tr>
<td>2</td>
<td>Set certificate trust status</td>
</tr>
<tr>
<td>3</td>
<td>Copy certificate to another database</td>
</tr>
<tr>
<td>4</td>
<td>Export certificate to a file</td>
</tr>
<tr>
<td>5</td>
<td>Delete certificate</td>
</tr>
<tr>
<td>6</td>
<td>Change label</td>
</tr>
<tr>
<td>0</td>
<td>Exit program</td>
</tr>
</tbody>
</table>

Enter option number (press ENTER to return to previous menu):

### Token Certificate Menu

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Show certificate information</td>
</tr>
<tr>
<td>2</td>
<td>Set certificate trust status</td>
</tr>
<tr>
<td>3</td>
<td>Copy certificate to another database/token</td>
</tr>
<tr>
<td>4</td>
<td>Export certificate to a file</td>
</tr>
<tr>
<td>5</td>
<td>Delete certificate</td>
</tr>
<tr>
<td>6</td>
<td>Change label</td>
</tr>
<tr>
<td>0</td>
<td>Exit program</td>
</tr>
</tbody>
</table>

Enter option number (press ENTER to return to the previous menu):

---

**Figure 5. Certificate Menus**

**Show certificate information**
This option displays information about the X.509 certificate.

**Set certificate trust status**
This option sets or resets the trusted status for the X.509 certificate. A certificate cannot be used for authentication unless it is trusted.

**Note:** All z/OS PKCS #11 token certificates are automatically created with the status set to trusted. Changing of the trust status is not supported for z/OS PKCS #11 token certificates.

**Copy certificate to another database/token**
This option copies the certificate to another token or a key database. An error is returned if the certificate is already in the token/database or if the label is not unique. A certificate and key may only be copied from a FIPS mode database to another FIPS mode database. A certificate and key may not be copied from a non-FIPS mode database or a PKCS #11 token to a FIPS mode database.

**Export certificate to a file**
This option exports the X.509 certificate to a file. The supported export formats are ASN.1 DER (Distinguished Encoding Rules) and PKCS #7 (Cryptographic Message Syntax). The export file will contain just the requested certificate when the DER format is selected. The export file will contain the requested certificate and its certification chain when the PKCS #7 format is selected.

**Delete certificate**
The certificate is deleted.

**Change label**
This option will change the label for the certificate.

**Manage Certificate Requests**
This option manages certificate requests. A list of request labels is displayed. Pressing the ENTER key without making a selection will display the next set of labels. Selecting one of the label numbers will display this menu:
Show key information
This option displays information about the private key associated with the certificate request.

Export certificate request to a file
This option exports the certificate request to a file in Base64 format. This file can then be sent to a certification authority for processing.

Delete certificate request and key
The certificate request and its associated private key are deleted.

Change label
This option will change the label for the certificate request.

Create New Certificate Request
This option will create a certificate request using either RSA encryption or DSA for the public and private keys (DSA only supported for key databases). The certificate request will be exported to a file in Base64 format. This file can then be sent to a certification authority for processing.

For key databases:
The label has a maximum length of 127 characters and is used to reference the certificate in the request database. The label will also be used when the certificate is received, so it must be unique in both the request and key databases. It must consist of characters which can be represented as 7-bit ASCII characters (letters, numbers, and punctuation) in the ISO8859-1 code page.

For tokens:
The label has a maximum length of 32 characters and is used to reference the certificate request. The label will also be used when the certificate is received, so it must be unique in the token. It must consist of characters which can be represented in the IBM1047 code page.

The subject name and one or more subject alternate names can be specified for the new certificate. The subject name is always an X.500 directory name while a subject alternate name can be an X.500 directory name, a domain name, an e-mail address, an IP address, or a uniform resource identifier. An X.500 directory name consists of common name, organization, and country attributes with optional organizational unit, city/locality, and state/province attributes. A domain name is one or more tokens separated by periods. An e-mail address consists of a user name and a domain name separated by '@'. An IP address is an IPv4 address (nnnn.nnn.nnn.nnn) or an IPv6 address (nnnn:nnnn:nnnn:nnnn:nnnn:nnnn:nnnn:nnnn). A uniform resource identifier consists of a scheme name, a domain name, and a scheme-specific portion (for example, http://www.endicott.ibm.com/main.html).
**Receive Requested Certificate or a Renewal Certificate**

This option will receive the signed certificate returned by the certification authority. The certificate can be either a new or renewal certificate issued in response to a certificate request or a renewal of an existing certificate without a corresponding certificate request. If the certificate was issued in response to a certificate request, the certificate request must still be in the request database or token. If this is a renewal certificate without a certificate request, the old certificate must still be in the key database or token and must have the same issuer name and public key. If the key database or token does not contain the private key of the old certificate or contains certificates signed by the old certificate, then the subject name must also be the same when renewing the certificate.

The certificate file must contain either an ASN.1 DER-encoded sequence as defined in RFC 2459 (X.509 Public Key Infrastructure), RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile) or a signed data message as defined in PKCS #7 (Cryptographic Message Syntax). The data can either be the binary value or the Base64 encoding of the binary value.

If the import file is in PKCS #7 format, the first certificate in the file must be the request certificate, otherwise the request will fail with 'unable to locate matching request'. The certification chain will be imported if it is contained in the import file. The certificate subject name will be used as the label for certificates added from the certification chain. A chain certificate will not be added if the label is not unique or if the certificate is already in the database or token.

Base64 data is in the local code page. A DER-encoded sequence must start with the encoding header ‘-----BEGIN CERTIFICATE-----’ and end with the encoding footer ‘-----END CERTIFICATE-----’. A PKCS #7 signed data message must start with the encoding header ‘-----BEGIN CERTIFICATE-----’ and end with the encoding footer ‘-----END CERTIFICATE-----’ or start with the encoding header ‘-----BEGIN PKCS #7 SIGNED DATA-----’ and end with the encoding footer ‘-----END PKCS #7 SIGNED DATA-----’.

An intermediate CA or end-entity certificate is a certificate signed by another entity. The key database or token must already contain a certificate for the issuer. The certificate will not be imported if the certificate authenticity cannot be validated or if the database or token already contains the certificate.

The certificate request entry will be deleted once the certificate has been received.

**Create a Self-Signed Certificate**

This option will create a self-signed certificate using either RSA or DSA encryption for the public and private keys and a certificate signature based on a SHA digest algorithm of the user's choice (if an RSA certificate is requested). Possible signature algorithms are:

- x509_alg_sha1WithRsaEncryption
- x509_alg_sha224WithRsaEncryption
- x509_alg_sha256WithRsaEncryption
- x509_alg_sha384WithRsaEncryption
- x509_alg_sha512WithRsaEncryption
- x509_alg_dsaWithSha1

The certificate can be created for use by a certification authority or an end user. A CA certificate can be used to sign other certificates and certificate revocation lists while an end user certificate can be used for authentication, digital signatures, and data encryption.

**Note:** DSA certificates are only supported in key database files.

**For key databases:**

The label has a maximum length of 127 characters and is used to reference the certificate in the request database. The label will also be used when the certificate is received, so it must be unique in both the request and key databases. It must consist of characters which can be represented as 7-bit ASCII characters (letters, numbers, and punctuation) in the ISO8859-1 code page.
For tokens:
The label has a maximum length of 32 characters and is used to reference the certificate request. The label will also be used when the certificate is received, so it must be unique in the token. It must consist of characters which can be represented in the IBM1047 code page.

The number of days until the certificate expires must be between 1 and 9999.

The subject name and one or more subject alternate names can be specified for the new certificate. The subject name is always an X.500 directory name while a subject alternate name can be an X.500 directory name, a domain name, an e-mail address, an IP address, or a uniform resource identifier. An X.500 directory name consists of common name, organization, and country attributes with optional organizational unit, city/locality, and state/province attributes. A domain name is one or more tokens separated by periods. An e-mail address consists of a user name and a domain name separated by '@'. An IP address is an IPv4 address (nnn.nnn.nnn.nnn) or an IPv6 address (nnnn:nnnn:nnnn:nnnn:nnnn:nnnn:nnnn:nnnn). A uniform resource identifier consists of a scheme name, a domain name, and a scheme-specific portion (for example, http://www.endicott.ibm.com/main.html).

Note: A self-signed end-entity certificate (server or client certificate) is not recommended for use in production environments and should only be used to facilitate test environments prior to production. Self-signed certificates do not imply any level of security or authenticity of the certificate because, as their name implies, they are signed by the same key that is contained in the certificate. On the other hand, certificates that are signed by a certificate authority indicate that, at least at the time of signature, the certificate authority approved the information contained in the certificate.

Import a Certificate

This option will add the contents of the import file to a key database file or z/OS PKCS #11 token. The import file may contain one or more certificates without private keys. When each certificate is added to the key database, it is marked as trusted. The expiration date associated with each certificate cannot exceed February 6, 2016.

When adding certificates from the import file to a FIPS key database file only certificates signed with FIPS signature algorithms using FIPS-approved key sizes may be imported. When processing a chain of certificates, processing of the chain will terminate if a non-FIPS certificate is encountered. Certificates processed prior to the failing certificate will be added to the key database file. It is the responsibility of the importer to ensure that the file came from a FIPS source in order to maintain meeting FIPS 140-2 criteria.

The import file must contain either an ASN.1 DER-encoded sequence as defined in RFC 2459 (X.509 Public Key Infrastructure), RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile) or a signed data message as defined in PKCS #7 (Cryptographic Message Syntax). The data can either be the binary value or the Base64 encoding of the binary value.

If the import file is in PKCS #7 format, only the first certificate and its certification chain will be imported. The certificate subject name will be used as the label for certificates added from the certification chain. A certification chain certificate will not be added to the database or z/OS PKCS #11 token if the label is not unique or if the certificate is already in the database or z/OS PKCS #11 token.

Base64 data is in the local code page. A DER-encoded sequence must start with the encoding header '-----BEGIN CERTIFICATE-----' and end with the encoding footer '-----END CERTIFICATE-----'. A PKCS #7 signed data message must start with the encoding header '-----BEGIN CERTIFICATE-----' and end with the encoding footer '-----END CERTIFICATE-----' or start with the encoding header '-----BEGIN PKCS #7 SIGNED DATA-----' and end with the encoding footer '-----END PKCS #7 SIGNED DATA-----'.

A root certificate is a self-signed certificate and will be imported as long as the certificate is not already in the key database or z/OS PKCS #11 token.
An intermediate CA or end-entity certificate is a certificate signed by another entity. The key database or z/OS PKCS #11 token must already contain a certificate for the issuer. The certificate will not be imported if the certificate authenticity cannot be validated or if the database already contains the certificate.

An existing certificate can be replaced by specifying the label of the existing certificate. The issuer name, subject name, and subject public key in the new certificate must be the same as the existing certificate. If the existing certificate has a private key, the private key is not changed when the certificate is replaced.

Import a Certificate and a Private Key
This option imports a certificate and the associated private key and adds it to the key database or z/OS PKCS #11 token. The certificate will be marked as trusted when it is added. When importing a certificate, the expiration date cannot exceed February 6, 2106.

The import file must contain an ASN.1 DER-encoded sequence as defined in PKCS #12 (Personal Information Exchange Syntax). The data can be either the binary value or the Base64 encoding of the binary value. Base64 data is in the local code page and must start with the encoding header ‘-----BEGIN CERTIFICATE-----’ and end with the encoding footer ‘-----END CERTIFICATE-----’.

A root certificate is a self-signed certificate and will be imported as long as the certificate is not already in the key database or z/OS PKCS #11 token.

Each certificate in the certification chain will be imported if it is present in the import file. The certificate subject name will be used as the label for certificates added from the certification chain. A certification chain certificate will not be added to the database or z/OS PKCS #11 token if the label is not unique or if the certificate is already in the database or z/OS PKCS #11 token.

Only certificates and keys encoded according to PKCS #12 Version 3 and protected with strong encryption can be imported into a FIPS database. Furthermore, only certificates and keys comprising FIPS signature algorithms and using FIPS-approved key sizes may be imported into a FIPS database.

Show the Default Key
The private key information for the default key is displayed.

Store Database Password
The database password is masked and written to the key stash file. The file name is the same as the key database file name but has an extension of ‘.sth’.

Show Database Record Length
The database record length is displayed. All records in the database have the same length and a database entry cannot span a database record.

**gskkyman Interactive Mode Examples**

*gskkyman* can be run from either an rlogin z/OS shell environment or from the OMVS shell command-line environment. The examples that follow were performed from the rlogin environment. If you use the OMVS shell command-line environment, the only difference is that all input will be done at the command prompt at the bottom of the screen.

These tasks will be performed in this topic:

- Creating, opening and deleting a key database file
- Changing a key database password

386 z/OS V1R11.0 System SSL Programming
• Storing an encrypted key database password
• Creating, opening and deleting a z/OS PKCS #11 token
• Creating a self-signed server or client certificate
• Creating a certificate request and processing the signed request
• Creating a certificate to be used with Diffie-Hellman key exchange (key database only)
• Managing keys and certificates:
  – Show certificate/key information
  – Marking a certificate (and private key) as the default certificate for the key database
  – Copying a certificate (and private key) to a different key database or z/OS PKCS #11 Token:
    - Copying a certificate without its private key
    - Copying a certificate with its private key
    - Copying a certificate with its private key to a key database on the same system
    - Copying a certificate with its private key to another z/OS PKCS #11 token or key database on the same system
  – Removing a certificate (and private key) from a key database or z/OS PKCS #11 token
  – Changing a certificate label
• Importing a certificate from a file as a trusted CA certificate
• Importing a certificate from a file with its private key
• Using gskkyman to be your own certificate authority (CA) (key database only)
• Migrating key database files to RACF key rings (key database only)
• Migrating key database files to z/OS PKCS #11 Tokens

**Starting gskkyman**

To start gskkyman, enter **gskkyman** at the command prompt (see Figure 7 on page 388).

**Note:** In the examples that follow, your input is shown in **bold**, and places where you press the Enter key are noted with `<enter>`. Figure 7 on page 388 shows the gskkyman start menu.
Creating, Opening and Deleting a Key Database File

To create a new key database, enter 1 at the command prompt on the Database Menu:
Figure 8. Creating a New Key Database

Figure 8 shows the input prompts that gskkyman produces when you choose 1 to create a new key database. As you can see, default choices are listed in parentheses. In the example, by pressing the Enter key at the Enter database record length prompt, the default of 5000 was chosen.

Note: When dealing with certificates which may be large in size or have large key sizes, for example 2048 or 4096, an initial key record length of 5000 may be required.

Note: The maximum length of the password specified for a key database file is 128 characters.

After entering the database record length, a message displays confirming that your database was created (see Figure 8). You are prompted to press Enter to continue. Doing so displays the Key Management Menu for the database you have created:
Figure 9 shows the Key Management Menu. Entering 0 at this prompt exits the gskkyman program. Pressing Enter at the prompt returns you to the Database Menu.

To open an existing key database file, on the Database Menu, enter option number 2 (see Figure 10 on page 391). You are then prompted for the key database name and password.

Note: Do not lose the key database password. There is no method to reset this password if you lose or forget the password. If the password is lost, the private keys stored in the key database are inaccessible, therefore, unusable.
The key database name is the file name of the key database. The input file name is interpreted relative to the current directory when `gskkyman` is invoked. You may also specify a fully qualified key database name.

After you enter the key database name and password, the Key Management Menu displays for the database you have selected to open, (see Figure 11).

![Figure 10. Opening an Existing Key Database File](image)

**Figure 10. Opening an Existing Key Database File**

The key database name is the file name of the key database. The input file name is interpreted relative to the current directory when `gskkyman` is invoked. You may also specify a fully qualified key database name.

After you enter the key database name and password, the Key Management Menu displays for the database you have selected to open, (see Figure 11).

![Figure 11. Key Management Menu](image)

**Figure 11. Key Management Menu**
To delete an existing database, from the **Database Menu**, select option **5** (see Figure 12):

```
Database Menu
1 - Create new database
2 - Open database
3 - Change database password
4 - Change database record length
5 - Delete database
6 - Create key parameter file
7 - Display certificate file (Binary or Base64 ASN.1 DER)
11 - Create new token
12 - Delete token
13 - Manage token
14 - Manage token from list of tokens
0 - Exit program
```

Enter option number: **5** <enter>
Enter key database name (press ENTER to return to menu): **mykey.kdb** <enter>

Enter 1 to confirm delete, 0 to cancel delete: **1** <enter>

Key database /home/sufwl1/ssl_cmd/mykey.kdb deleted.

Press ENTER to continue.

Figure 12. Deleting an Existing Key Database

You are prompted to enter the key database name that you wish to delete. Then you must enter 1 to confirm the delete, or 0 to cancel the delete. If you choose 1, a message displays to confirm the file has been deleted.

**Note:** If you delete an existing key database, the associated request database and database password stash file (if existent) **will also be deleted**. It's important to note that anyone with write access to a key database can delete that database either by removing it with the **rm** command or by using **gskkyman** subcommand.

### Changing a Key Database Password

You can change a key database password. From the **Database Menu**, select option **3**:
Figure 13 displays the prompts you are given. You first enter your current password. Then you select a new password, and enter it again to confirm. You can choose your password expiration in days or press Enter to have no expiration. A message displays to confirm the transaction.

Storing an Encrypted Key Database Password

In order for applications to use the key database file, the application must specify both the file name as well as its associated password. The password can either be specified directly or through a stash file containing the encrypted password. The stash file provides a level of security where the password does not have to be explicitly specified. To save the encrypted key database password, enter option 10 from the Key Management Menu:

Note: In these task descriptions, it is assumed that you have opened the key database and are displaying the Key Management Menu panel.
Figure 14 shows the message you receive after entering option 10 to store the database password. In this example, the database password was stored in a file called mykey.sth.

Creating, opening and deleting a z/OS PKCS #11 token

To create a new z/OS PKCS #11 token, enter 11 at the command prompt on the Database Menu:
The only input required when creating a new z/OS PKCS #11 token is the token name.

Note: Only users with SAF access level of UPDATE or CONTROL to the CRYPTOZ resource "so.token-name" have the authority to create the z/OS PKCS #11 token with the name "token-name".

Note: A z/OS PKCS #11 token contains no certificates or keys when first created.

After entering the token name, a message displays confirming that the z/OS PKCS #11 token was created (see Figure 15). You are prompted to press Enter to continue. Doing so re-displays the Database Menu.

To open an existing z/OS PKCS #11 token, enter either option 13 or option 14 on the Database Menu. If option 13 is used:
If option 14 is used:

Database Menu

1 - Create new database
2 - Open database
3 - Change database password
4 - Change database record length
5 - Delete database
6 - Create key parameter file
7 - Display certificate file (Binary or Base64 ASN.1 DER)

11 - Create new token
12 - Delete token
13 - Manage token
14 - Manage token from list of tokens
0 - Exit program

Enter option number: 14 <enter>

Token List

1 - TOKEN1
0 - Return to selection menu

Enter list-entry number (press ENTER to return to previous menu): 1 <enter>

Figure 16. Opening a z/OS PKCS #11 Token via token name

Figure 17. Opening a z/OS PKCS #11 Token via token list

After either entering the token name (if option 13 used) or selecting the token from a list of tokens (if option 14 is used), the Token Management Menu displays the z/OS PKCS #11 token selected (see
Note: Only users with SAF access level of READ, UPDATE or CONTROL to the CRYPTOZ resource "so.token-name" or "user.token.name" have the authority to open the z/OS PKCS #11 token with the name "token-name".

To delete an existing z/OS PKCS #11 token, enter either option 12 on the Database Menu, or select option 10 from the Token Management Menu.

If option 12 on the Database Menu is used:
If option 10 on the Token Management Menu is used:

```plaintext
Database Menu
1 - Create new database
2 - Open database
3 - Change database password
4 - Change database record length
5 - Delete database
6 - Create key parameter file
7 - Display certificate file (Binary or Base64 ASN.1 DER)
11 - Create new token
12 - Delete token
13 - Manage token
14 - Manage token from list of tokens
0 - Exit program

Enter option number: 12 <enter>

Enter token name (press ENTER to return to menu): TOKEN1 <enter>

To confirm token delete, enter token name again (press ENTER to cancel delete): TOKEN1 <enter>

Token successfully deleted

Press ENTER to continue.
```
Using either approach you are prompted to enter the token name in order to confirm the correct token is deleted. A message is displayed to confirm that the z/OS PKCS #11 token has been deleted. The token does not have to be empty before performing the delete.

**Note:** Only users with SAF access level of UPDATE or CONTROL to the CRYPTOZ resource "so.token-name" have the authority to delete the z/OS PKCS #11 token with the name "token-name".

### Creating a Self-Signed Server or Client Certificate

If your organization does not use a certificate authority (within the organization or outside the organization), a self-signed certificate can be generated for use by the program acting as an SSL server or client. In addition, since root CA certificates are also self signed certificates that are permitted to be used to sign other certificates (certificate requests), these procedures can also be used to create a root CA certificate. See [Marking a Certificate (and Private Key) as the Default Certificate](#) on page 412.

Programs acting as SSL servers (i.e. acting as the server side of the SSL handshake protocol) must have a certificate to use during the handshake protocol. A program acting as an SSL client requires a certificate when the SSL server requests client authentication as part of the SSL handshake.

**Note:** This is not recommended for production environments and should only be used to facilitate test environments prior to production. Self-signed certificates do not imply any level of security or authenticity of the certificate because, as their name implies, they are signed by the same key that is contained in the certificate. On the other hand, certificates that are signed by a certificate authority indicate that, at least at the time of signature, the certificate authority approved the information contained in the certificate.
Note: gskkyman supports the creation of X.509 Version 3 certificates.

When creating a self-signed certificate to be used to identify a server or client, from the Key Management Menu or Token Management Menu, enter 6. You will be prompted for a number of items to define the certificate. First you will be asked to select the type of certificate to be created.

Certificates that are intended to be used directly by a server or client are considered to be end-user certificates. Certificates intended to be used to sign other certificates are considered to be CA certificates. RSA key certificates are the most common. DSA key certificates represent certificates that follow the FIPS-186 government standard. The larger the key size, the more secure the generated key will be. The most commonly used size is 1024. Note that CPU usage increases as the key size increases. For example, CPU usage will increase by a factor of 6 when the key size is doubled.

If an RSA-based certificate is selected, you will be prompted to select the digest type for the signature algorithm from a list of SHA-based digest types. The default, if you press enter, is SHA-1. Figure 22 on page 401 is an example using the default. Previous releases of gskkyman created certificates using the SHA-T digest type.

Once the certificate type and signature algorithm is determined, you will be prompted to enter:

- a label to uniquely identify the key and certificate within the key database
- the individual fields within the subject name
- certificate expiration. The valid expiration range is 1 to 9999 days. The default value is 365 days.
- the subject alternate names (optional)

Figure 21. Creating a Self-Signed Certificate

Figure 22 on page 401 shows the creation of a self-signed certificate to be used as a server or client certificate in a key database file or z/OS PKCS #11 token. If creating a server or client certificate in a z/OS PKCS #11 token, the certificate type list will consist of supported options 1, 2, 5 and 6.
Once the certificate is created, the next step is to determine whether the certificate should be marked as the database’s or z/OS PKCS #11 token’s default certificate. Setting the certificate as the default certificate allows the certificate to be used by the SSL APIs without having to specify its label. For more information on setting the default certificate, see “Marking a Certificate (and Private Key) as the Default Certificate” on page 412.

In order for the SSL handshake to successfully validate the use of the self-signed certificates, the partner application needs to know about the signer of the certificate. For self-signed certificates, this means the self-signed certificate must be imported into the partner’s database or z/OS PKCS #11 token. For more information on importing certificates, see “Importing a Certificate from a File as a Trusted CA Certificate” on page 421.

Creating a Certificate Request

A program may require a certificate, associated with itself, depending on what side of the SSL connection the program is running. This requirement also depends on whether client authentication is requested as part of the SSL handshake. Programs acting as SSL servers (act as the server side of the SSL handshake protocol) must have a certificate to use during the handshake protocol. A program acting as an SSL client requires a certificate in the key database if the SSL server requests client authentication as part of the SSL handshake operation. The way in which certificates are used within an organization will determine whether you need to create a certificate request. If the organization chooses to use a certificate authority (within the organization or outside of the organization), then you must generate a certificate request.
To create a certificate request, enter 4 from the **Key Management Menu** or **Token Management Menu**.

---

**Figure 23. Key Management Menu/Token Management Menu**

When creating a certificate request, you are first prompted for the type of certificate to be requested. RSA key certificates are the most common. DSA key certificates represent certificates that follow the FIPS-186 government standard. The larger the key size, the more secure the encryption/decryption generated key will be. The most commonly used size is 1024.

After the certificate type is determined, you will be prompted to enter:

- a request file name to store the certificate request
- a label to uniquely identify the certificate request within the key database
- the individual fields within the subject name
- the individual fields within the subject alternate name (optional).

The Certificate Type menu appears:
Figure 24 shows the creation of a certificate request associated with a key database file. When creating a z/OS PKCS #11 Token certificate request, the certificate list will consist of supported options 1 and 2.

Enter option 0 to continue or option 1 to specify the subject alternate names. If option 1 is selected, the **Subject Alternate Name Type** menu appears:
When specifying subject alternate names, you are prompted for the type of the alternate name. After the alternate name type is determined, you will be prompted to enter:

- the individual fields within the subject name.

After the individual fields are completed, press enter to continue or select one of the subject alternate name types. Repeat the process.

Once the certificate request (and associated subject alternate names) is created, a file with the name you specified will exist in the current working directory or directory specified in the filename. If you choose to exit gskkyman, the program ends. Otherwise, the Key Management Menu or the Token Management Menu (see Figure 11 on page 391) displays, allowing additional operations to be performed.

The certificate request created is stored in a file that is in base64-encoded format. This format is what is typically required by certificate authorities that create certificates. This is the contents of the file created by the steps performed in Figure 24 on page 403.
Creating a certificate to be used with a fixed Diffie-Hellman key exchange (Key Database File Only)

Create a server certificate to be used during an SSL handshake using a fixed Diffie-Hellman key exchange. Fixed Diffie-Hellman requires both sides of the exchange to be based off of the same generation parameters. In order for each side to use the same generation parameters, a key parameter file must be created to be used as input to the certificate being signed.

To create a key parameter file, from the Database Menu, enter 6. First you will be asked to select the key type. Only the key types of 1024-bit DSA key or 2048-bit fixed Diffie-Hellman key are valid for use in a FIPS database. Once the key type is determined, you will be prompted to enter a key parameter file name. The file name is interpreted relative to the current directory when gskkyman is invoked. You may also specify a fully qualified file name.

Figure 26. Contents of certreq.arm after Certificate Request Generation
Once the key parameter file has been created, the next step is to create the signed certificate using an existing certificate in the key database file to sign the server certificate. From the Key Management Menu, select 1 - Manage keys and certificates to display the Key and Certificate Menu. In the Key and Certificate Menu, choose option 10 to create a signed certificate and key. This requires the displayed certificate to have signing capability.

Select the certificate type by choosing option 9, user or server certificate with 1024 Diffie-Hellman key from the Certificate Type Menu.

Figure 27. Creating a key parameter file to be used with Diffie-Hellman
Once the certificate type is determined, you will be prompted to enter:

- key parameter file created previously
- a label to uniquely identify the key and certificate within the key database
- the individual fields within the subject name
- certificate expiration (Valid expiration range is 1 to 9999 days. Default value is 365 days)
- the subject alternate names (optional).

Figure 28. Creating a certificate to be used with Diffie-Hellman

Once the certificate is created, the next step is to determine whether the certificate should be marked as the database's default certificate. Setting the certificate as the default certificate allows the certificate to be used by the SSL APIs without having to specify its label. For more information on setting the default certificate, see “Marking a Certificate (and Private Key) as the Default Certificate” on page 412.

Sending the Certificate Request

The certificate request file can either be transferred to another system (for example, ftp as an ASCII text file) and then transferred to the certificate authority or placed directly into a mail message sent to a certificate authority using cut-and-paste methods.
In addition to the certificate request file that is generated, a request database (.rdb) file is also created or altered. The request database file will be named the same as the key database file, except it will have an extension of .rdb. For example, a key database file of key.kdb will cause a request database file of key.rdb to be created. This request database file must be saved along with the key database in order for the response for the certificate request to be successfully processed.

**Receiving the Signed Certificate or Renewal Certificate**

Once a certificate is signed by the certificate authority in response to the certificate request, you must receive it into the key database or z/OS PKCS #11 token. This is for new certificates and renewal certificates.

To receive the certificate, you must store the Base64-encoded certificate in a file on the z/OS system to be read in by the `gskkyman` command. This file should be in the current working directory when `gskkyman` is started. If this file is on another working directory you will have to specify the fully qualified name.

**Note:** In order to receive the certificate the CA certificate must also exist in the key database or z/OS PKCS #11 token. To store a CA certificate, refer to “Importing a Certificate from a File as a Trusted CA Certificate” on page 421.

To receive a certificate issued on your behalf, from the **Key Management Menu**, see Figure 11 on page 391 and enter option 5.
You are prompted for the name of the file that contains the Base64-encoded certificate that was returned to you by the certificate authority in response to a previously submitted certificate request. After receiving the certificate, you press Enter to continue working with the **Key Management Menu**. Upon completion of this step and prior to the System SSL APIs using the certificate during the SSL handshake processing, you need to determine whether the certificate should be marked as the database's default certificate. Setting the certificate as the default certificate allows the certificate to be used by the SSL APIs without having to specify its label. For more information on setting the default certificate, see [“Marking a Certificate (and Private Key) as the Default Certificate” on page 412](#).

### Managing Keys and Certificates

Once certificates are added to the key database or z/OS PKCS #11 token, these are some common operations that can be performed with the certificates.

- Show certificate/key information
- Mark a certificate (and private key) as the default certificate for the key database or z/OS PKCS #11 token
- Export a certificate to a file, key database or z/OS PKCS #11 token
- Remove a certificate (and private key) from a key database or z/OS PKCS #11 token
- Change a certificate label

### Showing Certificate/Key Information

It is sometimes useful to display the information contained in the certificates that are stored in the key database. The information displayed includes, among others, the label, issuer/subject name, the version number of the certificate, the key size for the public/private key pair, and the expiration date.

To list information about certificates that contain private keys, from the **Key Management Menu** or **Token Management Menu** (see [Figure 11 on page 391](#)) select 1, (Manage keys and certificates). This displays the **Key and Certificate List**.

---

**Key and Certificate List**
- Database: `/home/sufwll/ssl_cmd/mykey.kdb`
- Expiration Date: 2008/12/02 10:11:12
- 1 - Test Server Cert
- 2 - Server Cert
- 0 - Return to selection menu

Enter label number (ENTER to return to selection menu, p for previous list): 2 <enter>

**Token Key and Certificate List**
- Token: TOKENABC
- 1 - Test Server Cert
- 2 - Server Cert
- 0 - Return to selection menu

Enter label number (ENTER to return to selection menu, p for previous list): 2 <enter>

---

**Figure 30. Key and Certificate List**

Select the number corresponding to the label for which you would like to display certificate/key information. The **Key and Certificate Menu** for the label you chose displays next (see [Figure 31 on page 410](#)).

---

Chapter 10. Certificate/Key Management
On the Key and Certificate Menu or the Token Key and Certificate Menu, you could choose 1 to display certificate information. This accesses the Certificate Information menu (see Figure 32 on page 411):
For a z/OS PKCS #11 certificate, the Record ID and Issuer Record ID will be N/A.

From the Certificate Information screen, you can also enter 1 to display certificate extensions:

```plaintext
Certificate Information

Label: Server Cert
Record ID: 13
Issuer Record ID: 13
Trusted: Yes
Version: 3
Serial number: 3c73c6d0000e8076
Issuer name: My Server Certificate
ID
IBM
Endicott
NY
US
Subject name: My Server Certificate
ID
IBM
Endicott
NY
US
Effective date: 2002/02/20
Expiration date: 2002/10/22
Signature algorithm: shalWithRsaEncryption
Issuer unique ID: None
Subject unique ID: None
Public key algorithm: rsaEncryption
Public key size: 1024
Public key: 30 81 89 02 81 81 00 E5 19 BF 6D A3 56 61 20 99
4B 71 F6 67 DE 89 BD EB B7 9E 86 BD 0A 91 0E FA
38 25 AF 46 88 82 ES 73 AB AO 98 24 5D BD 0D 1F CC
65 6E 0C BD 00 56 04 18 BD 9A 06 9B 10 A1 73 DF
B4 58 39 6B 6E C1 F6 15 D5 A8 A3 3F AA 12 06 BD
31 AC 7F BD 34 D7 BF 34 67 88 09 CD 14 11 E2 4E
45 56 69 1F 7B 02 8D DA Dc 47 91 29 88 36 C9 63
5C C5 EB 07 20 B7 78 A1 B7 32 80 78 30 BA 2A 2F
31 AA EE A3 67 DA 02 03 01 00 01

Number of extensions: 4

Enter 1 to display extensions, 0 to return to menu: 1 <enter>
```

Figure 32. Certificate Information

Note: For a z/OS PKCS #11 certificate, the Record ID and Issuer Record ID will be N/A.

From the Certificate Information screen, you can also enter 1 to display certificate extensions:

```plaintext
Certificate Extensions List

1 - subjectKeyIdentifier
2 - authorityKeyIdentifier
3 - keyUsage (critical)
4 - basicConstraints (critical)

Enter extension number (press ENTER to return to previous menu): 3 <enter>
```

Figure 33. Certificate Extensions List

Enter 3 on the Certificate Extensions List to show key usage information:
To display key information, from the Key and Certificate Menu or Token Key and Certificate Menu, choose 2, Show Key Information. This accesses the Key Information menu (see Figure 35):

```
Figure 34. Key Usage Information

Figure 35. Key Information menu
```

Note: For a z/OS PKCS #11 certificate, the Record ID and Issuer Record ID will be N/A.

Marking a Certificate (and Private Key) as the Default Certificate

Once a certificate has been added to the key database or z/OS PKCS #11 token through either a certificate request or as a self-signed certificate, it can be marked as the default certificate. Marking a certificate as the default certificate allows it to be used by the programs calling the System SSL APIs without having to explicitly supply the certificate's label.

To mark a certificate as the default certificate for the key database, from the Key Management Menu or Token Management Menu (see Figure 11 on page 391), choose 1, (Manage keys and certificates), and on the Key and Certificate List (see Figure 30 on page 409), choose the label number you want to work with. The Key and Certificate Menu or Token Key and Certificate Menu displays:
Choose 3 to set the certificate and private key as the default certificate for the key database or z/OS PKCS #11 token.

Copying a Certificate (and Private Key) to a Different Key Database or z/OS PKCS #11 Token

Once your certificates have been created, it may be necessary for you to transfer a certificate to another key database or z/OS PKCS #11 token on your system or a remote system. This transfer may be necessary for these reasons:

- The remote system requires the signing certificate to be in its key database or z/OS PKCS #11 token for validation purposes. The certificate does not need to contain the private key information. These certificates are normally certificate authority (CA) certificates but may also be a self-signed certificate.
- The server or client certificate is being used by another application in a separate key database file or z/OS PKCS #11 token.

Note: The source key database file or z/OS PKCS #11 token and the target key database file or z/OS PKCS #11 token must exist before the certificate can be copied. If the target is a z/OS PKCS #11 token, then only certificates with RSA keys of 1024 or 2048 bits may be copied. If the target is a FIPS database, then only a FIPS database can be the source.

Copying a Certificate Without its Private Key

To copy a certificate to a different platform or to a different system without its private key (certificate validation), from the Key Management Menu or the Token Management Menu, select 1 - Manage keys and certificates to display the Key and Certificate List or the Token Key and Certificate List respectively. Find the label of the certificate to be copied and enter the number associated with the label. In the Key and Certificate Menu or the Token Key and Certificate Menu, enter option 6 to export the certificate to a file. The Export File Format menu appears:
You are then prompted for what file format you would like for the exported certificate information.

The file format is determined by the support on the receiving system. When the receiving system implementation is z/OS System SSL V1R2 or earlier, the selected format must be one of the ASN.1 DER formats.

After selecting the export format, you will be asked for a file name. You can now transfer this file to the system and import the certificate. If copying to a remote system, this file can now be transferred (in binary if option 1 or 3 has been selected or in ASCII (TEXT) if option 2 or 4 has been selected) to the remote system. For information on receiving the certificate into the key database file or z/OS PKCS #11 token, see “Importing a Certificate from a File as a Trusted CA Certificate” on page 421. Upon successfully receiving the certificate, the certificate can now be used to validate the SSL’s partner certificate. This means that a client with the imported certificate can now validate the server’s certificate, while a server with the imported certificate can validate the client’s certificate when client authentication is requested.

You will also need to determine whether the certificate should be marked as the default certificate. Setting the certificate as the default certificate allows the certificate to be used by the SSL APIs without having to specify its label. For more information on setting the default certificate, see “Marking a Certificate (and Private Key) as the Default Certificate” on page 412.

**Copying a Certificate with its Private Key**

To copy a certificate to a different key database format or to a different system with its private key, the certificate must be exported to a PKCS #12 formatted file. PKCS #12 files are password-protected to allow encryption of the private key information. From the Key Management Menu or Token Management Menu, select 1 - Manage keys and certificates to display a list of certificates with private keys. Find the label of the certificate to be copied and enter the number associated with the label. In the Key and Certificate Menu or Token Key and Certificate Menu, enter option 7 to export the certificate and private key to a file.

The Export File Format menu appears:
The second display applies to z/OS PKCS #11 tokens.

You will then be prompted for what file format you would like for the exported certificate information. The file format is determined by the support on the receiving system. In most cases the format to be used is Binary PKCS #12 Version 3. When the receiving system implementation is z/OS System SSL V1R2 or earlier, the selected format must be Binary PKCS #12 Version 1. z/OS PKCS #11 tokens only support Version 3 PKCS #12 export. Export from a FIPS database must be PKCS #12 Version 3 using strong encryption.

After selecting the export format, you will be asked for a file name and password. You then will receive a message indicating that the certificate was exported. You can now transfer this file to the system and import the certificate into the key database file or z/OS PKCS #11 token. If copying to a remote system, this file can now be transferred (in binary) to the remote system. For information on receiving the certificate into the key database file, see "Importing a Certificate from a File with its Private Key" on page 423. Upon successfully receiving the certificate, the certificate can now be used to identify the program. For example, the certificate can be used as the SSL server program's certificate or it can be used as the SSL client program's certificate.

**Copying a Certificate and its Private Key to a Key Database on the Same System**

To copy a certificate and its private key from one key database to another key database on the same system, you will need to know the target key database file name and password. If the source database is a FIPS database, then the target database must also be a FIPS database. If the source database is a non-FIPS database, then the target must also be a non-FIPS database. From the **Key Management Menu**, select 1 - Manage keys and certificates to display the **Key and Certificate Menu**. Find the label of the certificate to be copied and enter the number associated with the label. From the **Key and Certificate Menu**, enter 5 to copy a certificate and key to another database:
You will then be prompted for the target key database name, and the target key database password. Once the certificate is copied to the other key database file, you will receive a message indicating that the certificate has been successfully copied.

**Copying a Certificate and its Private Key from a z/OS PKCS #11 Token on the Same System**

To copy a certificate and its private key from a z/OS PKCS #11 token to another z/OS PKCS #11 token or key database file on the same system, from the **Token Management Menu**, select **1 - Manage Keys and Certificates** to display the Token Key and Certificate List. Find the label of the certificate to be copied and enter the number associated with the label. From the **Token Key and Certificate Menu** enter **5** to copy a certificate and key to another token or a key database file. If the target is a key database on the same system, you will need to know the target's file name and password.
You will then be prompted to choose either a z/OS PKCS #11 token or a key database as the target of the copy. Figure 40 shows the prompts if a z/OS PKCS #11 token is chosen as the target. Once the certificate is copied, you will receive a message indicating that the certificate has been successfully copied.

Removing a Certificate (and Private Key)
You may want to remove a certificate if:
- The certificate has expired and is no longer useful.
- The certificate has been exported to a different key database or z/OS PKCS #11 token and is no longer needed in the current database or token.

Caution: Once you delete a certificate/private key pair, it cannot be recovered unless it has previously been stored somewhere else (another key database file, z/OS PKCS #11 token, a PKCS #12 file for certificate/private key pairs, or a DER-encoded or Base64-encoded file for certificates). Be sure you no longer require the certificate (and private key if one is associated with the certificate) before you remove it.

From the Key Management Menu or Token Management Menu, select 1 - Manage keys and certificates to display the Key and Certificate List or Token Key and Certificate List respectively. Find the label of the certificate and key to be deleted and enter the number associated with the label. From the Key and Certificate Menu or Token Key and Certificate Menu (see Figure 41 on page 418), choose 8 to delete the certificate and key.
Enter 1 to confirm the deletion of the certificate and key. A message appears, confirming that the record has been deleted. Once the certificate has been deleted, it can no longer be used for identification or verification purposes by the System SSL APIs during SSL handshake processing.

Changing a Certificate Label

Find the certificate label to be changed and enter the number associated with the label. In the Key and Certificate Menu or Token Key and Certificate Menu (see Figure 42), choose 9 to change the label:

Enter 1 to confirm delete, 0 to cancel delete: 1 <enter>
Record deleted.
Press ENTER to continue.

Figure 42. Changing a Certificate Label

Enter the new label name and press Enter. A message confirms that the label name has been changed.

Creating a Signed Certificate and Key

Creating a signed certificate and key allows for a fastpath method for creating a signed certificate that resides in the same key database file or z/OS PKCS #11 token as the displayed signing Certificate Authority certificate. From the Key Management Menu or Token Management Menu, select 1 - Manage keys and certificates to display the Key and Certificate List or Token Key and Certificate List.
respectively. Find the label of the signing Certificate Authority certificate and enter the number associated with the label. From the Key and Certificate Menu or Token Key and Certificate Menu (see Figure 43), choose option 10 to create a signed certificate and key.

**Note:** This requires the displayed certificate to have signing capability.

---

The **Certificate Type** menu appears. Once the certificate type is determined, you will be prompted to enter:

- a label to uniquely identify the key and certificate within the key database or z/OS PKCS #11 token
- the individual fields within the subject name
- certificate expiration. The valid range for a self-signed certificate is 1 to 9999 days. The default is 365 days.
Press option 0 to continue or option 1 to specify the subject alternate names. If option 1 is selected, the Subject Alternate Name Type menu appears.

When specifying subject alternate names, you are prompted for the type of the alternate name. After the alternate name type is determined, you will be prompted to enter:

- the individual fields within the subject name.
After the individual fields are completed, enter option 0 to continue or option 1 to specify another subject alternate name (repeat the process).

Creating a Certificate Renewal Request

Certificate renewal requests allow for existing signed certificates that are nearing their expiration dates to be renewed without having to create a brand new certificate request. The renewed certificate continues to contain the same subject name, public/private key pair. From the Key Management Menu or Token Management Menu, select 1 - Manage keys and certificates to display the Key and Certificate List or Token Key and Certificate List respectively. Find the label of the certificate to be renewed and enter the number associated with the label. From the Key and Certificate Menu or Token Key and Certificate Menu (see Figure 46 on page 421), choose option 11 to create a certificate renewal request.

Enter request file name (press ENTER to return to menu). The certificate request is created. Press enter to continue. Once the certificate renewal request is created, follow the same steps as a new certificate request. See "Sending the Certificate Request" on page 407 and "Receiving the Signed Certificate or Renewal Certificate" on page 408.

Importing a Certificate from a File as a Trusted CA Certificate

If you are using a certificate authority for generating your certificates that is not one of the default certificate authorities for which certificates are already stored in the key database, or if you are using a z/OS PKCS #11 token for which no default certificates exist, then you must import the certificate authority’s certificate into your key database file or z/OS PKCS #11 token before you use the System SSL APIs. If you are using client authentication, then the CA certificate must be imported into the key database or z/OS PKCS #11 token of the server program. The client program’s key database file or z/OS PKCS #11 token must have the CA certificate imported regardless of whether or not the SSL connection uses client authentication.

If you are using a self-signed certificate as the SSL server program’s certificate and your SSL client program is also using the System SSL APIs, then you must import the server’s self-signed certificate without its private key into the client program’s key database file or z/OS PKCS #11 token.

If you are using a self-signed certificate as the SSL client program’s certificate and your SSL server program is also using the System SSL APIs with client authentication requested, then you must import the client’s self-signed certificate without its private key into the server program’s key database file or z/OS PKCS #11 token.
If the CA certificate being imported was signed by another CA certificate, the complete chain must be present in the key database file or z/OS PKCS #11 token prior to the import.

If using a key database file, a number of well-known certificate authority (CA) certificates are stored in the key database when the key database is created. To get a certificate list, select 2 - Manage certificates from the Key Management Menu. Figure 47 and Figure 48 contain lists of CAs for which certificates are stored on key database creation:

To import a certificate without a private key into your key database file or z/OS PKCS #11 token, first get the certificate in a file with the file in either Base64-encoded, Binary encoded or PKCS #7 format. From the Key Management Menu or the Token Management Menu enter 7 to import a certificate:
You will be prompted to enter the certificate file name and your choice of a unique label that will be assigned to the certificate.

Once the certificate is imported, you will receive a message indicating the import was successful. The certificate is treated as “trusted” so that it can be used in verifying incoming certificates. For a program acting as an SSL server, this certificate is used during the verification of a client’s certificate. For a program acting as an SSL client, this certificate is used to verify the server’s certificate which is sent to the client during SSL handshake processing.

**Importing a Certificate from a File with its Private Key**

To store a certificate into a different key database format or to a different system with its private key, the certificate must be exported from the source system into a PKCS #12 format file (See “Copying a Certificate with its Private Key” on page 414 for more information). PKCS #12 files are password-protected to allow encryption of the private key information. If the CA certificate being imported was signed by another CA certificate, the complete chain must be present in the key database file or z/OS PKCS #11 token prior to the import. From the **Key Management Menu** or **Token Management Menu**, enter 8 to import a certificate and a private key:
You will be prompted to enter the certificate file name, password and your choice of a unique label to be assigned to the certificate.

Once the certificate is imported, you will receive a message indicating that import was successful. The next step is to determine whether the certificate should be marked as the database's or token's default certificate. Setting the certificate as the default certificate allows the certificate to be used by the SSL APIs without having to specify its label. For more information on setting the default certificate, see “Marking a Certificate (and Private Key) as the Default Certificate” on page 412.

A certificate and key can be imported into a FIPS key database providing it is a PKCS #12 Version 3 with strong encryption format. When adding certificates from the import file to a FIPS key database file only certificates signed with FIPS signature algorithms using FIPS-approved key sizes may be imported. When processing a chain of certificates, processing of the chain will terminate if a non-FIPS certificate is encountered. Certificates processed prior to the failing certificate will be added to the key database file. It is the responsibility of the importer to ensure that the file came from a source meeting FIPS 140-2 criteria in order to maintain adherence to the FIPS criteria.

Using gskkyman to be Your Own Certificate Authority (CA)

The gskkyman command provides the capability for you to act as your own Certificate Authority (CA). Being your own CA allows you to sign your own or anyone else’s certificate requests. This is very handy if you only need certificates within your private Web network and not for outside Internet commerce.

To be your own CA in a Web network, you must create a CA database and self-signed CA certificate using gskkyman. A server or client that wishes for you to sign a certificate must supply you with their certificate request. After signing the certificate, the server or client must receive the CA certificate and the newly signed certificate. The CA-signed certificate must then be received into either the client or server key database.

This table describes the steps needed to become your own CA to allow secure communication between a client and a server. This example reflects the steps followed when the CA is on a different system or is a different user than the issuer of the certificate request.
<table>
<thead>
<tr>
<th>Certificate Authority (System A)</th>
<th>Server or Client (System B)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 1 - Create a key database</strong></td>
<td><strong>Step 1 - Create a key database</strong></td>
</tr>
<tr>
<td>Create a key database using the <code>gskkyman</code> command:</td>
<td>Create a key database using the <code>gskkyman</code> command:</td>
</tr>
<tr>
<td>• From the <strong>Database Menu</strong>, select option 1 - <strong>Create new database</strong></td>
<td>• From the <strong>Database Menu</strong>, select option 1 - <strong>Create new database</strong></td>
</tr>
<tr>
<td>See “Creating, Opening and Deleting a Key Database File” on page 388 for details.</td>
<td>See “Creating, Opening and Deleting a Key Database File” on page 388 for details.</td>
</tr>
</tbody>
</table>

| **Step 2 - Create a Root Certificate Authority certificate** | **Step 2 - Create a Root Certificate Authority certificate** |
| Create a Certificate Authority certificate: | No action required. |
| • From the **Key Management Menu**, select option 6 - **Create a self-signed certificate** | |
| • From the **Certificate Type** menu, select one of the CA values for your certificate type | |
| See “Creating a Self-Signed Server or Client Certificate” on page 399 for details. | |

| **Step 3 - Create a certificate request** | **Step 3 - Create a certificate request** |
| No action required. | Create a certificate request: |
| | • From the **Key Management Menu**, select option 4 - **Create new certificate request** |
| | • From the **Certificate Type** menu, select one of the certificate types |
| | See “Creating a Certificate Request” on page 401 for details. |

| **Step 4 - Send the certificate request to the CA** | **Step 4 - Send the certificate request to the CA** |
| No action required. | Send the certificate request to the CA: See “Sending the Certificate Request” on page 407. |

<table>
<thead>
<tr>
<th><strong>Step 5 - Sign the certificate request</strong></th>
<th><strong>Step 5 - Sign the certificate request</strong></th>
</tr>
</thead>
</table>

Chapter 10. Certificate/Key Management 425
### Certificate Authority (System A)

Before signing a certificate for a client or server, you need to make sure that the requestor has a legitimate claim to request the certificate. After you have verified the claim, you can create a signed certificate.

To sign the certificate request, the **gskkyman** command must be issued using command-line options (see "gskkyman Command Line Mode Syntax" on page 427 for a description of the options). The **gskkyman** command must be issued with these parameters:

- `gskkyman -g -x num-of-valid-days -cr certificate-request-file-name -ct signed-certificate-file-name -k CA-key-database-file-name -l label`

**Example:** This command will allow you to sign a request certificate and allow the certificate to be valid for 360 days.

```
gskkyman -g -x 360 -cr server_request.arm -ct server_signed_cert.arm -k CA.kdb -l labelname
```

After you have entered the command, you will be prompted to enter the database password.

**Notes:**
1. The signed certificate will be an end user certificate unless the `-ca` option is specified.
2. The filename specified on the `-ct` option is created for you by the utility, and is the actual signed certificate file.
3. The valid certificate lifetime range is between 1 and 9999 days. The certificate end date will be set to the end date for the CA certificate if the requested certificate lifetime exceeds the CA certificate lifetime.

### Server or Client (System B)

No action required.

### Step 6 - Send the signed CA certificate and the newly signed certificate to the requestor

Export the signed CA certificate (created in Step 2) to a Base64 file (DER or PKCS #7) See "Copying a Certificate Without its Private Key" on page 413. Send (for example, without its private key ftp) the Base64 file and the newly signed certificate (created in Step 4) to the requestor.

No action required.

### Step 7 - Import the CA certificate

No action required.

Import the CA certificate. See "Importing a Certificate from a File as a Trusted CA Certificate" on page 421.

### Step 8 - Receive the signed certificate

No action required.

Receive the signed certificate. See "Receiving the Signed Certificate or Renewal Certificate" on page 408.

**Note:** Depending upon the SSL application, you may need to either send the CA certificate to the client, or the server application may actually present the certificate to the client for them during SSL session setup.
Migrating from Key Database Files to z/OS PKCS #11 token

If you need to migrate keys and certificates stored in an existing key database into a z/OS PKCS #11 token, follow these steps:

1. Export the certificate/private key to a password protected PKCS #12 file using gskkyman. Refer to "Copying a Certificate with Its Private Key" on page 414 for details on the steps for exporting certificates/private keys to a PKCS #12 file.

2. Import the certificate/private key from the PKCS #12 file into the z/OS PKCS #11 token using gskkyman. Refer to "Importing a Certificate from a File with its Private Key" on page 423.

Migrating Key Database Files to RACF Key Rings

If you need to migrate keys and certificates stored in an existing key database into a RACF key ring, follow these steps:

1. Export the certificate/private key to a password protected PKCS #12 file using gskkyman. Refer to "Copying a Certificate with Its Private Key" on page 414 for details on the steps for exporting certificates/private keys to a PKCS #12 file.

2. Copy the newly created PKCS #12 file to a z/OS dataset.

3. Use the RACDCERT command with the ADD operand and the dataset name created in step 2 to add the certificate/private key to the RACF database. The certificate should be added as TRUSTED. If the private key is to be stored in the ICSF PKDS, the ICSF keyword also needs to be specified on the RACDCERT command.

4. Use the RACDCERT command with the ADDRING operand to create a new key ring in RACF. Use the RACDCERT command with the CONNECT operand to add the certificate/private key to one or more existing RACF key rings.

gskkyman Command Line Mode Syntax

This topic describes the format and options of the gskkyman command.
gskkyman

The gskkyman command is used for key database management and z/OS PKCS #11 token management.

Format

```sh
gskkyman

# For command syntax
-gskkyman -h

# Display certificate details
-gskkyman -dc -k filename -l label
-gskkyman -dc -t token-name -l label

# Display certificate verbose details
-gskkyman -dcv -k filename -l label
-gskkyman -dcv -t token-name -l label

# Display key database expiration and record length
-gskkyman -dk -k filename

# Export a certificate and its associated private key
-gskkyman -e -k filename -l label -p filename
-gskkyman -e -t token-name -l label -p filename

# Sign a certificate for a certificate request
-gskkyman -g -x days -cr filename -ct filename -k filename -l label -ca -ic
-gskkyman -g -x days -cr filename -ct filename -t token-name -l label -ca -ic

# Display the command syntax
-gskkyman -h

# Import a certificate and its associated private key
-gskkyman -i -k filename -l label -p filename
-gskkyman -i -t token-name -l label -p filename

# Store the database password in the stash file
-gskkyman -s -k filename
```

Parameters

**function**

The function to be performed. It must follow the command name. The acceptable values are:

- **-dc** Display certificate details
- **-dcv** Display certificate verbose details
- **-dk** Display key database expiration and record length
- **-e** Export a certificate and its associated private key
- **-g** Sign a certificate for a certificate request
- **-h** Display the command syntax
- **-i** Import a certificate and its associated private key
- **-s** Store the database password in the stash file

**option**

The parameters necessary to accomplish the function. If the option provides a value, then the value must follow the option:

The acceptable values are:

- **-ca** A certification authority certificate will be generated if -ca is specified. An end user certificate will be generated if -ca is not specified.
- **-cr** Specifies the name of the certificate request file. You will be prompted for the file name if this option is not specified.
- **-ct** Specifies the name of the output generated signed certificate file. You will be prompted for the file name if this option is not specified. You may specify any name. If you specify an existing file name, the file will be overwritten.
- **-ic** The certification chain certificates will be included in the certificate file if -ic is specified. Otherwise, just the signed certificate will be included in the certificate file.
- **-k** Specifies the name of the key database. This option is mutually exclusive with the -t option. You will be prompted for the file name if this option is not specified. The length of the
fully-qualified file name cannot exceed 251 characters. If the file name does not end with an
extension of 1-3 characters, the length of the fully-qualified file name cannot exceed 247
characters. Finally, the key database name cannot end with .rdb or .sth.

-\ I \ Specifies the certificate label. The label must be enclosed in double quotes if it contains one or
more spaces. The label for the default key will be used if this option is not specified (export or
sign function) or you will be prompted for the label (import function). If more than one
certificate with the specified label exists (can occur for tokens), the user is prompted to either
cancel or choose the required certificate from a list that summarizes significant fields in the
certificate.

-\ p \ Specifies the name of the PKCS #12 file. You will be prompted for the file name if this option
is not specified.

-\ t \ Specifies the name of the token to be managed. This option is mutually exclusive with the -\ k
option. The name must consist of characters that are alphanumeric, national (@ x5B, # x7B, $
x7C) or period (.x4B). The first character must be alphabetic or national. Lowercase letters are
permitted but are folded to uppercase.

-\ x \ Specifies the number of days until the signed certificate expires and must be between 1 and
9999 days. The certificate will expire in 365 days if this option is not specified.

Results
If gskkyman is specified with no arguments the interactive menu-driven interface is used.

Usage
The gskkyman command is used to manage a token or a key database and its associated request
database. Interactive menus will be displayed if no command options are specified. Otherwise, the
requested token/database function will be performed and the gskkyman command will exit.

If the command specifies the -\ t \ (token name) option, then the requested function is performed for the
identified token.

If the command does not specify the -\ t \ option, then it is assumed that the function is to be performed for a
key database. If the -\ k \ option and the -\ t \ option are not supplied, the user is prompted for a key database
filename.

If both -\ k \ and -\ t \ are specified, the command is rejected and an error message is displayed.

For commands applied to a key database:

The key database contains certificates and private keys and normally has a filename extension of '.kdb'.
The request database contains requests for new certificates and always has a filename extension of '.rdb'.
The database stash file contains the masked database password and always has a filename extension of
'.sth'. Access to these files should be restricted to the database owner.

A certificate or request database consists of fixed-length records. The record length is specified when the
database is created and must be large enough to contain the largest certificate entry. A record length of
5000 should be sufficient for most applications. The record length can be increased if necessary after the
database has been created.

A temporary database file is created when a database is updated during gskkyman processing. The
temporary database file is created using the same name as the database file with " .new " appended to the
name. The database file is then rewritten and the temporary database file is deleted upon successful
completion of the rewrite operation. The temporary database file will not be deleted if an error occurs while
rewriting the database file. If this happens, you can replace the database file with the temporary database
file in order to recover from the error. If an error does occur and you do not rename or delete the temporary file, you will get an error on the next database update operation indicating the backup file already exists.

If all certificates in a key database are displayed with the -dc or -dcv command, then all certificates with private keys are outputted, followed by all certificates without private keys. When displaying all certificates in a token, the certificates are displayed in the order returned from the token so that certificates with private keys may be interspersed with certificates without private keys.
gskkyman Command Line Mode Examples

Command mode is entered when the gskkyman command is entered with parameters. The requested
token/database function will be performed and then the command will exit.

- Store the database password in the stash file
  
gskkyman -s -k filename

  The database password is masked and written to the key stash file. The file name is the same as the
  key database file name but has an extension of '.sth'. You will be prompted for the key database file
  name if the '-k' option is not specified. The '-t' option is invalid for the '-s' function.

- Export a certificate and the associated private key
  
gskkyman -e -k filename -l label -p filename

  The certificate and associated private key identified by the record label are exported to a file in PKCS
  #12 Version 3 format using strong encryption. The default key will be exported if the '-l' option is not
  specified. You will be prompted for the key database file name if the '-k' and the '-t' option is not
  specified. You will be prompted for the export file name if the '-p' option is not specified.

- Import a certificate and associated private key
  
gskkyman -i -t token-name -l label -p filename

  A certificate and associated private key are imported from a file in PKCS #12 format. You will be
  prompted for the label if the '-l' option is not specified. You will be prompted for the key database file
  name if the '-k' and the '-t' option is not specified. You will be prompted for the import file name if the '-p'
  option is not specified.

- Create a signed certificate for a certificate request
  
gskkyman -g -x days -cr filename -ct filename -k filename -l label -ca -ic

  The certificate request identified by the -cr parameter is processed and a signed certificate is created
  and written to the certificate file identified by the -ct parameter. The -x parameter specifies the number
  of days until the certificate expires and defaults to 365 days. The certificate is signed using the default
  key if the -l parameter is not specified. You will be prompted for the key database file name if the '-k'
  option is not specified. You will be prompted for the certificate request file name if the '-cr' option is not
  specified. You will be prompted for the signed certificate file name if the '-ct' option is not specified.
  The signed certificate will be an end user certificate unless the -ca option is specified. A certification
  authority certificate will have basic constraints and key usage extensions which allow the certificate to
  be used to sign other certificates and certificate revocation lists. An end user certificate will have basic
  constraints and key usage extensions which allow the certificate to be used for authentication, digital
  signatures, and data encryption (a DSA key cannot be used for data encryption).

  Any certificate can be used to sign the new certificate as long as the certificate has a private key, the
  basic constraints certificate extension (if present) has the CA indicator set, and the key usage certificate
  extension (if present) allows signing certificates. However, depending upon how the new certificate is
  subsequently used, it may fail the validation checking if the signing certificate is not a valid certification
  authority certificate.

  The signing algorithm that will be used to sign the new certificate is derived from the key algorithm of
  the signing certificate and the most secure and compatible SHA-based hash in use in either the signing
  certificate or the certificate request. Possible signing algorithms are:
  - x509_alg_sha1WithRsaEncryption
  - x509_alg_sha224WithRsaEncryption
  - x509_alg_sha256WithRsaEncryption
  - x509_alg_sha384WithRsaEncryption
  - x509_alg_sha512WithRsaEncryption
  - x509_alg_dsaWithSha1

  The certificate file will contain the generated X.509 certificate in DER-encoded Base64 format if the '-ic'
  option is not specified. The certificate file will contain the generated X.509 certificate and the certification
  chain certificates as a PKCS #7 message in Base64 format if the '-ic' option is specified.
• Display all certificates in a key database
  
gskkyman -dc -k filename

After being prompted for the key database password, the certificates will be displayed. You will be
prompted for the key database file name if the -k option is not specified. Due to the number of
certificates that can exist in a key database file, it is recommended that you redirect the output to a file.
This will allow for easy review of the certificates as well as any post-processing of the certificate output.

• Display key database expiration date:
  
gskkyman -dk -k filename

After being prompted for the key database password, the full key database path and file name,
extpiration date and record length are displayed. You will be prompted for the key database file name if
the -k option is not specified.

**gskkyman Command Line Mode Displays**

Command mode is entered when the **gskkyman** command is entered with parameters. The requested
token/database function will be performed and then the command will exit.

• gskkyman command-mode key database file display

  When the key database password is correctly entered:

  **Command:**
  
gskkyman -dk -k example.kdb

  **Output:**
  
  Database: /home/sufwl1/ssl_cmd/example.kdb
  Expiration Date: 2008/12/02 10:11:12
  Record length: 5000

• gskkyman command-mode certificate display

  **Command:**
  
gskkyman -dc -k example.kdb -l 'Test User'

  **Output for a single certificate:**

  **Label:**
  
  <Test User>

  **Trusted:**
  
  Yes

  **Version:**
  
  3

  **Serial number:**
  
  45ac4d2300a6023

  **Issuer's Name:**
  
  <CN=Test CA,OU=Test unit,O=IBM, L=Endicott, ST=NY, C=US>

  **Subject's Name:**
  
  <CN=Test User,OU=Test unit,O=IBM, L=Endicott, ST=NY, C=US>

  **Effective date:**
  
  2007/01/16 21:02:02

  **Expiration date:**
  
  2008/01/16 21:02:02

  **Signature algorithm:**
  
  sha1WithRsaEncryption

  **Issuer unique ID:**
  
  None

  **Subject unique ID:**
  
  None

  **Public key algorithm:**
  
  rsaEncryption

  **Public key size:**
  
  1024

  **Public key:**
  
  30 81 89 02 81 81 00 9A 9A BC 53 49 50 8B AF F9
  AF 00 A1 F3 A6 80 3A DA 2C A5 7C 65 A0 00 96 FA
  1A 71 74 74 B4 2A 95 92 AC 1D 76 F1 97 37 03 BC
Chapter 10. Certificate/Key Management

- gskkyman command-mode certificate display (verbose)

Command:
gskkyman -dcv -k example.kdb -l 'Test User'

Verbose output for a single certificate:

Label: <Test User>
Trusted: Yes
Version: 3
Serial number: 45ac4d2300a6023
Issuer's Name: <CN=Test CA,OU=Test unit,O=IBM, L=Endicott, ST=NY, C=US>
Subject's Name: <CN=Test User,OU=Test unit,O=IBM, L=Endicott, ST=NY, C=US>
Effective date: 2007/01/16 21:02:02
Expiration date: 2008/01/16 21:02:02
Signature algorithm: sha1WithRsaEncryption
Issuer unique ID: None
Subject unique ID: None
Public key algorithm: rsaEncryption
Public key size: 1024
Public key:
30 81 89 02 81 81 00 9A 9A BC 53 49 50 8B AF F9
AF 00 A1 F3 A6 80 3A DA 2C A5 7C 65 A0 00 96 FA
1A 71 74 74 B4 2A 95 92 AC 1D 76 F1 97 37 D3 BC
06 8B DC 83 2F 7F 08 B0 EA 1F F8 71 AC 8F 96 3E
6E DA F5 F8 D0 A6 51 A4 AF E6 21 F5 50 AC B7 06
83 BF 88 48 DF 51 DB 18 BF EC 7C 72 DA ED 6C 82
2B 93 7C AE 12 E8 CD 55 16 E1 05 53 63 C1 04 D1
91 AD 3E E5 70 87 00 0C 14 40 92 D9 6E DD ED 07
81 9D 93 34 DC 1F 05 02 03 01 00 01
Private key: Yes
Default key: No
Critical Extension:
  keyUsage:
    Digital signature
    Non-repudiation
    Key encipherment
    Data encipherment
Non-critical Extension: 1
  subjectAltName:
    EMAIL:
Non-critical Extension: 2
  subjectKeyIdentifier:
    91 0A 60 24 00 31 0A 75 39 F4 F6 56 D5 AD 35 35
    86 2D C6 F8
Non-critical Extension: 3
  authorityKeyIdentifier:
    Key ID:
      19 6E 03 37 AB 8B 0F 7B 9D A3 A6 8F CC B4 A2 CA
      AC FA B6 E8
Chapter 11. SSL Started Task

The SSL started task (GSKSRVR) provides sysplex session cache support, dynamic trace support and notification when changing from hardware to software cryptography. The SSL started task is an optional component of System SSL and does not need to be configured and started in order to use System SSL.

The default home directory for the SSL started task is /etc/gskssl/server. A different home directory can be specified by changing the definition of the HOME environment variable in the GSKSRVR procedure. The SSL started task will read the envar file in the home directory to set the environment variables. This file is a variable-length file where each line consists of a variable name and variable value separated by '='. Trailing blanks are removed from the variable value. Blanks lines and lines beginning with '#' are ignored.

GSKSRVR Environment Variables

These environment variables are processed by the System SSL started task.

**GSK_LOCAL_THREADS**
- Specifies the maximum number of threads which will be used to handle program call requests from SSL applications running on the same system as the GSKSRVR started task. The default value is 5 and the minimum value is 2. The default of 5 will be used if a valid value is not specified.

**GSK_SIDCACHE_SIZE**
- Specifies the size of the sysplex session cache in megabytes and is between 1 and 512 with a default of 20. The default of 20 will be used if a valid value is not specified.

**GSK_SIDCACHE_TIMEOUT**
- Specifies the sysplex session cache entry timeout in minutes and is between 1 and 1440 with a default of 60. The default of 60 will be used if a valid value is not specified.

**GSK_FIPS_STATE**
- Specifies that the System SSL started task is to execute in FIPS mode. The only value supported is GSK_FIPS_STATE_ON. If any other value is specified, message GSK01054E is issued with a status code of zero, and GSKSRVR executes in non-FIPS mode.

In order for the started task to perform sysplex session ID caching for FIPS mode application servers, the envar file must contain GSK_FIPS_STATE=GSK_FIPS_STATE_ON. If the started task executes in FIPS mode, then message GSK01057I is output to STDOUT. See Chapter 4, “System SSL and FIPS 140-2,” on page 13 for setup requirements necessary to execute in FIPS mode.

In order to have GSKSRVR execute in non-FIPS mode and only provide sysplex session ID caching for non-FIPS application servers, remove or comment out this environment variable.

GSKSRVR will start in non-FIPS mode without issuing GSK01054E or GSK01057I messages.

Configuring the SSL Started Task

1. Create the home directory for the SSL started task (the default is /etc/gskssl/server)
2. Copy the sample envar file from /usr/lpp/gskssl/examples/gksrvr.envar to /etc/gskssl/server/envar (change the directory name to match the home directory created) and modify the LANG, TZ, and NLSPATH values to meet local installation requirements.
3. Copy the sample started procedure from GSK.SGSKSAMP(GSKSRVR) to SYS1.PROCLIB(GSKSRVR)
4. Create the GSKSRVR user and associate it with the GSKSRVR started procedure. Replace 'nnnnnn' in the ADDUSER command with a non-zero value which is not assigned to another user.
ADDUSER GSKSRVR DFLTGRP(SYS1) NOPASSWORD OMVS(UID(nnnnnn) PROGRAM(/bin/sh) HOME(/etc/gskssl/server))

RDEFINE STARTED GSKSRVR.** STDATA(USER(GSKSRVR) GROUP(SYS1) TRUSTED)

SETROPTS RACLST(STARTED) REFRESH

5. Ensure that the pdsename.SIEALNKE and CEE.SCEERUN datasets are APF-authorized and are either in the link list concatenation or are specified as a STEPLIB for the GSKSRVR procedure.

6. Optionally, set up a message processing exit to automatically start the GSKSRVR started task. The GSK.SGSKSAMP(GSKMSGXT) program is a sample message processing exit for this purpose. In order to activate the exit, add this to the appropriate MPFLSTxx member in SYS1.PARMLIB.

BPXI004I,SUP(NO),USEREXIT(STARTSSL)

This will start GSKSRVR when OMVS initialization is complete, assuming the GSKMSGXT program was linked as STARTSSL and placed in a LNKLST dataset.

7. Optionally, set up an automatic restart management (ARM) policy for the GSKSRVR started task if the default ARM policy values are not appropriate. The element type is SYSSSL and should be assigned to restart level 2. The element name is GSKSRVR_sysname. For example, the element name for the GSKSRVR started task on system DCESEC4 would be GSKSRVR_DCESEC4. Since the normal operating mode is to run the GSKSRVR started task on each system in the sysplex, the GSKSRVR started task will register with ARM to be restarted only if the started task fails and not if the current system fails. The TERMTYPE parameter of the ARM policy can be used to override this registration if desired.

---

Server Operator Commands

These operator commands are supported by the System SSL server:

**STOP GSKSRVR or P GSKSRVR**

Causes an orderly shutdown of the server.

**MODIFY GSKSRVR,parameters or F GSKSRVR,parameters**

Causes a command to be executed by the server. Some parameters are:

**DISPLAY CRYPTO**

Displays the available encryption algorithms, whether hardware cryptographic support is available and the maximum encryption key size. ‘--’ will be displayed if the encryption algorithm is not available.

This command can be abbreviated as 'D CRYPTO'

**DISPLAY LEVEL**

Displays the current System SSL service level.

This command can be abbreviated as 'D LEVEL'

**DISPLAY SIDCACHE**

Displays the current and maximum data space sizes in megabytes followed by the session cache users and the number of cache entries for each user. The count will include expired cache entries until they are removed from the cache during an update to the hash list containing the expired entry. Each GSKSRVR started task maintains its own session cache for sessions created on that system. The 'DISPLAY SIDCACHE' command must be issued for each started task to display the cache entries for the entire sysplex. This can be done by issuing 'RO "ALL,F GSKSRVR,D SIDCACHE"'.

This command can be abbreviated as 'D SIDCACHE'

**DISPLAY XCF**

Displays the status of all instances of the GSKSRVR started task in the sysplex.

This command can be abbreviated as 'D XCF'
STOP Causes an orderly shutdown of the server. This is the same as entering the "STOP GSKSRVR" command.

TRACE OFF
Turns off tracing for the System SSL started task.

TRACE ON,level
Turns on tracing for the System SSL started task. The trace output is written to the file specified by the GSK_TRACE_FILE environment variable or to the default trace file if the GSK_TRACE_FILE environment variable is not defined. The level value specifies the trace level. Refer to the descriptions of the GSK_TRACE and GSK_TRACE_FILE environment variables for more information about SSL tracing.

Sysplex Session Cache Support
The sysplex session cache support makes SSL server session information available across the sysplex. An SSL session established with a server on one system in the sysplex can be resumed using a server on another system in the sysplex as long as the SSL client presents the session identifier obtained for the first session when initiating the second session. A server executing in FIPS mode cannot resume a session cached in non-FIPS mode. SSL V3, TLS V1.0 and TLS V1.1 server session information can be stored in the sysplex session cache while SSL V2 server session information and all client session information is stored only in the local SSL cache for the application process.

A client which has established a TLS V1.0 or TLS V1.1 session with negotiated TLS extensions to a server can only be resumed on a server which supports the same set of TLS extensions established in the original session. For example, if the original session negotiates the use of the maximum fragment length TLS extension, but the session is later resumed with a server that doesn’t support the maximum fragment length TLS extension, a full re-handshake will occur.

In order to use the sysplex session cache, each system in the sysplex must be using the same external security manager (for example, z/OS Security Server RACF) and a userid on one system in the sysplex must represent the same user on all other systems in the sysplex (that is, userid ZED on System A has the same access rights as userid ZED on System B). The external security manager must support the RACROUTE REQUEST=EXTRACT,TYPE=ENVRXTR and RACROUTE REQUEST=FASTAUTH functions.

The sysplex session cache must be enabled for each application server that is to use the support. This can be done by defining the GSK_SYSPLEX_SIDCACHE environment variable or by calling the gsk_attribute_set_enum() routine to set the GSK_SYSPLEX_SIDCACHE attribute. The session information for each new SSL V3, TLS V1.0 or TLS V1.1 session created by the SSL server will then be stored in the sysplex session cache and can be referenced by other SSL servers in the sysplex. The RACF user associated with the SSL server becomes the owner of the session information. Any SSL server running with the same RACF user can access the session information. SSL servers running with a different RACF user can access the session information if they have at least READ access to the GSK.SIDCACHE.<owner> profile in the FACILITY class.

For example, session information created by RACF user APPLSRV1 can be accessed by RACF user APPLSRV2 if APPLSRV2 has READ access to the GSK.SIDCACHE.APPLSRV1 profile in the FACILITY class. These RACF commands grant this access:

RDEFINE FACILITY GSK.SIDCACHE.APPLSRV1 UACC(NONE)
PERMIT GSK.SIDCACHE.APPLSRV1 CLASS(FACILITY) ID(APPLSRV2) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH

Component Trace Support
For information about component trace support, see "Component Trace Support" on page 440.
Hardware Cryptography Failure Notification

For information about cryptographic hardware failure notification, see Chapter 3, “Using Hardware Cryptographic Features with System SSL,” on page 9.
Chapter 12. Obtaining Diagnostic Information

All of the information and techniques described in this topic are for use primarily by IBM service personnel in determining the cause of a System SSL problem. If you encounter a problem and call the IBM Support Center, you may be asked to obtain trace information or enable one or more of the diagnostic messages described here.

Any environment variables described in this topic are usually set via the UNIX System Services `export` shell command. For usage information on this command, see the `z/OS UNIX System Services Command Reference` for information on setting environment variables outside of the shell, refer to the `z/OS XL C/C++ Programming Guide` and the `z/OS Language Environment Programming Guide`.

The facilities described below are not intended for use in a production environment and are for diagnostic purposes only.

Obtaining System SSL Trace Information

You can enable the System SSL trace by using the environment variable GSK_TRACE_FILE to specify the name of the trace file, and the GSK_TRACE environment variable to set the trace level. A single trace file is created, and there is no limit on the size of the trace file.

In order to create a readable copy of the trace information, use the System SSL `gsktrace` command as follows:

```
gsktrace input_trace_file > output_trace_file
```

Capturing Trace Data Through Environment Variables

In order to capture trace information using environment variables, the trace environment variables `GSK_TRACE` and `GSK_TRACE_FILE` must be exported prior to the start of the SSL application.

• **GSK_TRACE**
  Specifies a bit mask enabling System SSL trace options. No trace option is enabled if the bit mask is 0 and all trace options are enabled if the bit mask is 0xffffffff. The bit mask can be specified as a decimal (nnn), octal (0nnnn) or hexadecimal (0xhh) value.

  These trace options are available:

  - 0x01 = Trace function entry
  - 0x02 = Trace function exit
  - 0x04 = Trace errors
  - 0x08 = Include informational messages
  - 0x10 = Include EBCDIC data dumps
  - 0x20 = Include ASCII data dumps

• **GSK_TRACE_FILE**
  Specifies the name of the trace file and defaults to /tmp/gskssl.%.trc. The trace file is not used if the `GSK_TRACE` environment variable is not defined or is set to 0.

  The current process identifier is included as part of the trace file name when the name contains a percent sign (%). For example, if `GSK_TRACE_FILE` is set to /tmp/gskssl.%.trc and the current process identifier is 247, then the trace file name will be /tmp/gskssl.247.trc.

**Note:**

Care needs to be taken if the application being traced is multi-processed. If multiple processes write to the same trace file, file corruption may occur. To allow trace information to be obtained, the trace
Obtaining Diagnostic Information

file name specified should contain a ‘%’ character in the file name. This will allow the process identifier to be placed within the file name and each process to write to its own trace file.

It is recommended that if the default trace file value is not being used, the trace file name always contain a ‘%’ character. This will eliminate the need to know if the application being traced is multi-processed or not.

Once the trace file is produced, it must be formatted. To format the file, use the System SSL gsktrace command as follows:

```
gsktrace input_trace_file > output_trace_file
```

Component Trace Support

The System SSL started task provides component trace support for any SSL application running on the same system as the GSKSRVR started task. The trace records can be written to a trace external writer or they can be kept in an in-storage trace buffer which is part of the GSKSRVR address space. IPCS is used to format and display the trace records from either a trace dataset or an SVC dump of the GSKSRVR address space. Dataset hlq.SIEAMIGE containing the SSL trace record format routine to be used by IPCS must be accessible through either a steplib or in the lnklst.

The Component Trace input command supports the option JOBSUFFIX to enable wildcarding. JOBSUFFIX can be specified as ANY or NONE with NONE being the default. If you specify JOBSUFFIX=ANY any specified jobnames of seven letters of less will be considered to be a wildcard entry and tracing will be started for jobs whose names match those entries for the length of the entry.

Refer to MVS Diagnosis: Tools and Service Aids for more information on setting up and using component trace. Refer to MVS System Commands for more information on the TRACE command. Refer to MVS IPCS User’s Guide for more information on using IPCS to view a component trace.

Capturing Component Trace Data

The component trace can be started before the job to be traced is started or while the job is running. The trace will be active for the first instance of the job. For example, if the same job name is used for multiple jobs, only the first job with that name will be traced. Subsequent jobs with the same name will not be traced unless the component trace is stopped and then restarted.

A trace external writer is required if the trace records are to be written to a dataset. A sample started procedure is shipped as GSK.GSJKSAMP(GSKWTR). Copy this procedure to SYS1.PROCLIB(GSKWTR) and modify as necessary to meet your installation requirements. This MVS operator command will start the trace external writer:

```
TRACE CT,WTRSTART=GSKWTR
```

A single SSL component trace may be active at a time and the trace can include from 1 to 16 separate jobs. The trace buffer size must be between 64K and 512K and will default to 64K.

System SSL supports these options for CTRACE:

```
OPTIONS={(LEVEL={nnn | 15})[JOBSUFFIX={NONE | ANY})
```

**LEVEL**

A bit mask specifying the types of events that System SSL is to trace. At least one of these indicators must be specified in the supplied bit mask. All trace options are enabled if the bit mask is 0xffff. The bit mask can be specified as a decimal (nnn), octal (0nnnn) or hexadecimal (0xhh) value. The SSL trace level will be set to decimal 15 if level is not specified in the CTRACE options.
These trace options are available:

- 0x01 = Trace function entry
- 0x02 = Trace function exit
- 0x04 = Trace errors
- 0x08 = Include informational messages
- 0x10 = Include EBCDIC data dumps
- 0x20 = Include ASCII data dumps

**JOBSUFFIX**

A switch specifying how the list of jobnames provided via the JOBNAME parameter are to be filtered:

- **ANY**
  
  Any specified jobnames of 7 letters or less are considered to be wildcard entries and tracing is 
  started for jobs whose names match those entries for the length of the entry.

- **NONE**
  
  Only jobs whose names match precisely one of the entries supplied in the JOBNAME parameter 
  are traced. This is the default value.

For example, to start an SSL component trace for jobs CS390IP and DB1G which includes all non-dump 
trace entries and writes the trace records using the GSKWTR trace writer:

```
TRACE CT,ON,COMP=GSKSRVR
  R n, JOBNAME=(CS390IP, DB1G), OPTIONS=(LEVEL=15), WTR=GSKWTR, END
```

To start an SSL component trace for job CICS1 which includes all trace entries and writes the trace 
records using the GSKWTR trace writer:

```
TRACE CT,ON,COMP=GSKSRVR
  R n, JOBNAME=(CICS1), OPTIONS=(LEVEL=255), WTR=GSKWTR, END
```

These commands will stop the SSL component trace and close the trace writer dataset:

```
TRACE CT,OFF,COMP=GSKSRVR
TRACE CT,WTRSTOP=GSKWTR
```

System SSL does not require a default trace member in SYS1.PARMLIB since SSL component trace is 
not activated until the operator enters the TRACE command. SYS1.PARMLIB members can be created for 
frequently used trace commands and the member name can then be specified on the TRACE command to 
avoid the operator prompt for trace options.

Starting and stopping the in-storage trace is done the same way as the external writer trace except the 
external writer name on the trace command should not be specified.

These commands will start the SSL component trace using the in-storage trace table:

```
TRACE CT,ON,COMP=GSKSRVR
  R n, JOBNAME=(CS390IP, DB1G), OPTIONS=(LEVEL=15, JOBSUFFIX=ANY), END
```

This command will stop the SSL component trace using the in-storage trace table:

```
TRACE CT,OFF,COMP=GSKSRVR
```

See *z/OS MVS System Commands* for more details on using in-storage trace.

---

**Displaying the Trace Data**

The trace records are displayed using the IPCS CTRACE command.
Obtaining Diagnostic Information

The CTRACE ENTIDLIST parameter specifies the trace entries to be included in the display. The trace entry type is the same as the SSL trace level. For example, SSL function entry trace records have entry type 1, SSL function exit trace records have entry type 2, SSL error records have entry type 4, etc. All trace entries will be included if the ENTIDLIST parameter is not specified.

The CTRACE OPTIONS parameter specifies additional filtering for the trace records. The JOB(name), PID(hexid), and TID(hexid) options can be specified to filter the trace entries based on job name, process identifier, or thread identifier. All trace entries will be included if the OPTIONS parameter is not specified.

Note that the JOBNAME parameter on the CTRACE command is used to select the address space in a dump. Since the address space is always the GSKit address space, this parameter cannot be used to filter the trace entries. Instead, you must use the OPTIONS((JOB(name))) parameter to select the component trace entries for a specific job.

For example, to display SSL function entry and SSL function exit trace records for job KRBSRV48 thread 6:

```
IPCS CTRACE COMP(GSKSRVR) ENTIDLIST(1,2) OPTIONS((JOB(KRBSRV48),TID(6))) FULL
```

A range can be specified for the entry identifiers. For example, to display just the non-dump trace records:

```
IPCS CTRACE COMP(GSKSRVR) ENTIDLIST(1:15) FULL
```

Event trace records for System SSL

The FULL format of a component trace report is as follows:

```
COMPONENT TRACE FULL FORMAT
SYSTYPE(C01)
COMP(GSKSRVR)
**** ** 11/14/2005
```

```
SYSTYPE MNEUMONIC ENTRY ID TIME STAMP DESCRIPTION
------- -------- --------- --------------- -------------
1 C01 MESSAGE 00000004 20:43:45.522449 SSL_ERROR
2 Job TCP341 Process 00020032 Thread 00000002 gsk_read_v3_record
3 Socket closed by 192.168.50.80.1360.
```

1. Standard IPCS header line, which includes the system name (C01), System SSL trace entry format (MESSAGE or DUMP), entry ID, time stamp, and record description.
2. System SSL header line with job name, process id, thread id and function name information.
3. System SSL detail information. The format of this area's content is determined according to the System SSL record description on line 1. Trace records may have 0 or more detail lines.

The standard IPCS header line MNEMONIC, ENTRY ID and DESCRIPTION combinations are as follows:

- **MESSAGE**
  - **00000001** SSL_ENTRY
    - Explanation: Entry into the function named in the following System SSL header line (i.e. line 2) occurred
  - **00000002** SSL_EXIT
    - Explanation: Exit from the function named in the following System SSL header line occurred
  - **00000004** SSL_ERROR
    - Explanation: Error was detected by the function named in following line 2 with error description in line 3
  - **00000008** SSL_INFO
    - Explanation: Information generated by the function named in following line 2 - for example, supplied parameters
- **DUMP**
  - **00000010** SSL_EBCDIC_DUMP
    - Explanation: Dump of buffer contents formatted in EBCDIC, by the function named in following line 2
  - **00000020** SSL_ASCII_DUMP
    - Explanation: Dump of buffer contents formatted in ASCII, by the function name in following line 2
Obtaining Diagnostic Information

The System SSL header line contains the Job name, Process ID (in hex), Thread ID (in hex), and the name of the System SSL function that created the trace entry. If the trace entry is output while in SRB mode, then the Thread ID is FFFFFFFF.

The format of the System SSL detail line is similar for record descriptions SSL_ENTRY, SSL_EXIT, SSL_ERROR and SSL_INFO.

1. The start of a trace record reporting that a function has been entered. Not all functions create a trace record. If a function creates an SSL_ENTRY record, then it also creates a corresponding SSL_EXIT record.

2. The System SSL header record describing the job, process, thread and function creating the record.

3. The detail for the SSL_ENTRY record. Not all trace records create a detail line. Trace records may have multiple detail lines.

4. The start of a trace record for function gsk_read_v3_record. The fact that an SSL_EXIT record has not been encountered for function gsk_secure_socket_read (the previous trace record), indicates that gsk_read_v3_record is invoked either by gsk_secure_socket_read or another function invoked by gsk_secure_socket_read.

7. The start of an error trace record created by gsk_read_v3_record.

10. An error trace record created by gsk_secure_socket_read. The error occurred because of the error detected in gsk_read_v3_record.

13. The start of the trace record created by gsk_secure_socket_read on exit. It corresponds with the trace entry record on Line 1.

15. The first detail line and reports the return code returned by gsk_secure_socket_read.

16. The second detail line for the trace record. It is an example of a trace record with multiple detail lines.

The format of the System SSL detail for record descriptions SSL_EBCDIC_DUMP and SSL_ASCII_DUMP is as follows:
Obtaining Diagnostic Information

1. Standard IPCS header line.
2. System SSL header line.
3. The first line of the System SSL detail area. It describes the contents that are dumped in the detail lines. In this example, the SERVER_HELLO message sent to the client is output in the detail lines.
4. The first line of the contents dump. Each dump line consists of offset, 16 bytes of data in hex, and the same 16 bytes of data output in either ASCII or EBCDIC enclosed in asterisks.

Capturing Component Trace Data without an External Writer

In the event that there is not an external writer, you can dump the GSKSRVR address space.

To use a dump:

- Dump the GSKSRVR address space with the command:
  
  \[\text{DUMP COMM=(title of dump)}\]

- Reply with:
  
  \[\text{R x,JOBNAM=(GSKSRVR),SDATA=(RGN,LSQA,ALLNUC,PSA,TRT,CSA,SQA),END}\]

- Issue
  
  \[\text{TRACE CT,OFF,COMP=GSKSRVR}\]

  to turn the trace off.

Note: You need to take the dump before you turn off the CTRACE.
Chapter 13. Messages and Codes

This topic contains information for the various forms of messages and codes you may encounter:

- SSL Function Return Codes
- Deprecated SSL Function Return Codes
- ASN.1 Status Codes (014CExxx)
- CMS Status Codes (03353xxx)
- SSL Started Task Messages (GSK01nnn)
- Utility Messages (GSK00nnn)

## SSL Function Return Codes

This topic describes the SSL function return codes.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1    | Handle is not valid.  
**Explanation:** The environment or SSL handle specified on a System SSL function call is not valid.  
**User response:** Call the `gsk_environment_open()` function to create an environment handle or the `gsk_secure_socket_open()` function to create an SSL handle. |
| 3    | An internal error has occurred.  
**Explanation:** The System SSL runtime library has detected an internal processing error.  
**User response:** Collect a System SSL trace containing the error and then contact your service representative. |
| 4    | Insufficient storage is available  
**Explanation:** The System SSL runtime library is unable to obtain storage for an internal control block.  
**User response:** Increase the storage available to the application and then retry the failing operation. |
| 5    | Handle is in the incorrect state.  
**Explanation:** The SSL handle is in the incorrect state for the requested operation.  
**User response:** Correct the application to request SSL functions in the proper sequence. |
| 6    | Key label is not found.  
**Explanation:** The requested key label is not found in the key database, SAF key ring or z/OS PKCS #11 token.  
**User response:** Specify a label that exists in the key database, SAF key ring or z/OS PKCS #11 token. |
| 7    | No certificates available.  
**Explanation:** The key database, SAF key ring or z/OS PKCS #11 token does not contain any certificates, or the SSL client application does not have a certificate available when authentication is requested by the server.  
**User response:** Check for available certificates and add the user certificate and any necessary certification authority certificates to the key database, SAF key ring or z/OS PKCS #11 token if necessary. If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST’ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure the latest changes are available. Specify a certificate for the client application to use. |
| 8    | Certificate validation error.  
**Explanation:** An error is detected while validating a certificate. This error can occur if a root CA certificate is not found in the key database, SAF keyring or z/OS PKCS #11 token or if the certificate is not marked as a trusted certificate or if the certificate requires an algorithm or key size that is non-FIPS while executing in FIPS mode.  
**User response:** Verify that the root CA certificate is in the key database, SAF keyring or z/OS PKCS #11 token and is marked as trusted. Check all certificates in the certification chain and verify that they are trusted and are not expired. If the error occurred while executing in FIPS mode, check that only FIPS algorithms and key sizes are used by the certificate. If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST’ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure the latest changes are available. Collect a System SSL trace containing the error and then contact your service representative if the problem persists. |
9 Cryptographic processing error.

Explanation: An error is detected by a cryptographic function. This error may also occur if key sizes that are non-FIPS are used during an SSL handshake while operating in FIPS mode.

User response: If the error occurred while executing in FIPS mode, check that only FIPS key sizes are used. Collect a System SSL trace containing the error and then contact your service representative.

10 ASN processing error.

Explanation: An error is detected while processing a certificate field.

User response: Collect a System SSL trace containing the error and then contact your service representative.

11 LDAP processing error.

Explanation: An error is detected while setting up the LDAP environment or retrieving an LDAP directory entry.

User response: Ensure that the LDAP server is running and that there are no network errors. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

12 An unexpected error has occurred.

Explanation: An unexpected error is detected by the System SSL runtime.

User response: Collect a System SSL trace containing the error and then contact your service representative.

103 Incorrect key database record format.

Explanation: The record format for a key database entry is not correct. This error can occur if the name of a request database is provided instead of the name of a key database.

User response: Ensure that the correct database name is used. Collect a System SSL trace containing a dump of the keyfile entry and then contact your service representative if the error persists.

106 Incorrect key database password.

Explanation: The System SSL runtime is unable to decrypt a key database entry. Either the supplied database password is incorrect or the database is damaged.

User response: Ensure that the correct key database password is used. Recreate the database if the error persists.

109 No certification authority certificates.

Explanation: The key database, SAF key ring or z/OS PKCS #11 token does not contain any valid certification authority certificates. The SSL runtime needs at least one CA or self-signed certificate in order to perform client authentication.

User response: Add the necessary certificates to the key database, SAF key ring or z/OS PKCS #11 token and ensure that existing certificates are valid, have not expired and are marked as trusted certificates. If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST’ed, issue the SETROPTS RACLST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure the latest changes are available.

201 No key database password supplied.

Explanation: A password stash file is specified but the SSL runtime is unable to read the password from the stash file.

User response: Verify that the password stash file exists and both the file and directory path are accessible to the application. Recreate the password stash file if the error persists.

202 Error detected while opening the certificate database.

Explanation: An error is detected while opening the key database, SAF key ring or z/OS PKCS #11 token. This error can occur if no name is supplied or the database, key ring or token does not exist.

User response: Verify that the key database, SAF key ring or z/OS PKCS #11 token exists and is accessible by the application. Collect a System SSL trace containing the error and then contact your service representative.
containing the error and then contact your service representative if the error persists.

**203 Unable to generate temporary key pair**

**Explanation:** An error is detected while generating a temporary key pair.

**User response:** Collect a System SSL trace containing the error and then contact your service representative.

**204 Key database password is expired.**

**Explanation:** The key database password is expired.

**User response:** Use the gskkyman command to assign a new password for the key database.

**302 Connection is active.**

**Explanation:** An SSL secure connection operation cannot be completed due to an active request for the connection.

**User response:** Retry the failing request when the currently active request has completed.

**401 Certificate is expired or is not valid yet.**

**Explanation:** The current time is either before the certificate start time or after the certificate end time.

**User response:** Obtain a new certificate if the certificate is expired or wait until the certificate becomes valid if it is not valid yet.

**402 No SSL cipher specifications.**

**Explanation:** The client and server cipher specifications do not contain at least one value in common. Client and server cipher specifications may be limited depending on which System SSL FMIDs are installed. See Table 1 on page 13 for more information. Server cipher specifications are dependent on the type of algorithms used by the server certificate (RSA, DSA and/or Diffie-Hellman), which may limit the options available during cipher negotiation. This error can also occur if no SSL protocols are enabled or if all of the enabled protocols have empty cipher specifications or if the TLS protocol is not enabled while executing in FIPS mode. This error can also occur if using the TLS V1.1 protocol and only the 40-bit export ciphers are specified.

**User response:** Ensure that the client and the server have at least one cipher specification in common.

**403 No certificate received from partner.**

**Explanation:** The required certificate was not received from the communication partner.

**User response:** Ensure that the remote application is sending the certificate. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

**405 Certificate format is not supported.**

**Explanation:** The certificate received from the communication partner is not supported by the current version of the System SSL runtime.

**User response:** Collect a System SSL trace containing a dump of the unsupported certificate and then contact your service representative.

**406 Error while reading or writing data.**

**Explanation:** An I/O error was reported while the System SSL runtime was reading or writing data.

**User response:** Ensure that there are no network errors. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

**407 Key label does not exist.**

**Explanation:** The supplied label or the default key is not found in the key database or the certificate is not trusted or the certificate uses algorithms or key sizes that are non-FIPS while executing in FIPS mode.

**User response:** Supply a valid label or define a default key in the key database or specify a label for a certificate that uses FIPS algorithms or key sizes if executing in FIPS mode.

For more information, see Chapter 4, “System SSL and FIPS 140-2,” on page 13.

**408 Key database password is not correct.**

**Explanation:** The System SSL runtime is unable to decrypt a keyfile entry. Either the supplied keyfile password is incorrect or the keyfile is damaged.

**User response:** Ensure that the correct keyfile password is used. Recreate the keyfile if the error persists.

**410 SSL message format is incorrect.**

**Explanation:** An incorrectly formatted SSL message is received from the communication partner.

**User response:** Collect a System SSL trace containing a dump of the SSL message and then contact your service representative.
411 Message authentication code is incorrect.

Explanation: The message authentication code (MAC) for a message is not correct. This indicates the message was modified during transmission.

User response: Collect a System SSL trace containing a dump of the message and then contact your service representative if the error persists.

412 SSL protocol or certificate type is not supported.

Explanation: The SSL handshake is not successful due to an unsupported protocol or certificate type. This error can occur if there is no enabled SSL protocol shared by both the client and the server. When executing in FIPS mode, specifying the SSL V2 or SSL V3 protocol is ignored.

User response: Ensure that the desired SSL protocol is enabled on both the client and the server. Collect a System SSL trace containing a dump of the failing handshake and then contact your service representative if the problem persists.

413 Certificate signature is incorrect.

Explanation: The certificate signature is not correct for a certificate received from the communication partner.

User response: Ensure that a valid certificate is being sent by the communication partner. Collect a System SSL trace containing a dump of the incorrect certificate and then contact your service representative if the error persists.

414 Certificate is not valid.

Explanation: Either the local certificate or the peer certificate is not valid.

User response: Ensure that a valid certificate is being sent by the communication partner. Collect a System SSL trace containing a dump of the incorrect certificate and then contact your service representative if the error persists.

415 SSL protocol violation.

Explanation: The communication partner has violated the SSL protocol by sending a message out of sequence or by omitting a required field from a message.

User response: Collect a System SSL trace and then contact your service representative.

416 Permission denied.

Explanation: The System SSL runtime is unable to access a file or system facility.

User response: Ensure the application is authorized to access the file or facility. Collect a System SSL trace and then contact your service representative if the error persists.

417 Self-signed certificate cannot be validated.

Explanation: A self-signed certificate cannot be validated because it is not in the key database, SAF key ring or z/OS PKCS #11 token.

User response: Add the self-signed certificate to the key database, SAF key ring or z/OS PKCS #11 token. If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLST'ed, issue the SETROPTS RACLST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure the latest changes are available.

420 Socket closed by remote partner.

Explanation: The remote partner closed the socket. This error will also be reported if the remote partner has sent a close notification alert.

User response: None.

421 SSL V2 cipher is not valid.

Explanation: The SSL V2 cipher is not valid.

User response: Specify a valid cipher.

422 SSL V3 cipher is not valid.

Explanation: The SSL V3 cipher is not valid.

User response: Specify a valid cipher.

427 LDAP is not available.

Explanation: The System SSL runtime is unable to access the LDAP server.

User response: Ensure that the LDAP server is running and that there are no network problems. Collect a System SSL trace and then contact your service representative if the error persists.

428 Key entry does not contain a private key.

Explanation: The key entry does not contain a private key or the private key is not usable. This error can also occur if the private key is stored in ICSF and ICSF services are not available, if the private key size is greater than the supported configuration limit or the
Application is executing in FIPS mode. Certificates that are meant to represent a server or client must be connected to a SAF keyring with a USAGE value of PERSONAL and either be owned by the userid of the application or be SITE certificates. This error can occur when using z/OS PKCS #11 tokens if the userid of the application does not have appropriate access to the CRYPTOZ class.

If executing in FIPS mode, ensure that the certificate being used does not have its private key stored in ICSF.

User response: Ensure that the ICSF started task has been started prior to the application if the private key is stored in ICSF. When using z/OS PKCS #11 tokens, ensure the userid has appropriate access to the CRYPTOZ class.

429 SSL V2 header is not valid.
Explanation: The received message does not start with a valid SSL V2 header. This error can occur if an SSL V3 client attempts to establish a secure connection with an SSL V2 server.
User response: Enable the SSL V2 protocol on the client and then retry the request.

431 Certificate is revoked.
Explanation: The certificate has been revoked by the certification authority.
User response: Obtain a new certificate.

432 Session renegotiation is not allowed.
Explanation: An attempt to renegotiate the session parameters for an active connection is rejected by the peer application.
User response: SSL processing continues using the current session parameters.

433 Key exceeds allowable export size.
Explanation: The key size used for an export cipher suite exceeds the allowable maximum size. For RSA and DSA keys, the maximum export key size is 512 bits. If the certificate key is larger than 512 bits, the SSL runtime will use a temporary 512-bit key for the connection.
User response: Collect a System SSL trace and then contact your service representative.

434 Certificate key is not compatible with cipher suite.
Explanation: The certificate key is not compatible with the negotiated cipher suite. The negotiated cipher suite is dependent on the type of algorithms used by the server certificate (RSA, DSA and/or Diffie-Hellman) and those available for the client to use. This error can also occur if the client certificate uses an algorithm that is incompatible with the server certificate.
User response: Specify a certificate with the appropriate key type.

435 Certification authority is unknown.
Explanation: The key database does not contain a certificate for the certification authority.
User response: Obtain the certificate for the certification authority and add it to the key database. When using a SAF key ring, the CA certificate must be TRUSTed. If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLISTed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure the latest changes are available.

436 Certificate revocation list cannot be found.
Explanation: A certificate revocation list (CRL) cannot be found in the specified LDAP server.
User response: Contact the certification authority and obtain the required CRL.

437 Connection closed.
Explanation: For gsk_secure_socket_read(), a close notification has been received from the peer application. For gsk_secure_socket_write(), a close notification has been sent to the peer application. A close notification is sent when the gsk_secure_socket_shutdown() routine is called or when a close notification is received from the peer application. Additional data may not be sent by the application after the close notification has been sent to the peer application.
User response: None

438 Internal error reported by remote partner.
Explanation: The peer application has detected an internal error while performing an SSL operation and has sent an alert to close the secure connection.
User response: Check the error log for the remote application to determine the nature of the processing error.

439 Unknown alert received from remote partner.
Explanation: The peer application has sent an alert message which is not recognized by the System SSL runtime.
**User response:** Collect a System SSL trace and then contact your service representative.

<table>
<thead>
<tr>
<th>440 Incorrect key usage.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong> The key usage certificate extension does not permit the requested key operation. This error can occur if the key usage extension of a client or server certificate (if any) does not allow the appropriate key usage.</td>
</tr>
<tr>
<td>- RSA server certificates using 40-bit export ciphers with a public key size greater than 512 bits must allow digital signature.</td>
</tr>
<tr>
<td>- RSA or DSA server certificates using fixed Diffie-Hellman key exchange must allow key agreement.</td>
</tr>
<tr>
<td>- Other RSA server certificates must allow key encipherment.</td>
</tr>
<tr>
<td>- DSA server certificates using ephemeral Diffie-Hellman key exchange must allow digital signature.</td>
</tr>
<tr>
<td>- Client certificates using fixed Diffie-Hellman key exchange must allow digital signature.</td>
</tr>
<tr>
<td>- Otherwise client certificates must allow digital signature.</td>
</tr>
<tr>
<td><strong>User response:</strong> Specify a certificate with the appropriate key usage.</td>
</tr>
</tbody>
</table>

If the gskkyman utility was used to create either the client or server end-entity certificate, ensure that the appropriate option was selected from the Certificate Type menu to create a user or server certificate.

<table>
<thead>
<tr>
<th>442 Multiple certificates exist for label.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong> Access of certificate/key via label could not be resolved because multiple certificates/keys exist with the label.</td>
</tr>
<tr>
<td><strong>User response:</strong> Correct certificate/key store so that label specifies a unique record.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>443 Multiple keys are marked as the default.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong> Access of key via default status could not be resolved because multiple keys are marked as the default key.</td>
</tr>
<tr>
<td><strong>User response:</strong> Correct the certificate/key store so that only one key is marked as the default key.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>444 Error encountered generating random bytes.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong> The SSL/TLS handshake has encountered an error while generating random bytes.</td>
</tr>
<tr>
<td><strong>User response:</strong> Re-try the secure connection. Contact your service representative if the error persists.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>445 Key database is not a FIPS mode database.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong> While executing in FIPS mode, an attempt was made to open a key database that does not meet FIPS criteria.</td>
</tr>
<tr>
<td><strong>User response:</strong> Specify a key database that meets FIPS criteria if running in FIPS mode.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>446 TLS extension mismatch has been encountered.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong> The TLS client has received a message from the TLS server containing a TLS extension that was not requested. The TLS server must only respond to an extension that was sent by the TLS client.</td>
</tr>
<tr>
<td><strong>User response:</strong> Ensure that the TLS server is operating correctly. If the problem persists, collect a System SSL trace and contact your service representative.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>447 Required TLS extension has been rejected.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong> The TLS server or client has encountered a communicating partner that does not support a TLS extension that has been defined as required.</td>
</tr>
<tr>
<td><strong>User response:</strong> Ensure that the TLS extension data is correctly defined, and that both the TLS server and client support the required extension. If the problem persists collect a System SSL trace and contact your service representative.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>448 Requested server name is not recognized.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong> The TLS server has been unable to match the server names supplied in a &quot;Server Name Indication&quot; type TLS extension, and either the TLS server or TLS client has determined this scenario to be fatal.</td>
</tr>
<tr>
<td><strong>User response:</strong> Ensure that the TLS extension data is correct for both the TLS server and client.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>449 Unsupported fragment length was received.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong> The TLS server has received a Maximum Fragment Length TLS extension request from the TLS client that specifies an unsupported maximum fragment length. Supported maximum fragment lengths are 512 bytes, 1024 bytes, 2048 bytes and 4096 bytes.</td>
</tr>
<tr>
<td><strong>User response:</strong> Ensure that the TLS extension data is correct for the TLS server and the communicating partner. If the problem persists collect a System SSL trace and contact your service representative.</td>
</tr>
</tbody>
</table>
450 TLS extension length field is not valid.
Explanation: The TLS client or server has received a message containing a TLS extension that was not correctly formed. The TLS extension data contains a length field that has an incorrect value.
User response: Ensure that the TLS extension data is correct for both the TLS server and client. If the problem persists collect a System SSL trace and contact your service representative.

501 Buffer size is not valid.
Explanation: The socket buffer or buffer size is not valid.
User response: Specify a valid buffer and buffer size.

502 Socket request would block.
Explanation: The socket is in non-blocking mode and the socket request returned the EWOULDBLOCK error.
User response: Retry the gsk_secure_socket_read() or gsk_secure_socket_write() request when the socket is ready to send or receive data.

503 Socket read request would block.
Explanation: A socket read request issued as part of an SSL handshake returned the EWOULDBLOCK error.
User response: Retry the failing request when the socket is ready to receive data.

504 Socket write request would block.
Explanation: A socket write request issued as part of an SSL handshake return the EWOULDBLOCK error.
User response: Retry the failing request when the socket is ready to send data.

505 Record overflow.
Explanation: An SSL protocol record has a plain text record length greater than 16384 or an encrypted text record length greater than 18432.
User response: Ensure that data is not being corrupted during transmission. Obtain a System SSL trace containing a dump of the failing record and contact your service representative if the error persists.

601 Protocol is not SSL V3, TLS V1.0 or TLS V1.1.
Explanation: The requested function requires the SSL V3, TLS V1.0 or TLS V1.1 protocol.
User response: Ensure that the correct protocol is in use before issuing the request.
User response: Ensure that the TLS extension data is correctly defined. If the problem persists collect a System SSL trace and contact your service representative.
## Deprecated SSL Function Return Codes

This topic describes the deprecated SSL function return codes.

<table>
<thead>
<tr>
<th>Code</th>
<th>Error Description</th>
<th>Explanation</th>
<th>User Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Error detected while reading certificate database</td>
<td>An error is detected while reading the key database or retrieving entries on the SAF key ring or z/OS PKCS #11 token.</td>
<td>Collect a System SSL trace containing the error and then contact your service representative.</td>
</tr>
<tr>
<td>2</td>
<td>Error detected while opening the certificate database.</td>
<td>An error is detected while opening the key database, SAF key ring or z/OS PKCS #11 token. This error can occur if no name is supplied or the database, key ring or token does not exist.</td>
<td>Verify that the key database, SAF key ring or z/OS PKCS #11 token exists and is accessible by the application. Collect a System SSL trace containing the error and then contact your service representative if the error persists.</td>
</tr>
<tr>
<td>3</td>
<td>Incorrect key database record format.</td>
<td>The record format for a key database entry is not correct. This error can occur if the name of a request database is provided instead of the name of a key database.</td>
<td>Ensure that the correct database name is used. Collect a System SSL trace containing a dump of the keyfile entry and then contact your service representative if the error persists.</td>
</tr>
<tr>
<td>4</td>
<td>Key database password is not correct.</td>
<td>The System SSL runtime is unable to decrypt a keyfile entry. Either the supplied keyfile password is incorrect or the keyfile is damaged.</td>
<td>Ensure that the correct keyfile password is used and both the file and directory path are accessible to the application</td>
</tr>
<tr>
<td>9</td>
<td>Key label does not exist.</td>
<td>The supplied label or the default key is not found in the key database or the certificate is not trusted.</td>
<td>Supply a valid label or define a default key in the key database.</td>
</tr>
<tr>
<td>12</td>
<td>Key label is not found.</td>
<td>The requested key label is not found in the key database, SAF key ring or z/OS PKCS #11 token.</td>
<td>Specify a label that exists in the key database, SAF key ring or z/OS PKCS #11 token.</td>
</tr>
<tr>
<td>13</td>
<td>Duplicate subject names.</td>
<td>The key database, SAF key ring or z/OS PKCS #11 token contains multiple certificates with the same subject name as the DN specified in the gsk_secure_soc_init() initialization data.</td>
<td>Either remove the duplicate certificates or specify a label instead of a DN in the gsk_secure_soc_init() initialization data.</td>
</tr>
<tr>
<td>16</td>
<td>Incorrect key database password.</td>
<td>The System SSL runtime is unable to decrypt a key database entry. Either the supplied database password is incorrect or the database is damaged.</td>
<td>Ensure that the correct key database password is used. Recreate the database if the error persists.</td>
</tr>
<tr>
<td>17</td>
<td>Key database password is expired.</td>
<td>The key database password is expired.</td>
<td>Use the gskkyman command to assign a new password for the key database.</td>
</tr>
<tr>
<td>18</td>
<td>No certification authority certificates.</td>
<td>The key database, SAF key ring or z/OS PKCS #11 token does not contain any valid certification authority certificates. The SSL runtime needs at least one CA or self-signed certificate in order to perform client authentication.</td>
<td>Add the necessary certificates to the key database, SAF key ring or z/OS PKCS #11 token and ensure that existing certificates are valid and have not expired. If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST’ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure the latest changes are available.</td>
</tr>
<tr>
<td>19</td>
<td>No certificates available.</td>
<td>The key database, SAF key ring or z/OS PKCS #11 token does not contain any certificates, or the SSL client application does not have a certificate available when authentication is requested by the server.</td>
<td></td>
</tr>
</tbody>
</table>
User response: Check for available certificates and add the user certificate and any necessary certification authority certificates to the key database, SAF key ring or z/OS PKCS #11 token if necessary. Specify a certificate for the client application to use. If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLST'ed, issue the SETROPTS RACLST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure the latest changes are available.

70 Application is not APF-authorized.
Explanation: The gsk_srb_initialize() routine is called but the program is not APF authorized. SRB mode cannot be used by unauthorized applications.
User response: Contact your system programmer to get your application authorized.

71 Unable to establish ESTAE environment.
Explanation: The gsk_srb_initialize() routine is unable to establish the ESTAE error recovery environment.
User response: Contact your service representative.

72 Unable to create service thread.
Explanation: The gsk_srb_initialize() routine is unable to create a thread to handle SRB processing.
User response: Ensure that POSIX thread support is available to the application environment. Contact your service representative if the error persists.

100 Initialization parameter is not valid
Explanation: An initialization parameter for gsk_initialize() or gsk_secure_soc_init() is not valid.
User response: Ensure that all of the parameters are correct. Contact your service representative if the error persists.

102 Security type is not valid
Explanation: The security type specified in the initialization data for the gsk_initialize() routine is not valid.
User response: Specify a valid security type for the sec_types parameter.

103 SSL V2 session timeout is not valid.
Explanation: The SSL V2 session timeout specified in the initialization data for the gsk_initialize() routine is not valid.
User response: Specify a valid SSL V2 session timeout value.

104 SSL V3 session timeout is not valid.
Explanation: The SSL V3 session timeout specified in the initialization data for the gsk_initialize() routine is not valid.
User response: Specify a valid SSL V3 session timeout value.

-1 No SSL cipher specifications.
Explanation: The client and server cipher specifications do not contain at least one value in common. Client and server cipher specification may be limited depending on which System SSL FMIDs are installed. See Table 1 on page 1 for more information. Server cipher specifications are dependent on the type of algorithms used by the server certificate (RSA, DSA and/or Diffie-Hellman), which may limit the options available during cipher negotiation. This error can also occur if no SSL protocols are enabled or if all of the enabled protocols have empty cipher specifications.
User response: Ensure that the client and the server have at least one cipher specification in common.

-2 No certificate received from partner.
Explanation: The required certificate was not received from the communication partner.
User response: Ensure that the remote application is sending the certificate. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

-3 Certificate key is not compatible with cipher suite.
Explanation: The certificate key is not compatible with the negotiated cipher suite. The negotiated cipher suite is dependent on the type of algorithms used by the server certificate (RSA, DSA and/or Diffie-Hellman) and those available for the client to use. This error can occur if the client certificate uses an algorithm that is incompatible with the server certificate.
User response: Specify a certificate with the appropriate key type.

-5 SSL V2 header is not valid.
Explanation: The received message does not start with a valid SSL V2 header. This error can occur if an SSL V3 client attempts to establish a secure connection with an SSL V2 server.
User response: Enable the SSL V2 protocol on the client and then retry the request.
-6 Certificate format is not supported.
**Explanation:** The certificate received from the communication partner is not supported by the current version of the System SSL runtime.
**User response:** Collect a System SSL trace containing a dump containing the unsupported certificate and then contact your service representative.

-7 Session renegotiation is not allowed.
**Explanation:** An attempt to renegotiate the session parameters for an active connection is rejected by the peer application.
**User response:** SSL processing continues using the current session parameters.

-9 Certificate is revoked.
**Explanation:** The certificate has been revoked by the certification authority.
**User response:** Obtain a new certificate.

-10 Error while reading or writing data.
**Explanation:** An I/O error was reported while the System SSL runtime was reading or writing data.
**User response:** Ensure that there are no network errors. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

-11 SSL message format is incorrect.
**Explanation:** An incorrectly formatted SSL message is received from the communication partner.
**User response:** Collect a System SSL trace containing a dump of the SSL message and then contact your service representative.

-12 Message authentication code is incorrect.
**Explanation:** The message authentication code (MAC) for a message is not correct. This indicates the message was modified during transmission.
**User response:** Collect a System SSL trace containing a dump of the message and then contact your service representative if the error persists.

-13 SSL protocol or certificate type is not supported.
**Explanation:** The SSL handshake is not successful due to an unsupported protocol or certificate type. This error can occur if there is no enabled SSL protocol shared by both the client and the server.
**User response:** Ensure that the desired SSL protocol is enabled on both the client and the server. Collect a System SSL trace containing a dump of the failing handshake and then contact your service representative if the problem persists.

-14 Certificate signature is incorrect
**Explanation:** The certificate signature is not correct for a certificate received from the communication partner.
**User response:** Ensure that a valid certificate is being sent by the communication partner. Collect a System SSL trace containing a dump of the incorrect certificate and then contact your service representative if the error persists.

-15 Certificate is not valid
**Explanation:** Either the local certificate or the peer certificate is not valid. In order for a certificate to be valid, the complete certificate chain must be present in the key database file, SAF key ring or z/OS PKCS #11 token. Verify that the certificate in the certificate chain is marked trusted.
**User response:** Ensure that a valid certificate is being sent by the communication partner. Collect a System SSL trace containing a dump of the incorrect certificate and then contact your service representative if the error persists. If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST’ed, issue the SETROPTS RAACLST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure the latest changes are available.

-16 SSL protocol violation.
**Explanation:** The communication partner has violated the SSL protocol by sending a message out of sequence or by omitting a required field from a message.
**User response:** Collect a System SSL trace and then contact your service representative.

-17 Permission denied.
**Explanation:** The System SSL runtime is unable to access a file or system facility.
**User response:** Ensure the application is authorized to access the file or facility. Collect a System SSL trace and then contact your service representative if the error persists.

-18 Self-signed certificate cannot be validated.
**Explanation:** A self-signed certificate cannot be validated because it is not in the key database, SAF key ring or z/OS PKCS #11 token.
**User response:** Add the self-signed certificate to the key database, SAF key ring or z/OS PKCS #11 token. If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure the latest changes are available.

-19 Certification authority is unknown

**Explanation:** The key database does not contain a certificate for the certification authority.

**User response:** Obtain the certificate for the certification authority and add it to the key database. When using a SAF key ring, the CA certificate must be TRUSTed. If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure the latest changes are available.

-20 Insufficient storage is available.

**Explanation:** The System SSL runtime library is unable to obtain storage for an internal control block.

**User response:** Increase the storage available to the application and then retry the failing operation.

-21 Handle is in the incorrect state.

**Explanation:** The SSL connection handle is in the incorrect state for the requested operation.

**User response:** Correct the application to request SSL functions in the proper sequence.

-22 Socket closed by remote partner.

**Explanation:** The remote partner closed the socket.

**User response:** None.

-25 Certificate is expired or is not valid yet.

**Explanation:** The current time is either before the certificate start time or after the certificate end time.

**User response:** Obtain a new certificate if the certificate is expired or wait until the certificate becomes valid if it is not valid yet.

-26 Key exceeds allowable export size.

**Explanation:** The key size used for an export cipher suite exceeds the allowable maximum size. For RSA and DSA keys, the maximum export key size is 512 bits. If the certificate key is larger than 512 bits, the SSL runtime will use a temporary 512-bit key for the connection.

**User response:** Collect a System SSL trace and then contact your service representative.

-27 Key entry does not contain a private key.

**Explanation:** The key entry does not contain a private key or the private key is not usable. This error can also occur if the private key is stored in ICSF and ICSF services are not available, if the private key size is greater than the supported configuration limit or the application is executing in FIPS mode. Certificates that are meant to represent a server or client must be connected to a SAF key ring with a USAGE value of PERSONAL and either be owned by the userid of the application or be SITE certificates. This error can occur when using z/OS PKCS #11 tokens if the userid of the application does not have appropriate access to the CRYPTOZ class.

**User response:** Specify a key entry containing a private key value. Ensure that the ICSF started task is running if the private key is stored in ICSF. When using z/OS PKCS #11 tokens ensure the userid has appropriate access to the CRYPTOZ class.

-28 Function parameter is not valid.

**Explanation:** A parameter specified on an SSL function call is not valid.

**User response:** Ensure that the parameters on the failing function call are correct. Contact your service representative if the error persists.

-30 Socket request would block.

**Explanation:** The socket is in non-blocking mode and the socket request returned the EWOULDBLOCK error.

**User response:** Retry the gsk_secure_soc_read() or gsk_secure_soc_write() request when the socket is ready to send or receive data.

-34 Certificate revocation list cannot be found.

**Explanation:** A certificate revocation list (CRL) cannot be found in the specified LDAP server.

**User response:** Contact the certification authority and obtain the required CRL.

-35 Certificate validation error.

**Explanation:** An error is detected while validating a certificate. This error can occur if a root CA certificate is not found in the key database, SAF keyring or z/OS PKCS #11 token or if the certificate is not marked as a trusted certificate or if the certificate requires an
User response: Verify that the root CA certificate is in the key database, SAF key ring or z/OS PKCS #11 token and is marked as trusted. Check all certificates in the certification chain and verify that they are trusted and are not expired. Collect a System SSL trace containing the error and then contact your service representative if the problem persists. If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure the latest changes are available.

User response: If the error occurred while executing in FIPS mode, check that only FIPS key sizes are used. Collect a System SSL trace containing the error and then contact your service representative.

For more information, see Chapter 4, “System SSL and FIPS 140-2,” on page 13.

### -36 Cryptographic processing error.

**Explanation:** An error is detected by a cryptographic function. This error may also occur if key sizes that are non-FIPS are used during an SSL handshake while operating in FIPS mode.

**User response:** If the error occurred while executing in FIPS mode, check that only FIPS key sizes are used. Collect a System SSL trace containing the error and then contact your service representative.

For more information, see Chapter 4, “System SSL and FIPS 140-2,” on page 13.

### -37 ASN processing error.

**Explanation:** An error is detected while processing a certificate field.

**User response:** Collect a System SSL trace containing the error and then contact your service representative.

### -38 LDAP processing error.

**Explanation:** An error is detected while setting up the LDAP environment or retrieving an LDAP directory entry.

**User response:** Ensure that the LDAP server is running and that there are no network errors. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

### -39 LDAP is not available.

**Explanation:** The System SSL runtime is unable to access the LDAP server.

**User response:** Ensure that the LDAP server is running and that there are no network problems. Collect a System SSL trace and then contact your service representative if the error persists.

### -40 SSL V2 cipher is not valid.

**Explanation:** The SSL V2 cipher is not valid.

**User response:** Specify a valid cipher.

### -41 SSL V3 cipher is not valid.

**Explanation:** The SSL V3 cipher is not valid.

**User response:** Specify a valid cipher.

### -42 Bad handshake specification.

**Explanation:** The handshake specification for the gsk_secure_soc_init() routine is not valid.

**User response:** Specify a valid value for the hs_type field in the gsk_secure_soc_init() initialization data.

### -43 No read function.

**Explanation:** No read function is provided for the gsk_secure_soc_init() routine.

**User response:** Specify a read function for the skread field in the gsk_secure_soc_init() initialization data.

### -44 No write function.

**Explanation:** No write function is provided for the gsk_secure_soc_init() routine.

**User response:** Specify a write function for the skwrite field in the gsk_secure_soc_init() initialization data.

### -46 Socket write request would block.

**Explanation:** A socket write request issued as part of an SSL handshake return the EWOULDBLOCK error.

**User response:** Retry the failing request when the socket is ready to send data.

### -47 Connection is active.

**Explanation:** An SSL secure connection operation cannot be completed due to an active request for the connection.

**User response:** Retry the failing request when the currently active request has completed.

### -48 Connection closed.

**Explanation:** For gsk_secure_soc_read(), a close notification has been received from the peer application. For gsk_secure_soc_write(), a close notification has been sent to the peer application. A close notification is sent when a close notification is received from the peer application. Additional data may not be sent by the application after the close notification has been sent to the peer application.
User response: None.

-51 Protocol is not SSL V3 or TLS V1.0.
Explanation: The requested function requires the SSL V3 or TLS V1.0 protocol.
User response: Ensure that the correct protocol is in use before issuing the request.

-53 Internal error reported by remote partner.
Explanation: The peer application has detected an internal error while performing an SSL operation and has sent an alert to close the secure connection.
User response: Check the error log for the remote application to determine the nature of the processing error.

-54 Unknown alert received from remote partner.
Explanation: The peer application has sent an alert message which is not recognized by the System SSL runtime.
User response: Collect a System SSL trace and then contact your service representative.

-55 Incorrect key usage.
Explanation: The key usage certificate extension does not permit the requested key operation. This error can occur if the key usage extension of a client or server certificate (if any) does not allow the appropriate key usage.
- RSA server certificates using 40-bit export ciphers with a public key size greater than 512 bits must allow digital signature.
- RSA or DSA server certificates using fixed Diffie-Hellman key exchange must allow key agreement.
- Other RSA server certificates must allow key encipherment.
- DSA server certificates using ephemeral Diffie-Hellman key exchange must allow digital signature.
- Client certificates using Diffie-Hellman key exchange must allow key agreement.
- Otherwise client certificates must allow digital signature.
User response: Specify a certificate with the appropriate key usage.
If the gskkyman utility was used to create either the client or server end-entity certificate, ensure that the appropriate option was selected from the Certificate Type menu to create a user or server certificate.

-56 Multiple certificates exist for label.
Explanation: Access of certificate/key via label could not be resolved because multiple certificates/keys exist with the label.
User response: Correct certificate/key store so that label specifies a unique record.

-57 Multiple keys are marked as the default.
Explanation: Access of key via default status could not be resolved because multiple keys are marked as the default key.
User response: Correct certificate/key store so that only one key is marked as the default key.

-70 SRB processing is not initialized.
Explanation: The gsk_srb_initialize() routine has not been called to initialize the SRB support.
User response: Call gsk_srb_initialize() before making any calls to GSKSRBRD or GSKSRBWT.

-71 SRB lock timeout.
Explanation: The GSKSRBRD or GSKSRBWT routine is unable to obtain the lock for the SRB control area.
User response: Ensure that the SRB processing threads are not suspended (for example, a synchronous dump will suspend thread execution while the dump is processed). Contact your service representative if the error persists.

-72 SRB suspend failed.
Explanation: The GSKSRBRD or GSKSRBWT routine is unable to suspend execution while waiting for the completion of the read or write request.
User response: Contact your service representative.

-73 Unknown SRB service request.
Explanation: The SRB service task does not recognize the function request.
User response: Contact your service representative.

-99 An unexpected error has occurred.
Explanation: An unexpected error is detected by the System SSL runtime.
User response: Collect a System SSL trace containing the error and then contact your service representative.
-100 Buffer size is not valid.
   **Explanation:** The socket buffer or buffer size is not valid.
   **User response:** Specify a valid buffer and buffer size.

-101 Handle is not valid.
   **Explanation:** The SSL connection handle specified on a System SSL function call is not valid.
   **User response:** Call the `gsk_secure_soc_init()` function to create an SSL connection handle.

-104 Error encountered generating random bytes.
   **Explanation:** The SSL/TLS handshake has encountered an error while generating random bytes.
   **User response:** Re-try the secure connection. Contact your service representative if the error persists.

-105 Key database is not a FIPS mode database.
   **Explanation:** While executing in FIPS mode, an attempt was made to open a key database that does not meet FIPS criteria.
   **User response:** Specify a key database that meets FIPS criteria if running in FIPS mode.
ASN.1 Status Codes (014CExxx)

This topic describes the ASN.1 status codes.

014CE001  No more data.
Explanation:  The end of an ASN.1 encoded stream is reached prematurely. This error can occur if an encoded stream is truncated.
User response:  Verify that the encoded certificate is not modified. Contact your service representative if the error persists.

014CE002  Data value overflow.
Explanation:  A decoded data value is too large to be represented as the specified data type.
User response:  Contact your service representative.

014CE003  Length value is not valid.
Explanation:  The length of an encoded item is not valid. This error can occur if an encoded stream is truncated.
User response:  Verify that the encoded certificate is not modified. Contact your service representative if the error persists.

014CE004  Data encoding is not valid.
Explanation:  The encoded data violates the ASN.1 encoding rules.
User response:  Contact your service representative.

014CE005  Parameter is not valid
Explanation:  An application parameter is not valid.
User response:  Correct the application to specify valid parameters for the failing function call. Contact your service representative if the error persists.

014CE006  Insufficient memory is available.
Explanation:  There is not enough memory available to allocate a required control block or data element.
User response:  Increase the memory available to the application and then retry the request. Contact your service representative if the error persists.

014CE007  Indefinite-length encoding is not allowed
Explanation:  An indefinite-length encoding is encountered for a data element that requires a length value.
User response:  Contact your service representative.

014CE008  Data element must be an ASN.1 primitive.
Explanation:  A constructed element is encountered instead of an ASN.1 primitive.
User response:  Contact your service representative.

014CE009  Data element must be constructed.
Explanation:  An ASN.1 primitive is encountered instead of a constructed element.
User response:  Contact your service representative.

014CE00A  Data value is not present
Explanation:  An ASN.1 element has no value and does not have a default value.
User response:  Contact your service representative.

014CE00B  Indefinite-length encoding is not supported.
Explanation:  Indefinite-length encoding is not support for the current structure. An X.509 certificate is encoded using ASN.1 DER (Distinguished Encoding Rules) which does not allow the use of indefinite-length encodings.
User response:  Contact your service representative.

014CE00C  Unused bit count is not valid
Explanation:  The unused bit count for a bit string must be between 0 and 7.
User response:  Correct the application if this error occurs while encoding a bit string. Contact your service representative if this error occurs while decoding a bit string.

014CE00D  Unused bit count is not valid for a segmented bit string.
Explanation:  The unused bit count must be zero for each segment other than the final segment of a bit string.
User response:  Contact your service representative.

014CE00E  Data type is not correct.
Explanation:  An unexpected data type is encountered while decoding a data element.
User response:  Contact your service representative.
014CE00F  Excess data found at end of data element
Explanation: There is unprocessed encoded data after decoding a data element.
User response: Contact your service representative.

014CE010  Required data element is missing.
Explanation: A required data element is not found when decoding an encoded structure.
User response: Contact your service representative.

014CE011  Selection is not within the valid range.
Explanation: The selection for an ASN.1 element is not within the valid range for that element.
User response: Contact your service representative.

014CE012  No selection found
Explanation: No selection found for an ASN.1 element.
User response: Contact your service representative.

014CE013  Syntax already set.
Explanation: The decoding syntax has already been set for an ASN.1 element.
User response: Contact your service representative.

014CE014  Character string cannot be converted.
Explanation: A character string cannot be converted to the target code page. This error can occur when a character string contains characters which cannot be represented in the target code page.
User response: Ensure that the character string uses characters which are valid for the target code page. Contact your service representative if the error persists.

014CE015  Codeset is not allowed
Explanation: The requested codeset is not valid for the current data element.
User response: Contact your service representative.

014CE016  Attribute value is not valid.
Explanation: An attribute value is not valid.
User response: Contact your service representative.

014CE017  Attribute value separator is missing.
Explanation: An X.500 attribute value separator is missing.
User response: Ensure that the name string is correctly formed. Each attribute consists of an attribute type and an attribute value separated by an equal sign. Contact your service representative if the error persists.

014CE018  Attribute value is missing.
Explanation: An X.500 attribute value is missing.
User response: Correct the application to specify an attribute for each relative distinguished name.

014CE019  Object identifier syntax error
Explanation: The syntax of an object identifier is not valid. The object identifier consists of one or more decimal numbers separated by periods.
User response: Correct the application to specify a valid object identifier.

014CE01A  PKCS #12 version is not correct.
Explanation: The PKCS #12 version is not correct.
User response: Contact your service representative.

014CE01B  Interval is not valid.
Explanation: The certificate interval is not valid.
User response: Contact your service representative.

014CE01C  Object identifier element count is not valid.
Explanation: An object identifier must have at least three elements.
User response: Correct the application to provide a valid object identifier.

014CE01D  Incorrect value for the first object identifier element.
Explanation: The first element of an object identifier must be 0, 1, or 2.
User response: Correct the application to provide a valid object identifier.

014CE01E  Incorrect value for the second object identifier element.
Explanation: The second element of an object identifier must be between 0 and 39 if the first element is 0 or 1.
**User response:** Correct the application to provide a valid object identifier.

---

**014CE01F Unknown attribute identifier.**

**Explanation:** An unrecognized attribute identifier is encountered while decoding a certificate extension or an X.509 name. As a result, the attribute value cannot be decoded.

**User response:** Ensure that the name string is correctly formed. Each attribute consists of an attribute type and an attribute value separated by an equal sign. Contact your service representative if the error persists.

---

**014CE020 Unknown critical certificate extension.**

**Explanation:** The X.509 certificate contains a critical extension that is not recognized by the System SSL runtime. The certificate cannot be processed.

**User response:** Obtain a new certificate without the unknown critical certificate extension.

---

**014CE021 X.500 name syntax error.**

**Explanation:** The syntax of an X.500 distinguished name is not valid. Refer to RFC 2253 (String Representation of Distinguished Names) for more information on the format of a distinguished name.

**User response:** Correct the application to specify a valid distinguished name.

---

**014CE022 Version is not supported.**

**Explanation:** The version number in a certificate, certificate request, or certificate revocation list is not supported by the current level of System SSL.

**User response:** Obtain a new certificate, certificate request, or certificate revocation list with a supported version number.
CMS Status Codes (03353xxx)

This topic describes some CMS status codes.

03353001  Insufficient memory is available.

**Explanation:** There is not enough memory available to allocate a required control block or data element.

**User response:** Increase the memory available to the application and then retry the request. Contact your service representative if the error persists.

03353002  Certificate extension is not supported.

**Explanation:** An X.509 certificate extension is either not supported by the current level of the System SSL runtime or is not supported by the certificate version. The certificate extension is not processed. If the extension is marked as a critical extension, the X.509 certificate cannot be processed.

**User response:** Upgrade the System SSL runtime if a later software level supports the certificate extension.

03353003  Cryptographic algorithm is not supported.

**Explanation:** An X.509 cryptographic algorithm is not supported by the current level of the System SSL runtime. This error can also occur if the current operation does not support the specified cryptographic algorithm. When running in FIPS mode, this error may occur if an attempt is made to use an algorithm not supported in FIPS mode.

**User response:** Ensure that the cryptographic algorithm is supported for the requested operation or that it is supported if executing in FIPS mode. Upgrade the System SSL runtime if a later software level supports the cryptographic algorithm.

03353004  Signature is not correct

**Explanation:** The signature is incorrect for an X.509 certificate or certificate revocation list. This usually means the certificate has been modified since it was signed by the issuing Certificate Authority.

**User response:** Verify that the certificate has not been modified. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

03353005  Cryptographic request failed.

**Explanation:** A cryptographic request failed with an unexpected error. This error can occur if the hardware cryptographic support becomes unavailable after the application has been initialized.

**User response:** Collect a System SSL trace containing the error and then contact your service representative.

03353006  Input/Output request canceled.

**Explanation:** An input/output operation is canceled by the user. This can occur if the user cancels a terminal read request by pressing an attention key or by pressing the enter key without entering any data.

**User response:** None

03353007  Input/Output request failed.

**Explanation:** An input/output operation fails.

**User response:** Verify that the file or keyring can be accessed and is not damaged. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

03353008  Verification password does not match.

**Explanation:** The user is prompted to verify the password by entering it a second time. The user did not enter the same password both times.

**User response:** Enter the same password when prompted.

03353009  File or keyring not found

**Explanation:** A file or keyring cannot be opened because it is not found.

**User response:** Verify that the correct name is specified. Contact your service representative if the error persists.

0335300A  Database is not valid.

**Explanation:** The key database or the request database is not valid. This error can occur if the wrong database password is used when opening the database or if the database format is not supported by the current level of the System SSL runtime.

**User response:** Verify that the database has not been modified or truncated. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

0335300B  Message not found.

**Explanation:** The System SSL runtime is unable to locate a message in the message catalog.

**User response:** Verify that the message catalog can be accessed by the application and can be located...
using the NLSPATH environment variable. Contact your service representative if the error persists.

0335300C Handle is not valid.
Explanation: The handle passed to the System SSL runtime is not valid. This error can occur if the handle has been closed or is not the proper type for the requested function.
User response: Pass a valid handle to the System SSL routine.

0335300D Record deleted.
Explanation: The requested record is deleted.
User response: None

0335300E Record not found.
Explanation: The requested record is not found.
User response: None

0335300F Incorrect database type
Explanation: The database does not support the requested operation. This error can occur if the database type is not valid. It can also occur if an attempt is made to add a request record to a key database or a key record to a request database.
User response: Specify an operation supported by the database.

03353010 Database is not open for update.
Explanation: A request to modify the key or request database cannot be completed because update mode was not requested when the database was opened or an update was requested on a FIPS mode database while in non-FIPS mode.
User response: Request update mode when opening a database for modification.

03353011 Mutex request failed.
Explanation: A mutex operation failed.
User response: Contact your service representative.

03353012 Backup file already exists.
Explanation: Before updating a database file, the System SSL runtime creates a backup file with the same name with ".new" appended to the name. This file is then deleted after the database file has been rewritten. The file is not deleted if an error occurs while rewriting the database file.
User response: Correct the problem that caused the database update to fail. Then copy the backup file to the database file and delete the backup file.

03353013 Database already exists.
Explanation: A request to create a new database cannot be completed because the database file already exists.
User response: Choose a different name for the new database or delete the existing database.

03353014 Record is too big.
Explanation: A new record cannot be added to the database because it is larger than the database record length.
User response: Create a new database with a larger record length.

03353015 Database password is expired.
Explanation: The database password is expired.
User response: Change the database password.

03353016 The password is not correct.
Explanation: The wrong password is specified for a key database, an encrypted private key, or an import file. This error can also occur if the file has been modified.
User response: Specify the correct password.

03353017 Access denied.
Explanation: The database or keyring cannot be opened because the permissions do not allow access by the current user.
User response: Ensure that the user has read/write access to the database if opening the database for update mode; otherwise ensure that the user has read access to the database or keyring.

03353018 Database is locked for update.
Explanation: Another process has opened the database in update mode. Only one process may have the database open for update at a time.
User response: Wait until the database has been closed by the other process and then retry the request.

03353019 Record length is too small.
Explanation: The database record length is less than the minimum value of 2500.
User response: Specify a record length of 2500 or greater.
0335301A  No private key.
Explanation: A private key request cannot be processed because the database entry does not contain a private key. This error can occur if the private key is stored in the Integrated Cryptographic Service Facility (ICSF) but the CSF started task is not running.
User response: Verify that the CSF started task is running if the private key is stored in ICSF. Otherwise, repeat the failing request using a database entry containing a private key.

0335301B  Record label is not valid.
Explanation: The record label is not valid. A label may contain letters, numbers, and punctuations. A record label may not be an empty string.
User response: Provide a valid record label.

0335301C  Record label is not unique.
Explanation: A record label must be unique within a database.
User response: Provide a unique record label.

0335301D  Record type is not valid.
Explanation: The database record type is not valid.
User response: Provide a valid database record type.

0335301E  Duplicate certificate.
Explanation: An attempt is made to add a certificate to a key database but the database already contains the certificate. A certificate is a duplicate if the issuer name and certificate serial number are the same.
User response: Delete the existing certificate before adding the new certificate.

0335301F  Incorrect Base64 encoding.
Explanation: An encoded stream cannot be decoded because it contains an incorrect Base64 encoding. A Base64 encoding consists of a header line, encoded text, and a footer line. The encoded text is encoded using a 64-character subset in groups of 4 characters.
User response: Ensure that the encoded stream has not been truncated or modified. Base64 encoding uses text data and must be in the local code page. Contact your service representative if the error persists.

03353020  Unrecognized file or message encoding.
Explanation: A file or message cannot be imported because the format is not recognized.
User response: Contact your service representative if the error persists.

System SSL supports X.509 DER-encoded certificates, PKCS #7 signed data messages, and PKCS #12 personal information exchange messages for certificate import files. The import file data may be the binary data or the Base64-encoding of the binary data.

System SSL supports PKCS #7 data, encrypted data, signed data, and enveloped data for messages. This error can also occur if the message is not constructed properly.

User response: Ensure that the import file or message has not been modified. A Base64-encoded import file must be converted to the local code page when it is moved to another system while a binary import file must not be modified when it is moved to another system.

03353021  Certificate is not yet valid.
Explanation: The current time is earlier than the beginning of the certificate validity.
User response: Either wait until the certificate is valid or request a new certificate with an earlier starting date from the certification authority.

03353022  Certificate is expired
Explanation: The current time is after the end of the certificate validity.
User response: Request a new certificate from the certification authority.

03353023  Name format is not supported.
Explanation: An unsupported name format is encountered while validating a certificate.
User response: Contact your service representative.

03353024  Issuer certificate not found.
Explanation: An issuer certificate is not found while validating a certificate. This error can occur if the issuer certificate required for a new certificate is not in the key database or if the required issuer certificate is not trusted or has expired.
User response: Ensure that the key database contains the required issuer certificate and that the certificate is marked as trusted. Refer to "Database Menu" on page 375 for information on displaying the contents of an external certificate file in order to verify which issuer certificate is required. Contact your service representative if the error persists.

03353025  Certification path is too long.
Explanation: The certification path length exceeds the maximum specified in the certification authority certificate.

Chapter 13. Messages and Codes 465
03353031  Signer certificate not found.
Explanation:  An signer certificate is not found while creating or processing a signed message.
User response:  Provide a certificate for each signer, including signers of authenticated attributes.

03353032  Content type is not supported.
Explanation:  An unsupported PKCS #7 content type is encountered.
User response:  Refer to the Programming Reference for the failing routine to determine the supported content types.

03353033  Recipient certificate not found.
Explanation:  A recipient certificate is not found while creating or processing an enveloped message.
User response:  Provide at least one recipient certificate.

03353034  Encryption key size is not supported.
Explanation:  The encryption key size is not supported by the System SSL runtime.
User response:  Refer to the System SSL information to determine which key sizes are supported. In general, when executing in non-FIPS mode, 40-bit keys and 128-bit keys are supported for RC2 and RC4, 56-bit keys are supported for DES, 168-bit keys are supported for Triple DES, and 128-bit keys and 256-bit keys are supported for AES. RSA keys must be between 512 and 4096 bits, DSS keys must be between 512 and 1024 bits, and Diffie-Hellman keys must be between 512 and 2048 bits.

When executing in FIPS mode, 168-bit keys are supported for Triple DES, and 128-bit keys and 256-bit keys are supported for AES. RSA keys must be between 1024 and 4096 bits, DSS keys must be 1024 bits, and Diffie-Hellman keys must be 2048 bits.

This error can also occur if the requested key size is not compatible with the supplied key generation parameters. Refer to the System SSL information to determine which key sizes are supported.

03353035  Encryption key parity is not correct.
Explanation:  DES and Triple DES encryption keys must have odd parity for each key byte.
User response:  Verify that the key is generated correctly. Contact your service representative if the error persists.

03353036  Encryption key is weak.
Explanation:  A small subset of the possible DES and Triple DES encryption keys are weak and can be broken more easily than the rest of the keys. For this reason, the weak keys should be avoided when generating a DES or Triple DES key.
User response:  Contact your service representative.

03353037  Initial vector size is not correct.
Explanation:  The initial vector used by the encryption routine is not the correct length.
User response:  Contact your service representative.

03353038  Encryption data size is not correct.
Explanation:  The length of the encryption data is not correct. For symmetric key algorithms using cipher block chaining, the encryption data must be a multiple of the cipher block size. For asymmetric key algorithms, the encryption data must be the same length as the cipher key modulus.
User response:  Verify that the encryption data has not been truncated. Contact your service representative if the error persists.

03353039  Encryption block format is not correct.
Explanation:  The encryption block format is not correct following decryption. This error can occur if the wrong key is used to decrypt the block.
User response:  Verify that the correct key is being used to decrypt the data. Contact your service representative if the error persists.

0335303A  Number does not have a modular inverse.
Explanation:  The cryptographic support is unable to find an inverse for a number.
User response:  Contact your support representative.

0335303B  LDAP processing error.
Explanation:  An error is detected while setting up the LDAP environment or retrieving an LDAP directory entry.
User response:  Ensure that the LDAP server is running and that there are no network errors. Collect a System SSL trace containing the error and then contact your service representative if the error persists.
0335303C LDAP is not available.

**Explanation:** The System SSL runtime is unable to access the LDAP server.

**User response:** Ensure that the LDAP server is running and that there are no network problems. Collect a System SSL trace and then contact your service representative if the error persists.

0335303D Digest data size is not correct.

**Explanation:** The length of the digest data is not correct. Digest data size by algorithm is:
- MD2 – 16 bytes
- MD5 – 16 bytes
- SHA-1 – 20 bytes
- SHA-224 – 28 bytes
- SHA-256 – 32 bytes
- SHA-384 – 48 bytes
- SHA-512 – 64 bytes

**User response:** Verify that the data has not been truncated. Contact your service representative if the error persists.

0335303E Database name is not valid.

**Explanation:** The database file name or SAF key ring name is not valid. The length of the fully-qualified database file name must be between 1 and 251 while the length of the SAF key ring must be between 1 and 237.

**User response:** Provide a valid database name.

0335303F Database open failed.

**Explanation:** The System SSL runtime is unable to open the HFS database file, SAF key ring or z/OS PKCS #11 Token.

**User response:** Verify that the database file, SAF key ring or z/OS PKCS #11 Token exists and is accessible by the application. Collect a System SSL trace and then contact your service representative if the error persists.

03353040 Self-signed certificate not in database.

**Explanation:** A self-signed certificate cannot be validated because it is not in the key database, SAF key ring or z/OS PKCS #11 token.

**User response:** Add the self-signed certificate to the key database, SAF key ring or z/OS PKCS #11 token.

This code may also occur if the intermediate certificate on the key ring was not marked Trusted. If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure the latest changes are available.

03353041 Certificate is revoked.

**Explanation:** A certificate is revoked and cannot be used.

**User response:** Obtain a new certificate from the certification authority.

03353042 Issuer name is not valid.

**Explanation:** The certificate issuer name must be a non-empty X.509 distinguished name.

**User response:** Obtain a new certificate with a valid issuer name.

03353043 Subject name is not valid.

**Explanation:** The certificate subject name must be either a non-empty distinguished name or an empty distinguished name with a SubjectAltName certificate extension.

**User response:** Obtain a new certificate with a valid subject name.

03353044 Name constraints violated.

**Explanation:** The certificate name is not allowed by the certification path name constraints.

**User response:** Report the problem to the certification authority.

03353045 No content data.

**Explanation:** The PKCS #7 content information does not contain any content data.

**User response:** Change the application to provide content data for the content information.

03353046 Message version is not supported.

**Explanation:** An unsupported message version is encountered.

**User response:** Refer to the Programming Reference for the failing routine to determine the supported message versions.

03353047 Subject name is same as signer name.

**Explanation:** A request to create a new certificate cannot be processed because the requested subject name is the same as the subject name of the signing certificate.

**User response:** Choose a different subject name for the new certificate.
03353048  Diffie-Hellman group parameters are not valid.

**Explanation:** The Diffie-Hellman group parameters are not valid. The subprime Q must be greater than 1 and less than the prime P. The base G must be greater than 1 and less than the prime P. Refer to RFC 2631 (Diffie-Hellman Key Agreement Method) for more information on how the Diffie-Hellman parameters are generated.

**User response:** Verify that the correct parameters are supplied when calling the failing routine. Contact the certification authority if the Diffie-Hellman group parameters are obtained from an X.509 certificate. Otherwise, collect a System SSL trace and then contact your service representative.

03353049  Diffie-Hellman values are not valid.

**Explanation:** The Diffie-Hellman values are not valid. The private value X must be greater than 1 and less than the prime P. The public value Y must be greater than 1 and less than the prime P. In addition, the result of raising the public value Y to the power of the subprime Q modulo the prime P must be equal to 1. Refer to RFC 2631 (Diffie-Hellman Key Agreement Method) for more information on how the Diffie-Hellman values are generated.

**User response:** Contact the certification authority if the Diffie-Hellman values are obtained from an X.509 certificate. Otherwise, collect a System SSL trace and then contact your service representative.

0335304A  Digital Signature Standard parameters are not valid.

**Explanation:** The Digital Signature Standard parameters are not valid. The subprime Q must be greater than 1 and less than the prime P. The base G must be greater than 1 and less than the prime P. Refer to FIPS 186-2 (Digital Signature Standard) for more information on how the parameters are generated.

**User response:** Verify that the correct parameters are supplied when calling the failing routine. Contact the certification authority if the DSS parameters are obtained from an X.509 certificate. Otherwise, collect a System SSL trace and then contact your service representative.

0335304B  Certificate not valid for host.

**Explanation:** A server certificate does not contain the current host name as either the common name (CN) element of the subject name or as a DNS entry for the subject alternate name.

**User response:** Obtain a new certificate containing the desired host name.

---

0335304C  No certificate in import file.

**Explanation:** The import file does not contain an X.509 certificate.

**User response:** Specify a valid certificate import file.

0335304D  The content-type authenticated attribute is not allowed.

**Explanation:** The set of authenticated attributes supplied within the `attributes_signers` parameter must NOT include the content-type authenticated attribute as this is automatically provided by `gsk_make_signed_data_content_extended()` and `gsk_make_signed_data_msg_extended()`.

**User response:** Do not include content-type or message-digest in the set of authenticated attributes supplied to `gsk_make_signed_data_content_extended()` or `gsk_make_signed_data_msg_extended()`.

0335304E  The message-digest authenticated attribute is not allowed.

**Explanation:** The set of authenticated attributes supplied via the `attributes_signers` parameter must NOT include the message-digest authenticated attribute as this is automatically provided by `gsk_make_signed_data_content_extended()` and `gsk_make_signed_data_msg_extended()`.

**User response:** Do not include content-type or message-digest in the set of authenticated attributes supplied to `gsk_make_signed_data_content_extended()` or `gsk_make_signed_data_msg_extended()`.

0335304F  Attribute identifier is not valid.

**Explanation:** The attribute identifier is not valid.

**User response:** Specify a valid attribute identifier.

03353050  Enumeration is not valid.

**Explanation:** The enumeration value is not valid.

**User response:** Specify a valid enumeration value.

03353051  CA certificate not supplied

**Explanation:** A signing CA certificate was not supplied on the call.

**User response:** Supply a CA certificate on the function call.
03353052 Validation option is not valid.
Explanation: The specified validation option is not valid.
User response: Specify a valid validation option.

03353053 Certificate request not supplied.
Explanation: A certificate request structure was not supplied on the call.
User response: Supply a certificate request structure on the function call.

03353054 Public key info not supplied.
Explanation: A pkcs_public_key_info structure was not supplied on the call.
User response: Supply a pkcs_public_key_info structure on the function call.

03353055 Modulus bits not supplied.
Explanation: The number of modulus bits was not supplied on the call.
User response: Supply the number of modulus bits on the function call.

03353056 Exponent not supplied.
Explanation: A gsk_buffer structure containing the exponent was not supplied on the call.
User response: Supply a gsk_buffer structure containing the exponent on the function call.

03353057 Private key info not supplied.
Explanation: A pkcs_private_key_info structure was not supplied on the call.
User response: Supply a pkcs_private_key_info structure on the function call.

03353058 Modulus not supplied.
Explanation: A gsk_buffer structure containing the modulus was not supplied on the call.
User response: Supply a gsk_buffer structure containing the modulus on the function call.

03353059 Public exponent not supplied.
Explanation: A gsk_buffer structure containing the public exponent was not supplied on the call.
User response: Supply a gsk_buffer structure containing the public exponent on the function call.

0335305A Private exponent not supplied.
Explanation: A gsk_buffer structure containing the private exponent was not supplied on the call.
User response: Supply a gsk_buffer structure containing the private exponent on the function call.

0335305B First prime not supplied.
Explanation: A gsk_buffer structure containing the first prime was not supplied on the call.
User response: Supply a gsk_buffer structure containing the first prime on the function call.

0335305C Second prime not supplied.
Explanation: A gsk_buffer structure containing the second prime was not supplied on the call.
User response: Supply a gsk_buffer structure containing the second prime on the function call.

0335305D First prime exponent not supplied.
Explanation: A gsk_buffer structure containing the first prime exponent was not supplied on the call.
User response: Supply a gsk_buffer structure containing the first prime exponent on the function call.

0335305E Second prime exponent not supplied.
Explanation: A gsk_buffer structure containing the second prime exponent was not supplied on the call.
User response: Supply a gsk_buffer structure containing the second prime exponent on the function call.

0335305F CRT coefficient not supplied.
Explanation: A gsk_buffer structure containing the CRT coefficient was not supplied on the call.
User response: Supply a gsk_buffer structure containing the CRT coefficient on the function call.

03353060 Certificate revocation list cannot be found.
Explanation: The security setting for gsk_crl_security_level is set to HIGH and the required certificate revocation list (CRL) cannot be found in the specified LDAP server.
User response: Contact the certification authority and obtain the required CRL.
03353061 Multiple certificates exist for label.

Explanation: Access of certificate/key via label could not be resolved because multiple certificates/keys exist with the label.

User response: Correct certificate/key store so that label specifies a unique record.

03353062 Multiple keys are marked as the default.

Explanation: Access of key via default status could not be resolved because multiple keys are marked as the default key.

User response: Correct the certificate/key store so that only one key is marked as the default key.

03353064 Digest type and key type are incompatible.

Explanation: The specified digest algorithm and the key algorithm are incompatible.

User response: Specify a digest algorithm that is compatible with the signing key algorithm.

03353065 Generate random bytes input buffer not valid.

Explanation: The input buffer to gsk_generate_random_bytes is not valid.

User response: Ensure a valid gsk_buffer structure has been supplied to the gsk_generate_random_bytes API. Contact your service representative if the error persists.

03353066 Generate random bytes produced duplicate output.

Explanation: The Random Number Generator has produced identical consecutive blocks of output data. If in FIPS mode, any further attempts to use System SSL will continue to fail until the application is restarted or the executing process is reinitialized.

User response: Restart the SSL application or process to reinitialize the SSL DLLs. If the problem persists, collect a System SSL trace containing the error and contact your service representative.

03353067 Known Answer Test has failed.

Explanation: A Known Answer Test has failed to match the expected results. Any further attempts to use System SSL will continue to fail until the application is restarted or the executing process is reinitialized.

User response: Restart the SSL application or process to reinitialize the SSL DLLs. If the problem persists, collect a System SSL trace containing the error and contact your service representative.

03353068 API is not supported.

Explanation: The API is not supported. An attempt was made to use an API that is not supported in the current mode of operation (FIPS or non-FIPS).

User response: Ensure that the API being utilized is supported in the mode in which the application is executing. If you are invoking a FIPS-only API, you will need to restart your application in FIPS mode.

03353069 Key database is not a FIPS mode database.

Explanation: While executing in FIPS mode, an attempt was made to open a key database that is non-FIPS.

User response: Specify a key database that meets FIPS 140-2 criteria if running in FIPS mode.

0335306A Key database can only be opened for update if running in FIPS mode.

Explanation: While executing in non-FIPS mode, an attempt was made to open a FIPS key database for update.

User response: To open a FIPS key database for update, you must be executing in FIPS mode.

0335306B Cannot switch from non-FIPS mode to FIPS mode.

Explanation: While executing in non-FIPS mode, an attempt was made to switch to FIPS mode.

User response: Once executing in non-FIPS mode it is not possible to switch to FIPS mode.

0335306C Attempt to execute in FIPS mode failed.

Explanation: A request to execute in FIPS mode failed because the required System SSL DLLs could not be loaded.

User response: Ensure that the Cryptographic Services Security Level 3 FMID is installed.

0335306D Acceptable policy intersection cannot be found.

Explanation: The Certificate Policies extension of the certificate does not contain an acceptable policy as required by the application or an issuing certificate.

User response: Ensure the certificate chain is valid and the user certificate is intended to be used for the required purpose.
Variable argument count is not valid.

Explanation: The specified variable argument count is not valid.

User response: Specify a valid variable argument count.

Required certificate extension is missing.

Explanation: A certificate extension that is mandatory for the certificate to be used for the required purpose has not been found.

User response: Ensure the certificate chain is correct and complies with the validation mode defined for the connection. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

Certificate extension data is incorrect.

Explanation: A certificate extension has incorrect data or has a necessary field missing.

User response: Ensure the certificate chain is correct and complies with the validation mode defined for the connection. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

Certificate extension data has an incorrect critical indicator.

Explanation: A critical indicator for a certificate extension is incorrect. Either the extension is required to be marked critical and is marked non-critical or is required to be marked non-critical and is marked critical.

User response: Ensure the certificate chain is correct and complies with the validation mode defined for the connection. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

Certificate contains a duplicate extension.

Explanation: The certificate or CRL undergoing validation contains multiple certificates or CRL extensions of the same type.

User response: Ensure the certificate chain is correct and complies with the validation mode defined for the connection. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

Cannot match CRL distribution points.

Explanation: The DN in the Issuing Distribution Point extension of the CRL does not match a suitable DN in the certificate undergoing validation. The DN in the Issuing Distribution Point extension must match either:

- a DN of type fullName in the Distribution Point of the CRL Distribution Points extension of the certificate undergoing validation
- the CRLIssuer field in the Distribution Point of the CRL Distribution Points extension of the certificate undergoing validation
- the Certificate Issuer name, if no CRL Distribution Point extension exists in the certificate undergoing validation

User response: Ensure the certificate chain is correct and complies with the validation mode defined for the connection. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

FIPS mode key generation failed pair-wise consistency check.

Explanation: While executing in FIPS mode, a key pair was generated that failed a pair-wise consistency check. Any further attempts to use System SSL will continue to fail until the application is restarted or the executing process is reinitialized.

User response: Restart the SSL application or process to reinitialize the SSL DLLs. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.
SSL Started Task Messages (GSK01nnn)

This topic describes SSL started task messages.

**GSK01001I System SSL version version.release Service level level.**

**Explanation:** This message displays the System SSL version, release, and service level.

**User response:** None

**GSK01002E Insufficient storage available.**

**Explanation:** The SSL server is unable to obtain storage for an internal control block.

**User response:** Increase the storage available to the GSKSRVR started task and then retry the request.

**GSK01003I SSL server initialization complete.**

**Explanation:** The server initialization is complete.

**User response:** None

**GSK01004I SSL server shutdown requested.**

**Explanation:** The system operator has entered a STOP command for the SSL server.

**User response:** None

**GSK01005I Unrecognized SSL server command: Specify DISPLAY, TRACE or STOP.**

**Explanation:** An unrecognized command name is specified on a MODIFY operator command. The valid SSL server commands are DISPLAY, TRACE and STOP.

**User response:** Specify a valid SSL server command.

**GSK01006E Incorrect command option specified.**

**Explanation:** An incorrect SSL server command option is specified.

The valid DISPLAY command options are:
- CRYPTO - Display the available encryption algorithms.
- LEVEL - Display the System SSL version, release, and service level.
- SIDCACHE - Display the sysplex session cache status.
- XCF - Display SSL sysplex status.

The valid TRACE command options are:
- OFF - Turn of SSL tracing
- ON,level - Enable SSL tracing using the specified trace level.

**User response:** Specify a valid command option.

**GSK01007E Missing command option.**

**Explanation:** An SSL server command is entered which requires a command option but no command option is entered.

**User response:** Enter a complete SSL server command.

**GSK01008I Sysplex status.**

**Explanation:** This message is displayed in response to the SSL server DISPLAY XCF command. The remaining lines in this multi-line message display the status of each SSL server in the sysplex. A server is ACTIVE if the GSKSRVR started task is running. A security server is INACTIVE if the GSKSRVR started task has been stopped. No entry is displayed for a system where the GSKSRVR started task has not been started.

**User response:** None

**GSK01009I Cryptographic status.**

**Explanation:** This message is displayed in response to the SSL server DISPLAY CRYPTO command. The remaining lines in this multi-line message display the available encryption algorithms.

**User response:** None

**GSK01010A The SSL server is already running.**

**Explanation:** The GSKSRVR started task is already running. Only one instance of the SSL server may be active in the same system.

**User response:** Stop the GSKSRVR started task before starting a new instance of the SSL server.

**GSK01011A The SSL server is not APF-authorized.**

**Explanation:** The GSKSRVR started task is not running with APF authorization.

**User response:** Add the pdsename.SIEALNKE dataset to the list of APF-authorized datasets and then restart the GSKSRVR started task. If you are using a STEPLIB or JOBLIB for the GSKSRVR started task, verify that all datasets in the concatenation are APF-authorized.

**GSK01012A Unable to make address space non-swappable: Error error-code.**

**Explanation:** The SSL server is unable to make its address space non-swappable. The error code is the value returned by the SYSEVENT system service.
User response: Verify that the GSKSRVR started task is APF-authorized. Refer to the SYSEVENT description in z/OS MVS Authorized Assembler Services Reference for more information. Contact your service representative if the error persists.

GSK01013I SSL server restart registration complete on system.
Explanation: The GSKSRVR started task has successfully registered with ARM (Automatic Restart Management) on the indicated system. The GSKSRVR started task will be automatically restarted if it fails unexpectedly (it will not be restarted if it detects an error and stops). The ARM element type is SYSSSL and the ARM element name is GSKSRVR_system-name. The ARM policy can be used to override the default registration values if needed.
User response: None

GSK01014I SSL server restarting on system.
Explanation: The GSKSRVR started task is being restarted following an unexpected failure. The RESTART_ATTEMPTS value in the ARM policy determines the number of restarts which will be attempted.
User response: None

GSK01015E Unable to register for restart: Error error-code, Reason reason-code.
Explanation: The GSKSRVR started task is unable to register with ARM (Automatic Restart Management). The IXCARM request failed with the indicated error and reason codes.
User response: Refer to the IXCARM description in z/OS MVS Sysplex Services Reference for more information. Contact your service representative if the error persists.

GSK01016E Unable to deregister for restart: Error error-code, Reason reason-code.
Explanation: The GSKSRVR started task is unable to deregister with ARM (Automatic Restart Management). The IXCARM request failed with the indicated error and reason codes.
User response: Refer to the IXCARM description in z/OS MVS Sysplex Services Reference for more information. Contact your service representative if the error persists.

GSK01017I SSL server restart deregistration complete on system.
Explanation: The GSKSRVR started task has successfully deregistered with ARM (Automatic Restart Management) on the indicated system. The SSL server will no longer be automatically restarted if it fails unexpectedly.
User response: None

GSK01018I Trace option processed: trace-option.
Explanation: The indicated trace request has been processed by the SSL server.
User response: None

GSK01019E Unable to create mutex: error-text.
Explanation: The GSKSRVR started task is unable to create a mutex for the indicated reason.
User response: Refer to the pthread_mutex_init description in z/OS C/C++ Run-Time Library Reference for more information. Contact your service representative if the error persists.

GSK01020E Unable to lock mutex: error-text.
Explanation: The GSKSRVR started task is unable to lock a mutex for the indicated reason.
User response: Refer to the pthread_mutex_lock description in z/OS C/C++ Run-Time Library Reference for more information. Contact your service representative if the error persists.

GSK01021E Unable to create thread: error-text.
Explanation: The GSKSRVR started task is unable to create a thread for the indicated reason.
User response: Refer to the pthread_create description in z/OS C/C++ Run-Time Library Reference for more information. Contact your service representative if the error persists.

GSK01022E Unable to initialize local services: Error error-code, Reason reason-code.
Explanation: The GSKSRVR started task is unable to initialize the local services support. The error code indicates the failing system function and the reason code is the error code returned by the system function.
These error codes are defined:
- 1 = The job step is not APF-authorized.
- 2 = The security server is already running.
- 3 = The ESTAEX request failed.
- 5 = The LXRES request failed.
- 6 = The ETCRE request failed.
- 7 = The ETCON request failed.
- 8 = The IEANTCR request failed.
- 9 = The CTRACE DEFINE request failed.
User response: Verify that the GSKSRVR started task is APF-authorized. Refer to the system function description in z/OS MVS Authorized Assembler Services
Reference for more information. Contact your service representative if the error persists.

GSK01023E Unable to create session cache data space: Error error-code, Reason reason-code.

Explanation: The GSKSRVR started task is unable to create the session cache data space.

These error codes are defined:
1 = DSPSERV CREATE failed.
The reason code contains the DSPSERV return code in the upper halfword and bits 8-23 of the DSPSERV reason code in the lower halfword.
2 = ALESERV ADD failed.
The reason code is the ALESERV return code.

User response: Refer to the DSPSERV or ALESERV description in z/OS MVS Authorized Assembler Services Reference for more information. Contact your service representative if the error persists.

GSK01024E Unable to initialize cross-system services: Error error-code, Reason reason-code.

Explanation: The GSKSRVR started task is unable to initialize cross-system services.

These error codes are defined:
1 = The job step is not APF-authorized.
3 = IXCJOIN failed.
The reason code contains the IXCJOIN return code in the upper halfword and the IXCJOIN reason code in the lower halfword.
4 = IXCQUERY failed.
The reason code contains the IXCQUERY return code in the upper halfword and the IXCQUERY reason code in the lower halfword.

User response: Refer to the IXCJOIN or IXCQUERY description in z/OS MVS Sysplex Services Reference for more information. Contact your service representative if the error persists.

GSK01025I System name has joined the GSKSRVR group.

Explanation: The GSKSRVR started task has completed initialization on the indicated system and is now a member of the GSKSRVGP cross-system group.

User response: None

GSK01026I System name has left the GSKSRVR group.

Explanation: The GSKSRVR started task is stopping on the indicated system and has left the GSKSRVGP cross-system group.

User response: None

GSK01027I Cross-system services ended due to sysplex partitioning.

Explanation: The local system is leaving the sysplex. As a result, GSKSRVR cross-system services are no longer available.

User response: None

GSK01028E Local program call request failed: Error error-code.

Explanation: The GSKSRVR started task is unable to process a local program call request.

These error codes are defined:
• 8 = Parameter buffer overflow.
• 12 = Unable to allocate storage.
• 16 = Local service support is not enabled.
• 20 = Program call task abended.
• 24 = Unable to obtain control lock.
• 28 = Requested function is not supported.

User response: Contact your service representative.

GSK01029I Cross-system services are not available.

Explanation: The DISPLAY XCF command cannot be processed because cross-system services are not available.

User response: None

GSK01030I Maximum number of lines displayed.

Explanation: The maximum number of lines allowed for a multi-line write-to-operator message has been reached.

User response: None

GSK01031I No session cache users.

Explanation: The DISPLAY SIDCACHE command was issued but there are no session cache users to display.

User response: None

GSK01032I Session cache status

Explanation: This message is displayed in response to the SSL server DISPLAY SIDCACHE command. The remaining lines in this multi-line message display the cache users.

User response: None
GSK01033E Unable to extend the session cache data space: Error error-code, Reason reason-code.

Explanation: The GSKSRVR started task is unable to increase the size of the session cache data space.

The error codes have these values:

1 = DSPSERV EXTEND failed.
The reason code contains the DSPSERV return code in the upper halfword and bits 8-23 of the DSPSERV reason code in the lower halfword.

User response: The new session cache entry is not stored in the session cache data space. Refer to the DSPSERV description in z/OS MVS Authorized Assembler Services Reference for more information. Contact your service representative if the error persists.

GSK01034E Unable to send cross-system message: Error error-code, Reason reason-code.

Explanation: The GSKSRVR started task is unable to send a message to another member of the GSKSRVGP group.

The error codes have these values:

1 = Unable to obtain XCF control lock on target system.
2 = Cross-system services are not available.
3 = Requested token not found on target system.
4 = User not authorized to access token data.
5 = Unable to allocate storage on the target system.
6 = Target replica is not a member of the GSKSRVGP group.
7 = Target replica is not active.
8 = IXCMSGO failed. The reason code contains the IXCMSGO return code in the upper halfword and the IXCMSGO reason code in the lower halfword.
9 = IXCMSGI failed on the target system. The reason code contains the IXCMSGI return code in the upper halfword and the IXCMSGI reason code in the lower halfword.
10 = Request function code is not supported.
11 = Request canceled.
12 = Unknown notification message.
13 = No response received from target system.
14 = Unable to allocate storage on the local system.
15 = IXCMSGI failed on the local system. The reason code contains the IXCMSGI return code in the upper halfword and the IXCMSGI reason code in the lower halfword.

User response: Verify that the GSKSRVR started task is running on the local system. Contact your service representative if the error persists.

GSK01035E SSL server is not available.

Explanation: The SSL server task is not available. This error will occur if the GSKSRVR started task is not running, has not completed initialization, or is stopping.

User response: Wait until the GSKSRVR started task is available and then retry the failing request.

GSK01036E No job name specified.

Explanation: No job name was specified on the TRACE CT command when starting a component trace.

User response: Specify at least one job name when starting a component trace.

GSK01037E Unable to call SSL server: Error error-code, Reason reason-code.

Explanation: The command processor for the TRACE CT command is unable to call the GSKSRVR started task.

These error codes are defined:

- 8 = Parameter buffer overflow
- 12 = Unable to allocate storage
- 16 = Local service support is not enabled
- 20 = Program call task abended (the reason is the abend code)
- 24 = Unable to obtain control lock
- 28 = Requested function is not supported

User response: Specify a valid SSL trace option.

GSK01038E Incorrect trace option specified.

Explanation: The OPTIONS parameter on the TRACE CT command does not specify a valid SSL trace option. The only valid option is LEVEL=n where n is the requested SSL trace level. Refer to the description of the GSK_TRACE environment variable for more information on setting the SSL trace level.

User response: Specify a valid SSL trace option.

GSK01039E The trace buffer size must be between 64K and 512K.

Explanation: The trace buffer size specified on the TRACE CT command must be between 64K and 512K.

User response: Specify a valid trace buffer size.

GSK01040I SSL component trace started.

Explanation: The SSL component trace has been started. The jobs specified on the TRACE CT command may be already running or may be started after the TRACE CT command is processed. However, any jobs that are already running must have been started after
the GSKSRVR started task was started.

User response: None

GSK01041I SSL component trace ended.
Explanation: The SSL component trace has ended.
User response: None

GSK01042E Incorrect OPTIONS syntax
Explanation: The OPTIONS parameter syntax on the IPCS CTRACE command is not correct for an SSL component trace. SSL supports three options: JOB, PID, and TID. The CTRACE OPTIONS parameter is specified as CTRACE COMP(GSKSRVR) OPTIONS((JOB(name),PID(hexid),TID(hexid))).
User response: Specify a valid OPTIONS parameter.

GSK01043E Incorrect trace option.
Explanation: An incorrect trace option was specified on the IPCS CTRACE command for an SSL component trace. SSL supports three options: JOB, PID, and TID. The CTRACE OPTIONS parameter is specified as CTRACE COMP(GSKSRVR) OPTIONS((JOB(name),PID(hexid),TID(hexid))). The job name must be 1-8 characters. The hexadecimal identifier for PID and TID must be 1-8 hexadecimal digits.
User response: Specify a valid OPTIONS parameter.

GSK01044E Duplicate trace option.
Explanation: An SSL trace option is specified more than once on the IPCS CTRACE command.
User response: Do not specify the same trace option more than once.

GSK01045E Incorrect hexadecimal value.
Explanation: The value for the PID and TID trace options for the IPCS CTRACE command must be a hexadecimal value consisting of 1-8 hexadecimal digits.
User response: Specify a valid hexadecimal value.

GSK01046I Trace filter options: option list
Explanation: The IPCS CTRACE command specifies one or more trace entry filter options.
User response: None

GSK01047I SSL component trace started for jobname/JobID.
Explanation: The SSL component trace has started for the indicated job. This message is displayed once for each job that matches the jobnames specified in the TRACE CT command. Tracing is started and the message is displayed when SSL component trace has been started and activation has been detected by the System SSL APIs.
User response: None

GSK01048W Component trace buffer overflow.
Explanation: Both of the SSL component trace buffers are full and additional trace entries cannot be added until the trace writer has written the current data to the trace dataset. Trace entries will be discarded until the trace writer has emptied one of the trace buffers.
User response: Increase the trace buffer size specified on the TRACE command and restart the component trace.

GSK01049A The SSL server must be started as a started task.
Explanation: The GSKSRVR was not started as a started task. The userid of the GSKSRVR start task needs to be defined to the started procedure. See "Configuring the SSL Started Task" on page 435 for more information.
User response: Start GSKSRVR as a started task.

GSK01050I SSL Component trace started for Jobname/JobID/ProcessID
Explanation: The SSL component trace has started for the indicated process. This message is displayed each time component trace is started for each SSL process whose jobname matches one of the jobnames specified in the TRACE CT command. Tracing is started and the message is displayed when SSL component trace has been started and activation has been detected by the System SSL APIs. This message is written to the system log only.
User response: None.

GSK01051E Jobname/ASID Hardware encryption error. ICSF hardware encryption processing is unavailable
Explanation: The specified job has encountered a severe hardware encryption error during ICSF hardware processing. Encryption functions will be processed in software. Refer to the system log for algorithm specific detail.
User response: Ensure that ICSF hardware encryption services are installed and functioning correctly. Restart the SSL application or process to reinitialize the SSL DLLs.
GSK01052W  Jobname/ASID Hardware encryption error. Algorithm encryption processing switched to software

Explanation: The specified job has encountered a severe hardware encryption error. Hardware processing for the specified algorithm has been disabled. Any future encryption or decryption using this algorithm will be performed in software for the particular SSL application or process.

User response: Ensure that ICSF hardware encryption services are installed and functioning correctly. Restart the SSL application or process to reinitialize the SSL DLLs.

GSK01053E  Known Answer Tests failed with status code

Explanation: The FIPS power-on known answer tests failed with the reported CMS status code. System SSL is unable to execute in FIPS mode.

User response: Refer to "CMS Status Codes (03353xxx)" on page 463 for information on the reported status code. Collect a System SSL trace of the failing application and contact your service representative if the error persists.

GSK01054E  SSL server starting in non-FIPS mode.

Explanation: The environment variable GSK_FIPS_STATE was specified in the envar file in the GSKSRVR home directory, yet the started task was unable to execute in FIPS mode. The started task is started in non-FIPS mode.

If the indicated CMS status code is zero, then the value specified for the environment variable was not GSK_FIPS_STATE_ON, consequently FIPS mode was not attempted. If the indicated CMS status code is non-zero, an attempt was made to set FIPS mode but failed.

The System SSL started task will continue to execute in non-FIPS mode. In non-FIPS mode, GSKSRVR does not provide sysplex session ID caching for FIPS mode application servers. Sysplex session ID caching is provided only for non-FIPS mode application servers.

User response: If the indicated status is zero, correct the environment variable GSK_FIPS_STATE so that it either specifies the value ‘GSK_FIPS_STATE_ON’ or remove the environment variable if FIPS mode is not required for the started task. If the indicated status is non-zero, refer to "CMS Status Codes (03353xxx)" on page 463 for information on the reported status code. Collect a System SSL trace of the failing application and contact your service representative if the error persists.

GSK01057I  SSL server starting in FIPS mode.

Explanation: GSK_FIPS_STATE=GSK_FIPS_STATE_ON was specified in the envar file in the GSKSRVR home directory.

The System SSL started task has initialized successfully and will execute in FIPS mode. In FIPS mode, GSKSRVR provides sysplex session ID caching for both FIPS mode and non-FIPS mode application servers.

User response: None
Utility Messages (GSK00nnn)

This topic describes utility messages.

GSK00001E Unable to open trace file name: error-message

Explanation: The gsktrace command is unable to open the trace file.

User response: Verify that the trace file exists and can be accessed by the user issuing the gsktrace command. Contact your service representative if the error persists.

GSK00002E Unable to read trace file name: error-message

Explanation: The gsktrace command is unable to read the trace file.

User response: Verify that there are no filesystem errors and that the trace file has not been modified. Contact your service representative if the error persists.

GSK00003E Trace record length size exceeds the maximum length.

Explanation: A record in the trace file is longer than the maximum length for a trace record. This probably means the trace file has been modified.

User response: Verify that the trace file has not been modified and was created by a compatible level of the System SSL runtime.

GSK00004R Enter password:

Explanation: The System SSL runtime is needs a database or certificate password.

User response: Enter the requested password.

GSK00005R Re-enter password:

Explanation: The System SSL runtime is verifying the password.

User response: Enter the same password you entered for the first password prompt.

GSK00006E File name is not a valid SSL trace file.

Explanation: The gsktrace command is unable to process the file because it is not in the proper format. This error can occur if the trace file was created by an earlier level of the System SSL runtime.

User response: Process the trace file using the gsktrace command that is at the same level as the System SSL runtime which created the trace file.

GSK00007R Enter new password:

Explanation: The System SSL runtime is needs a new database password.

User response: Enter the requested password.

GSK00008E z/OS PKCS #11 function function-name failed with return code return-code

Explanation: The indicated z/OS PKCS #11 function failed with the reported return code. The return code is displayed in hexadecimal with its decimal value in parenthesis. Information about the function and return code value can be found in [z/OS Cryptographic Services ICSF Writing PKCS #11 Applications]

User response: Follow the information given in the [z/OS Cryptographic Services ICSF Writing PKCS #11 Applications] publication for the reported function and return code. If the problem cannot be resolved, contact your service representative.
## Appendix A. Environment Variables

These tables contain all the environment variables used by the system application and read during the startup of the application.

### Table 8. SSL-Specific Environment Variables

<table>
<thead>
<tr>
<th>Environment Variables</th>
<th>Usage</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSK_CERT_VALIDATION_MODE</td>
<td>Specifies which Internet standard is to be used for certificate validation.</td>
<td>A value of '2459' specifies certificate validation against RFC2459 only. A value of '3280' specifies certificate validation against RFC3280 only. A value of 'ANY' specifies certificate validation against RFC2459 initially - if that fails, validate against RFC3280.</td>
</tr>
<tr>
<td>GSK_CLIENT_AUTH_NOCERT_ALERT</td>
<td>Specifies whether the SSL server application will accept a connection from a client where client authentication has been requested and the client fails to supply an X.509 certificate.</td>
<td>A value of 'OFF' or '0' will allow connections with clients where client authentication has been requested and the client fails to supply an X.509 certificate. A value of 'ON' or '1' will terminate connections with clients where client authentication has been requested and the client fails to supply an X.509 certificate. The default value is 'OFF'.</td>
</tr>
<tr>
<td>GSK_CRL_CACHE_TIMEOUT</td>
<td>Specifies the number of hours that a cached CRL will remain valid.</td>
<td>The valid timeout values are 0 through 720 and defaults to 24. A value of 0 will disable the CRL cache.</td>
</tr>
</tbody>
</table>
| GSK_CRL_SECURITY_LEVEL      | Specifies the level of security SSL applications will use when contacting LDAP servers to check CRLs for revoked certificates during certificate validation. CRLs located will be cached according to the GSK_CRL_CACHE_TIMEOUT setting of the SSL environment. To enforce contact with the LDAP server for each CRL check, CRL caching must be disabled. If a CRL is not defined an empty CRL will be placed in the CRL cache to prevent repeated calls to the LDAP server. This entry will not be cleared until the CRL cache timeout is reached. See “gsk_attribute_set_numeric_value()” on page 62 and Appendix A, “Environment Variables,” on page 481 for additional information on the GSK_CRL_CACHE_TIMEOUT setting. | LOW - Certificate validation will not fail if the LDAP server cannot be contacted.  
MEDIUM - Certificate validation requires the LDAP server to be contactable, but does not require a CRL to be defined. This is the default.  
HIGH - Certificate validation requires the LDAP server to be contactable and a CRL to be defined. |
### Environment Variables

**Table 8. SSL-Specific Environment Variables (continued)***

<table>
<thead>
<tr>
<th>Environment Variables</th>
<th>Usage</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSK_EXC_ABEND_DUMP</td>
<td>Specifies whether the SSL condition handler should call the cdump() service to dump the current thread before resuming the failing routine. The dump will be placed in the current directory unless LE has been instructed to use a different directory by the _CEE_DMPTARG environment variable. See <em>Language Environment for OS/390 &amp; VM Programming Reference</em> for more information on LE callable services.</td>
<td>A value of '1' will enable SSL dumps and a value of '0' will disable SSL dumps. The default is '0'. The export file will contain just the requested certificate when the DER format is selected. The export file will contain the requested certificate and its certification chain when the PKCS #7 format is selected.</td>
</tr>
<tr>
<td>GSK_HW_CRYPTO</td>
<td>Specifies whether the hardware cryptographic support will be used. Note that ICSF (Integrated Cryptographic Service Facility) must be configured and running in order for System SSL to use the hardware cryptographic support. SHA-1, SHA-2, DES, Triple DES and AES hardware functions can be used without ICSF if the zArchitecture message-security assist is installed. For more information on hardware cryptographic support, refer to Chapter 3, &quot;Using Hardware Cryptographic Features with System SSL,&quot; on page 9.</td>
<td>A value of '0' will disable the use of the hardware support while a value of '65535' will enable the use of the hardware support. Available hardware support will be used if this environment variable is not defined.</td>
</tr>
<tr>
<td>GSK_KEY_LABEL</td>
<td>Specifies the label of the key used to authenticate the application. The default key will be used if a key label is not specified.</td>
<td></td>
</tr>
</tbody>
</table>
### Environment Variables

**Table 8. SSL-Specific Environment Variables (continued)**

<table>
<thead>
<tr>
<th>Environment Variables</th>
<th>Usage</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GSK_KEYRING_FILE</strong></td>
<td>Specifies the name of the key database HFS file, SAF key ring or z/OS PKCS #11 token. A key database is used if the GSK_KEYRING_PW or GSK_KEYRING_STASH environment variable is also specified. Otherwise a SAF key ring or z/OS PKCS #11 token is used. Note that certificate private keys are not available when using a SAF key ring owned by another user. The user must have READ access to resource USER.token-name in the CRYPTOZ class.</td>
<td>The SAF key ring name is specified as “userid/keyring”. The current userid is used if the userid is omitted. The z/OS PKCS #11 token name is specified as “TOKEN/token-name”.</td>
</tr>
<tr>
<td><strong>GSK_KEYRING_PW</strong></td>
<td>Specifies the password for the key database.</td>
<td>NULL or value consisting of up to 128 characters.</td>
</tr>
<tr>
<td><strong>GSK_KEYRING_STASH</strong></td>
<td>Specifies the name of the key database password stash file.</td>
<td>The stash file name always has an extension of <code>.sth</code> and the supplied name will be changed if it does not have the correct extension. The GSK_KEYRING_PW environment variable will be used instead of the GSK_KEYRING_STASH environment variable if it is also specified.</td>
</tr>
<tr>
<td><strong>GSK_LDAP_SERVER</strong></td>
<td>Specifies one or more blank-separated LDAP server host names. The LDAP server is used to obtain CA certificates when validating a certificate and the local database does not contain the required certificate. The local database must contain the required certificates if no LDAP server is specified. Even when an LDAP server is used, root CA certificates must be found in the local database since the LDAP server is not a trusted data source. The LDAP server is also used to obtain certificate revocation lists.</td>
<td>Each host name can contain an optional port number separated from the host name by a colon.</td>
</tr>
<tr>
<td><strong>GSK_LDAP_PASSWORD</strong></td>
<td>Specifies the password to use when connecting to the LDAP server.</td>
<td></td>
</tr>
<tr>
<td><strong>GSK_LDAP_PORT</strong></td>
<td>Specifies the LDAP server port. Port 389 will be used no LDAP server port is specified.</td>
<td>Port must be between 1 and 65535.</td>
</tr>
<tr>
<td><strong>GSK_LDAP_USER</strong></td>
<td>Specifies the distinguished name to use when connecting to the LDAP server.</td>
<td></td>
</tr>
<tr>
<td><strong>GSK_PROTOCOL_SSLV2</strong></td>
<td>Specifies whether the SSL V2 protocol is supported. The SSL V2 protocol should be disabled whenever possible since the SSL V3 protocol provides significant security enhancements. This variable has no effect when operating in FIPS mode.</td>
<td>A value of &quot;0&quot;, &quot;OFF&quot; or &quot;DISABLED&quot; will disable the SSL V2 protocol while a value of &quot;1&quot;, &quot;ON&quot; or &quot;ENABLED&quot; will enable the SSL V2 protocol.</td>
</tr>
</tbody>
</table>
### Environment Variables

**Table 8. SSL-Specific Environment Variables (continued)**

<table>
<thead>
<tr>
<th>Environment Variables</th>
<th>Usage</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSK_PROTOCOL_SSLV3</td>
<td>Specifies whether the SSL V3 protocol is supported. This variable has no effect when operating in FIPS mode.</td>
<td>A value of &quot;0&quot;, &quot;OFF&quot; or &quot;DISABLED&quot; will disable the SSL V3 protocol while a value of &quot;1&quot;, &quot;ON&quot; or &quot;ENABLED&quot; will enable the SSL V3 protocol.</td>
</tr>
<tr>
<td>GSK_PROTOCOL_TLSV1</td>
<td>Specifies whether the TLS V1.0 protocol is supported.</td>
<td>A value of &quot;0&quot;, &quot;OFF&quot; or &quot;DISABLED&quot; will disable the TLS V1.0 protocol while a value of &quot;1&quot;, &quot;ON&quot; or &quot;ENABLED&quot; will enable the TLS V1.0 protocol.</td>
</tr>
<tr>
<td>GSK_PROTOCOL_TLSV1_1</td>
<td>Specifies whether the TLS V1.1 protocol is supported.</td>
<td>A value of &quot;0&quot;, &quot;OFF&quot; or &quot;DISABLED&quot; will disable the TLS V1.1 protocol while a value of &quot;1&quot;, &quot;ON&quot; or &quot;ENABLED&quot; will enable the TLS V1.1 protocol.</td>
</tr>
<tr>
<td>GSK_RNG_ALLOW_ZERO_BYTES</td>
<td>Specifies whether the SSL random number generator, gsk_generate_random_bytes will include bytes with a zero value in the random byte output stream, or remove them. The GSK_RNG_ALLOW_ZERO_BYTES environment variable is processed during System SSL initialization and is not checked afterwards.</td>
<td>A value of “TRUE”, “ON” or “1” will set the random number generator to retain bytes with a zero value in the output stream. A value of “FALSE”, “OFF” or “0” will result in bytes with a zero value being removed. The default setting is “TRUE”.</td>
</tr>
<tr>
<td>GSK_SSL_HW_DETECT_MESSAGE</td>
<td>Setting this environment variable to 1 will cause a series of messages to be written to stderr during System SSL initialization. These messages will display the current status of the hardware cryptographic support. These messages are intended for diagnostic use only and are not translated based on the setting of the LANG environment variable.</td>
<td>The output is a series of statements indicating currently available hardware support.</td>
</tr>
<tr>
<td>GSK_SSL_ICSF_ERROR_MESSAGE</td>
<td>Setting this environment variable to 1 will cause a message to be written to stderr when an ICSF function returns an error. These messages are intended for diagnostic use only and are not translated based on the setting of the LANG environment variable.</td>
<td></td>
</tr>
<tr>
<td>GSK_STDERR_FILE</td>
<td>Specifies the fully-qualified name of the file to receive standard error messages generated using SSL message services. Messages displayed from externally documented messages will be written to stderr if this environment variable is not defined.</td>
<td></td>
</tr>
<tr>
<td>Environment Variables</td>
<td>Usage</td>
<td>Valid Values</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GSK_STDOUT_FILE</td>
<td>Specifies the fully-qualified name of the file to receive standard output messages generated using SSL message services. Messages displayed from externally documented messages will be written to stdout if this environment variable is not defined.</td>
<td>A value of &quot;0&quot;, &quot;OFF&quot; or &quot;DISABLED&quot; will disable sysplex session caching while a value of &quot;1&quot;, &quot;ON&quot; or &quot;ENABLED&quot; will enable sysplex session caching.</td>
</tr>
<tr>
<td>GSK_SYSPLEX_SIDCACHE</td>
<td>Specifies whether sysplex session caching is supported for this application.</td>
<td></td>
</tr>
<tr>
<td>GSK_T61_AS_LATIN1</td>
<td>Specifies the character set for ASN.1 TELETEXSTRING conversions. The T.61 character set is supposed to be used for strings tagged as TELETEXSTRING. The X.690 ASN.1 definition specifies the 7-bit T.61 character set (ISO IR-102). However, many certificate authorities issue certificates using the 8-bit ISO8859-1 character set (ISO IR-100) instead of the 7-bit T.61 character set. This causes conversion errors when the certificate is decoded. To add to the confusion, the 8-bit T.61 character set (ISO IR-103) is also used by some implementations.</td>
<td>If the GSK_T61_AS_LATIN1 environment variable is set to YES or 1, the 8-bit ISO8859-1 character set will be used when processing a TELETEX string. If the GSK_T61_AS_LATIN1 environment variable is set to NO or 0, the 8-bit T.61 character set will be used. The default is to use the ISO8859-1 character set. The GSK_T61_AS_LATIN1 environment variable is processed during System SSL initialization and is not checked afterwards. Note that selecting the incorrect character set can cause strings to be converted incorrectly.</td>
</tr>
</tbody>
</table>
| GSK_TRACE                     | Specifies a bit mask enabling System SSL trace options. No trace option is enabled if the bit mask is 0 and all trace options are enabled if the bit mask is 0xffff. The bit mask can be specified as a decimal (nnn), octal (0nnnn) or hexadecimal (0xhh) value. | These trace options are available:  
0x01 = Trace function entry  
0x02 = Trace function exit  
0x04 = Trace errors  
0x08 = Include informational messages  
0x10 = Include EBCDIC data dumps  
0x20 = Include ASCII data dumps                                                                                                                                                           |
### Environment Variables

**Table 8. SSL-Specific Environment Variables (continued)**

<table>
<thead>
<tr>
<th>Environment Variables</th>
<th>Usage</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSK_TRACE_FILE</td>
<td>Specifies the name of the trace file and defaults to /tmp/gskssl.%.trc. The gsktrace command is used to format the trace file. The trace file is not used if the GSK_TRACE environment variable is not defined or is set to 0. The current process identifier is included as part of the trace file name when the name contains a percent sign (%). For example, if GSK_TRACE_FILE is set to /tmp/gskssl.%.trc and the current process identifier is 247, then the trace file name will be /tmp/gskssl.247.trc.</td>
<td>Must be set to the name of an HFS file in a directory for which the executing application has write permission.</td>
</tr>
</tbody>
</table>
| GSK_V2_CIPHER_SPECS         | Specifies the SSL V2 cipher specifications in order of preference as a string consisting of 1 or more 1-character values. These cipher specifications are supported:  
   1 = 128-bit RC4 encryption with MD5 message authentication (128-bit secret key)  
   2 = 128-bit RC4 export encryption with MD5 message authentication (40-bit secret key)  
   3 = 128-bit RC2 encryption with MD5 message authentication (128-bit secret key)  
   4 = 128-bit RC2 export encryption with MD5 message authentication (40-bit secret key)  
   6 = 56-bit DES encryption with MD5 message authentication (56-bit secret key)  
   7 = 168-bit Triple DES encryption with MD5 message authentication (168-bit secret key) | The default is “713642” if security level 3 is installed, “642” otherwise. |
| GSK_V2_SESSION_TIMEOUT      | Specifies the session timeout value in seconds for the SSL V2 protocol. | The valid timeout values are 0 through 100, default value is 100. |
| GSK_V2_SIDCACHE_SIZE        | Specifies the number of session identifiers that can be contained in the SSL V2 cache. | The valid cache sizes are 0 through 32000 and defaults to 256. The SSL V2 cache will be disabled if 0 is specified. The session identifier cache will be allocated using the requested size rounded up to a power of 2 with a minimum size of 16. |
### Table 8. SSL-Specific Environment Variables (continued)

<table>
<thead>
<tr>
<th>Environment Variables</th>
<th>Usage</th>
<th>Valid Values</th>
</tr>
</thead>
</table>
| GSK_V3_CIPHER_SPECS         | Specifies the SSL V3 cipher specifications in order of preference as a string consisting of 1 or more 2-character values. The SSL V3 cipher specifications are used for the SSL V3, TLS V1.0 and TLS V1.1 protocols.         | If executing in non-FIPS mode, the default is “050435363738392F303132330A1613100D0915120F0C0306020100” when security level 3 is installed, ”0915120F0C0306020100” otherwise. If executing in FIPS mode, the default is “35363738392F303132330A1613100D”.

These cipher specifications are supported:

00 = No encryption or message authentication and RSA key exchange  
01 = No encryption with MD5 message authentication and RSA key exchange  
02 = No encryption with SHA-1 message authentication and RSA key exchange  
03 = 40-bit RC4 encryption with MD5 message authentication and RSA key exchange  
04 = 128-bit RC4 encryption with MD5 message authentication and RSA key exchange  
05 = 128-bit RC4 encryption with SHA-1 message authentication and RSA key exchange  
06 = 40-bit RC2 encryption with MD5 message authentication and RSA key exchange  
09 = 56-bit DES encryption with SHA-1 message authentication and RSA key exchange  
0A = 168-bit Triple DES encryption with SHA-1 message authentication and RSA key exchange  
0C = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate  
0D = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate  
0F = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate  
10 = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
### Environment Variables

Table 8. SSL-Specific Environment Variables (continued)

<table>
<thead>
<tr>
<th>Environment Variables</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSK_V3_CIPHER_SPECS (Continued)</td>
<td>Valid Values</td>
</tr>
<tr>
<td>12 = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate</td>
<td></td>
</tr>
<tr>
<td>13 = 168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate</td>
<td></td>
</tr>
<tr>
<td>15 = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate</td>
<td></td>
</tr>
<tr>
<td>16 = 168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate</td>
<td></td>
</tr>
<tr>
<td>2F = 128-bit AES encryption with SHA-1 message authentication and RSA key exchange</td>
<td></td>
</tr>
<tr>
<td>30 = 128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate</td>
<td></td>
</tr>
<tr>
<td>31 = 128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate</td>
<td></td>
</tr>
<tr>
<td>32 = 128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate</td>
<td></td>
</tr>
<tr>
<td>33 = 128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate</td>
<td></td>
</tr>
<tr>
<td>35 = 256-bit AES encryption with SHA-1 message authentication and RSA key exchange</td>
<td></td>
</tr>
<tr>
<td>36 = 256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate</td>
<td></td>
</tr>
<tr>
<td>37 = 256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate</td>
<td></td>
</tr>
</tbody>
</table>

If executing in non-FIPS mode, the default is "050435363738392F303132330A1613100D0915120F0C0306020100" when security level 3 is installed, "0915120F0C0306020100" otherwise. If executing in FIPS mode, the default is "35363738392F303132330A1613100D".

---

488 z/OS V1R11.0 System SSL Programming
### Table 8. SSL-Specific Environment Variables (continued)

<table>
<thead>
<tr>
<th>Environment Variables</th>
<th>Usage</th>
<th>Valid Values</th>
</tr>
</thead>
</table>
| GSK_V3_CIPHER_SPECS (Continued)        | • 38 = 256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate  
• 39 = 256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate  
If using the TLS V1.1 protocol, export ciphers are not supported. The 40 bit ciphers (cipher specifications “03” and “06”) will be ignored if specified. | If executing in non-FIPS mode, the default is “050435363738392F303132330A1613100D0915120F0C0306020100” when security level 3 is installed, “0915120F0C0306020100” otherwise. If executing in FIPS mode, the default is “35363738392F303132330A1613100D0915120F0C0306020100” otherwise. If executing in FIPS mode, the default is “35363738392F303132330A1613100D0915120F0C0306020100” otherwise. If executing in FIPS mode, the default is “35363738392F303132330A1613100D0915120F0C0306020100” otherwise. If executing in FIPS mode, the default is “35363738392F303132330A1613100D0915120F0C0306020100” otherwise. If executing in FIPS mode, the default is “35363738392F303132330A1613100D0915120F0C0306020100” otherwise. | |
| GSK_V3_SESSION_TIMEOUT                 | Specifies the session timeout value in seconds for the SSL V3, TLS V1.0 and TLS V1.1 protocols. | The valid timeout values are 0 through 86400 and defaults to 86400. The timeout will be disabled if 0 is specified. |
| GSK_V3_SIDCACHE_SIZE                  | Specifies the number of session identifiers that can be contained in the SSL V3 cache. The SSL V3 session cache is used for the SSL V3, TLS V1.0 and TLS V1.1 protocols. | The valid cache sizes are 0 through 64000 and defaults to 512. The SSL V3 cache will be disabled if 0 is specified. The session identifier cache will be allocated using the requested size rounded up to a power of 2 with a minimum size of 16. |
| GSKV2CACHESIZE                        | Used to control the size limit for a V2 session cache. This variable is for use only with the deprecated API set. | The valid cache sizes are 0 through 32000 and defaults to 256. |
| GSKV3CACHESIZE                        | Used to control the size limit for a V3 session cache. This variable is for use only with the deprecated API set. | The valid cache sizes are 0 through 64000 and defaults to 512 entries. |

Table 9 on page 490 contains system environment variables used by SSL. For more information, see the topic on shell variables in the [z/OS UNIX System Services Command Reference](https://www.ibm.com/support/knowledgecenter/en/S hitters_zos813/com.ibm.zos.r160.doc/zosman/man/zh humble.html).
### Environment Variables

**Table 9. System Environment Variables used by SSL**

<table>
<thead>
<tr>
<th>System Environment Variables</th>
<th>Usage</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBPATH</td>
<td>Used to specify the directory to search for a DLL (Dynamic Link Library) filename. If it is not set, the working directory is searched.</td>
<td></td>
</tr>
<tr>
<td>NLSPATH</td>
<td>Specifies where the message catalogs are to be found.</td>
<td></td>
</tr>
<tr>
<td>PATH</td>
<td>Contains a list of directories that the system searches to find executable commands. Directories in this list are separated with colons. Searches each directory in the order specified in the list until it finds a matching executable. If you want the shell to search the working directory, put a null string in the list of directories (for example, to tell the shell to search the working directory first, start the list with a colon or semicolon).</td>
<td></td>
</tr>
<tr>
<td>STEPLIB</td>
<td>Identifies a STEPLIB variable to be used in building a process image for running an executable file. A STEPLIB is a set of private libraries used to store a new or test version of an application program, such as a new version of a runtime library.</td>
<td>STEPLIB can be set to the values CURRENT or NONE or to a list of MVS data set names. The default is CURRENT, which passes on the TASKLIB, STEPLIB, or JOBLIB allocations that are part of the invoker's MVS program search order environment to the process image created for an executable file. The value NONE indicates you do not want a STEPLIB environment for executable files. You can specify up to 255 MVS data set names, separated by colons, as a list of data sets used to build a STEPLIB variable.</td>
</tr>
</tbody>
</table>
Appendix B. Sample C++ SSL Files

A sample set of files is shipped to provide an example of what is needed to build a C++ System SSL application. These files build one DLL (SECURES) and two programs: client and server. These sample files are located in /usr/lpp/gskssl/examples:

- Makefile
- client.cpp
- server.cpp
- common.hpp
- common.cpp
- secures.h
- secures.cpp
- utils.hpp
- utils.cpp
- display_certificate.c

Note: Reference the sample source for SSL environment and connection attributes. For example, filename and password attributes are hard coded in the kdb file.

server (source file: server.cpp) is a multithreaded program that opens a socket on IP address 127.0.0.1, port 4321 and listens for client requests. server can run in either secure (using SSL) mode or nonsecure (using normal socket reads and writes) mode. By default, server runs with one socket listen thread and 20 work threads. The socket listen thread listens for connections from clients and puts each request onto the work list. The work threads check the work list for work and then perform the work. The number of work threads can be specified using the -numthreads parameter when starting server.

client (source file: client.cpp) is a single threaded program that connects to the server program and exchanges one or more data packets. client can also run in secure or nonsecure mode, but its mode must match the mode of the server to which it is connecting. The number of connections, the number of read/write packets per connection, the number of bytes in each write packet, and the number of bytes in each read packet can be specified. Multiple clients can be run simultaneously to the same server.

The files included in the examples are:

Makefile
   This file builds the example DLL and programs.

client.cpp
   This file contains the routines that implement the client function.

server.cpp
   This file contains the routines that implement the server function.

common.hpp
   This contains the prototypes and defines for the routines in common.cpp.

common.cpp
   This file contains a set of routines called by client and server to set up, accept, open, and close connections, and to read and write data. All data that is read or written in the form of packets that contain a header containing a command, length, and cookie. This implements a higher level communication protocol used between the client and server programs. For example, this higher level protocol allows the client to send a "STOP" request to the server, which stops the server program.
Sample C++ SSL Files

secures.h
This file contains prototypes and defines for the routines in secures.cpp.

secures.cpp
This file implements a set of APIs that are similar to the normal sockets APIs, except that the routines work in either secure (SSL) or nonsecure mode. These routines are called by code in client.cpp, server.cpp, and common.cpp.

utils.hpp
This file contains the prototype for the routine in utils.cpp, some structure definitions, and several defined constants.

utils.cpp
This file contains routines that server and client programs use to check command line options.

display_certificate.c
This file is a sample program to decode and display an X.509 certificate.

Use the ? option to obtain help information.
Appendix C. Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen-readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size.

Using assistive technologies

Assistive technology products, such as screen-readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using it 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, SA22-7787, z/OS TSO/E User’s Guide, SA22-7794, and z/OS ISPF User’s Guide Volume I, SC34-4822, 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.
Appendix D. Notices

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

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 (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:
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.

Programming Interface Information

z/OS Cryptographic Services System SSL Programming, SC24-5901 primarily documents intended Programming Interfaces that allow the customer to write programs to obtain services of System SSL.

z/OS Cryptographic Services System SSL Programming, SC24-5901 also documents information that is NOT intended to be used as Programming Interfaces of System SSL. This information is identified where it occurs, by an introductory statement to a chapter or section.
Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:
- BookManager
- Domino
- eServer
- IBM
- IBMLink
- MVS
- OS/2
- OS/390
- RACF
- Resource Link
- WorldRegistry
- z9
- z10
- z/OS
- zSeries

Microsoft, Windows, Windows NT and the Windows logo are registered trademarks of Microsoft Corporation.

RC2 and RC4 are registered trademarks of RSA Security, Inc.

Other company, product and service names may be trademarks or service marks of others.
**Bibliography**

This bibliography provides a list of some of the publications that are useful when using the z/OS System SSL support. The complete title, order number, and a brief description is given for each publication.

### z/OS Security Server Publications
- **z/OS Integrated Security Services LDAP Server Administration and Use, SC24-5923**
  This information describes how to install, configure, and run the stand-alone LDAP daemon (SLAPD). It is intended for administrators who will maintain the server and database.
  This information explains RACF concepts and describes how to plan for and implement RACF.
- **z/OS Security Server RACF Command Language Reference, SA22-7687**
  This information describes the functions and syntax of all RACF commands.

### z/OS Cryptographic Services Publications
- **z/OS Cryptographic Services ICSF System Programmer's Guide, SA22-7526**
  This information describes how to initialize, customize, operate, and diagnose the z/OS Integrated Cryptographic Service Facility (ICSF).
- **z/OS Cryptographic Services ICSF Administrator's Guide, SA22-7521**
  This information describes how to manage cryptographic keys by using the z/OS Integrated Cryptographic Service Facility (ICSF), which is part of the z/OS Cryptographic Services.
- **z/OS Cryptographic Services ICSF Overview, SA22-7519**
  This information contains the overview and planning information for the z/OS Integrated Cryptographic Service Facility (ICSF).
- **z/OS Cryptographic Services ICSF Application Programmer's Guide, SA22-7522**
  This information describes how to use the callable services provided by the z/OS Integrated Cryptographic Service Facility (ICSF).
- **z/OS Cryptographic Services ICSF Writing PKCS #11 Applications, SA23-2231**
  This information describes how to write PKCS #11 applications.

### IBM C/C++ Language Publication
- **z/OS XL C/C++ Programming Guide, SC09-4765**
  This information describes how to develop applications in the C/C++ language in z/OS.

### Other IBM z/OS Publications
- **z/OS Information Roadmap, SA22-7500**
  This information lists the complete titles and order numbers of the information for all products that are part of z/OS.
- **z/OS MVS System Messages, Vol 1 (ABA-AOM), SA22-7631**
- **z/OS MVS System Messages, Vol 2 (ARC-ASA), SA22-7632**
- **z/OS MVS System Messages, Vol 3 (ASB-BPX), SA22-7633**
- **z/OS MVS System Messages, Vol 4 (CBD-DMO), SA22-7634**
- **z/OS MVS System Messages, Vol 5 (EDG-GFS), SA22-7635**
- **z/OS MVS System Messages, Vol 6 (GOS-IEA), SA22-7636**
- **z/OS MVS System Messages, Vol 7 (IEB-IEE), SA22-7637**
- **z/OS MVS System Messages, Vol 8 (IEF-IGD), SA22-7638**
- **z/OS MVS System Messages, Vol 9 (IGF-IWM), SA22-7639**
- **z/OS MVS System Messages, Vol 10 (IXC-I2P), SA22-7640**
  These contain messages and their routing and descriptor codes.
- **z/OS Program Directory, GI10-0670**
  The Program Directory contains information about installing the Cryptographic Services base element using SMP/E.
- **z/OS MVS Programming: Assembler Services Reference ABE-HSP, SA22-7606** and **z/OS MVS Programming: Assembler Services Reference IAR-XCT, SA22-7607**
  These describe Assembler services information.
Index

A
accepting a secure socket connection 354
accessibility 493
accessing DLLs 374
APRs
  gsk_attribute_get_buffer() 37
  gsk_attribute_get_cert_info() 40
  gsk_attribute_get_data() 45
  gsk_attribute_get_enum() 47
  gsk_attribute_get_numeric_value() 50
  gsk_attribute_set_buffer() 52
  gsk_attribute_set_callback() 55
  gsk_attribute_set_enum() 59
  gsk_attribute_set_numeric_value() 62
  gsk_attribute_set_tls_extension() 64
  gsk_environment_close() 67
  gsk_environment_init() 68
  gsk_environment_open() 70
  gsk_free_cert_data() 75
  gsk_free_memory() 346
  gsk_get_cert_by_label() 76
  gsk_get_cipher_info() 347
  gsk_get_dn_by_label() 348
  gsk_get_update() 83
  gsk_initialize() 349
  gsk_list_free() 84
  gsk_secure_soc_close() 353
  gsk_secure_soc_init() 354
  gsk_secure_soc_read() 361
  gsk_secure_soc_reset() 363
  gsk_secure_soc_write() 364
  gsk_secure_socket_close() 85
  gsk_secure_socket_init() 86
  gsk_secure_socket_misc() 90
  gsk_secure_socket_open() 92
  gsk_secure_socket_read() 93
  gsk_secure_socket_write() 97
  gsk_unregister() 369
  gsk_user_set() 370
  GSRSRBBD() 367
  GSRSRBW() 368, 428
  using in an System SSL program 5

cipher information
  querying 347
client, authentication certificate selection 26
client, System SSL program 22
compiling an System SSL application 24
component trace support 437
creating
  SSL environment 19

D
diagnostic information 439
disability 493
distinguished name
  returning pointer for 348
DLLs, accessing 374

E
elements of an System SSL program 5
ending secure socket connection 353
environment variables 481
establishing System SSL environment 349
examples
  parts shipped in HFS 3

F
FIPS 140-2 13
FIPS mode
  algorithms and key sizes 13
  application changes 17
  certificate stores 17
  certificates 13
  SAF keyrings and PKCS #11 tokens 17
  SSL started task 18
  SSL/TLS protocol 14
  system setup and requirements 14
FMID
  Cryptographic Services Security Level 3 1
  Cryptographic Services System SSL 1
  Japanese 1

G
gsk_add_record() 105
  gsk_attribute_get_buffer() 37
  gsk_attribute_get_cert_info() 40
  gsk_attribute_get_data() 45
  gsk_attribute_get_enum() 47
  gsk_attribute_get_numeric_value() 50
  gsk_attribute_set_buffer() 52
  gsk_attribute_set_callback() 55
  gsk_attribute_set_enum() 59
  gsk_attribute_set_numeric_value() 62
  gsk_attribute_set_tls_extension() 64
  gsk_change_database_password() 107

B
bibliography 499
building a z/OS System SSL application 19
building an System SSL application 24

C
callback routine for IO 27
certificate
  removing 417
  self-signed, creating 399
  certificate management 373

© Copyright IBM Corp. 1999, 2009
Index

**H**
- handshake process 354
- hardware cryptography failure notification 438
  - header file, gskssl.h
    - See gskssl.h header file 3
  - HFS
    - contents 3
      - parts shipped 3

**I**
- initializing data areas for System SSL 354
- initiating a secure socket connection 354
- installation information 2
- installation PDS and PDSE
  - members of 3
  - name of 1

**K**
- key database file
  - reading 349
  - uninitialize 369
- key management 373
- key ring 373
- keyboard 493

**L**
- LANG environment variable, setting 374

**M**
- managing PKI private keys and certificates 373
- migrating from deprecated SSL interfaces 33

**GSK**
- gsk_name_compare() 276
- gsk_name_to_dn() 277
- gsk_open_database_using_stash_file() 281
- gsk_open_database() 279
- gsk_open_directory() 283
- gsk_open_keyring() 284
- gsk_perform_kat() 286
- gsk_query_crypto_level() 287
- gsk_query_database_label() 288
- gsk_query_database_record_length() 289
- gsk_rdtime() 290
- gsk_read_content_msg() 291
- gsk_read_data_content() 292
- gsk_read_data_msg() 293
- gsk_read_encrypted_data_content() 294
- gsk_read_encrypted_data_msg() 295
- gsk_read_enveloped_data_content_extended() 300
- gsk_read_enveloped_data_content() 298
- gsk_read_enveloped_data_msg_extended() 304
- gsk_read_enveloped_data_msg() 302
- gsk_read_signed_data_content_extended() 308
- gsk_read_signed_data_content() 306
- gsk_read_signed_data_msg_extended() 314
- gsk_read_signed_data_msg() 311
- gsk_read_wrapped_content() 317
- gsk_receive_certificate() 318
- gsk_replace_record() 319
- gsk_secure_soc_close API 353
- gsk_secure_soc_init() API 354
- gsk_secure_soc_read() API 361
- gsk_secure_soc_reset() API 363
- gsk_secure_soc_write() API 364
- gsk_secure_socket_close() 85
- gsk_secure_socket_init() 86
- gsk_secure_socket_misc() 90
- gsk_secure_socket_open() 92
- gsk_secure_socket_read() 93
- gsk_secure_socket_write() 97
- gsk_set_default_key() 321
- gsk_set_directory_enum() 322
- gsk_sign_certificate() 323
- gsk_sign_crl() 325
- gsk_sign_data() 327
- gsk_soc_init_data_structure 354
- gsk_uninitialize() API 369
- gsk_user_set() API 370
- gsk_validate_certificate_mode() 332
- gsk_validate_certificate() 329
- gsk_validate_hostname() 336
- gsk_validate_server() 338
- gsk_verify_certificate_signature() 339
- gsk_verify_crl_signature() 341
- gsk_verify_data_signature() 343
- gskkyman utility (continued)
  - HFS location 3
    - key database files 374
    - key management menu 379
    - overview 373
    - private key
      - removing 417
    - setting LANG environment variable 374
    - setting NLSPATH environment variable 374
    - setting PATH environment variable 374
    - setting STEPLIB environment variable 374
    - setting up the environment 374
    - token management menu 379
    - using 373
    - z/OS PKCS #11 tokens 375
  - GSJSRBRD() 367
  - GSJSRWT() 368, 428
- gskssl.h header file
  - gsk_soc_init_data_structure 354
  - HFS location 3
setting (continued)
- LANG environment variable 374
- NLSPATH environment variable 374
- PATH environment variable 374
- STEPLIB environment variable 374

shortcut keys 493

software dependencies 1

SSL (Secure Sockets Layer)
  description 1

SSL environment
  creating 19

SSL System
  callback routine for IO 27

STEPLIB environment variable, setting 374

structure
  gsk_soc_init_data 354

structure of an System SSL program 5

System SSL
  APIs 35
  client authentication certificate selection 26
  elements of a program 5
  environment variables 481
  establishing environment 349
  FIPS 140-2 13
  how it works 5
  migrating 33
  obtaining trace information 439
  parts shipped in HFS 3
  parts shipped in PDS and PDSE 3
  refreshing security parameters 363
  removing settings for the environment 369
  session ID (SID) cache replacement 28
  using hardware cryptographic features 9

System SSL application
  building 24
  overview 6
  writing a client program 22
  writing a server program 21
  writing a source program 19
  writing and building 19

U

using hardware cryptographic features with System SSL 9

W

writing
  system SSL client program 22
  system SSL server program 21
  system SSL source program 19
  z/OS System SSL application 19
Readers’ Comments — We’d Like to Hear from You

z/OS
Cryptographic Services
System Secure Sockets Layer
Programming

Publication No. SC24-5901-08

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.
Submit your comments using one of these channels:
• Send your comments to the address on the reverse side of this form.
• Send your comments via e-mail to: mhvrdfs@us.ibm.com

If you would like a response from IBM, please fill in the following information:

Name

Address

Company or Organization

Phone No.

E-mail address