

z/OS



Cryptographic Services System Secure Sockets Layer Programming

Version 2 Release 2

Note

Before using this information and the product it supports, read the information in "Notices" on page 703.

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

© **Copyright IBM Corporation 1999, 2015.**

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

Contents

Figures vii

Tables ix

About this document xi

Intended audience xi

How this information is organized xi

Conventions used in this information xii

Where to find more information xiii

Internet sources xiii

How to send your comments to IBM xv

If you have a technical problem xv

Summary of changes xvii

Summary of changes for z/OS Version 2 Release 2 (V2R2) xvii

Summary of changes for z/OS Version 2 Release 1 xx

Chapter 1. Introduction 1

Software dependencies 1

Installation information 2

System SSL parts shipped in the UNIX System

Services file system 2

System SSL parts shipped in PDS and PDSE 2

Chapter 2. How System SSL works for secure socket communication 5

Using System SSL on z/OS 6

System SSL application overview 6

Chapter 3. Using cryptographic features with System SSL 11

Guidelines for using hardware cryptographic features 11

Overview of hardware cryptographic features and System SSL 12

Random byte generation support 14

Elliptic Curve Cryptography support 14

Diffie-Hellman key agreement 16

RACF CSFSERV resource requirements 16

PKCS #11 and setting CLEARKEY resource within CRYPTOZ class 18

PKCS #11 Cryptographic operations using ICSF handles 18

Chapter 4. System SSL and FIPS 140-2 19

Algorithms and key sizes 19

Random byte generation 20

Diffie-Hellman key agreement 20

Certificates 21

SSL/TLS protocol 21

System SSL module verification setup 22

Performance guideline 24

Certificate stores 25

SAF key rings and PKCS #11 tokens 25

PKCS #12 files 25

Application changes 26

SSL started task 26

Sysplex session ID cache 27

Chapter 5. Writing and building a z/OS System SSL application 29

Writing a System SSL source program 29

Create an SSL environment 29

System SSL server program 31

System SSL client program 33

Building a z/OS System SSL application 34

Running a z/OS System SSL application 35

System SSL application programming considerations 35

Non-Blocking I/O 36

Client authentication certificate selection 38

I/O routine replacement 39

Use of user data 39

Session ID (SID) cache 40

Session renegotiation notification 42

TLS extensions 42

Suite B cryptography support 45

SSL/TLS partner certificate revocation checking 46

Chapter 6. Migrating from deprecated SSL interfaces 55

Chapter 7. API reference 57

`gsk_attribute_get_buffer()` 60

`gsk_attribute_get_cert_info()` 65

`gsk_attribute_get_data()` 70

`gsk_attribute_get_enum()` 72

`gsk_attribute_get_numeric_value()` 79

`gsk_attribute_set_buffer()` 82

`gsk_attribute_set_callback()` 87

`gsk_attribute_set_enum()` 92

`gsk_attribute_set_numeric_value()` 102

`gsk_attribute_set_tls_extension()` 108

`gsk_environment_close()` 111

`gsk_environment_init()` 112

`gsk_environment_open()` 114

`gsk_free_cert_data()` 121

`gsk_get_all_cipher_suites()` 122

`gsk_get_cert_by_label()` 123

`gsk_get_cipher_suites()` 128

`gsk_get_ssl_vector()` 129

`gsk_get_update()` 130

`gsk_list_free()` 131

`gsk_secure_socket_close()` 132

`gsk_secure_socket_init()` 133

`gsk_secure_socket_misc()` 142

gsk_secure_socket_open()	144
gsk_secure_socket_read()	145
gsk_secure_socket_shutdown()	148
gsk_secure_socket_write()	150
gsk_strerror()	152

Chapter 8. Certificate Management Services (CMS) API reference 153

gsk_add_record()	159
gsk_change_database_password()	162
gsk_change_database_record_length()	164
gsk_close_database()	165
gsk_close_directory()	166
gsk_construct_certificate()	167
gsk_construct_private_key()	171
gsk_construct_private_key_rsa()	173
gsk_construct_public_key()	175
gsk_construct_public_key_rsa()	177
gsk_construct_renewal_request()	178
gsk_construct_self_signed_certificate()	181
gsk_construct_signed_certificate()	184
gsk_copy_attributes_signers()	188
gsk_copy_buffer()	189
gsk_copy_certificate()	190
gsk_copy_certificate_extension()	191
gsk_copy_certification_request()	192
gsk_copy_content_info()	193
gsk_copy_crl()	194
gsk_copy_name()	195
gsk_copy_private_key_info()	196
gsk_copy_public_key_info()	197
gsk_copy_record()	198
gsk_create_certification_request()	199
gsk_create_database()	203
gsk_create_database_renewal_request()	205
gsk_create_database_signed_certificate()	208
gsk_create_renewal_request()	214
gsk_create_revocation_source()	216
gsk_create_self_signed_certificate()	222
gsk_create_signed_certificate()	226
gsk_create_signed_certificate_record()	229
gsk_create_signed_certificate_set()	234
gsk_create_signed_crl()	239
gsk_create_signed_crl_record()	242
gsk_decode_base64()	246
gsk_decode_certificate()	247
gsk_decode_certificate_extension()	248
gsk_decode_certification_request()	250
gsk_decode_crl()	251
gsk_decode_import_certificate()	252
gsk_decode_import_key()	253
gsk_decode_issuer_and_serial_number()	255
gsk_decode_name()	256
gsk_decode_private key()	257
gsk_decode_public key()	258
gsk_decode_signer_identifier()	259
gsk_delete_record()	260
gsk_dn_to_name()	261
gsk_encode_base64()	264
gsk_encode_certificate_extension()	265
gsk_encode_ec_parameters()	267

gsk_encode_export_certificate()	268
gsk_encode_export_key()	270
gsk_encode_export_request()	273
gsk_encode_issuer_and_serial_number()	274
gsk_encode_name()	275
gsk_encode_private_key()	276
gsk_encode_public_key()	277
gsk_encode_signature()	278
gsk_encode_signer_identifier()	279
gsk_export_certificate()	280
gsk_export_certification_request()	282
gsk_export_key()	284
gsk_factor_private_key()	287
gsk_factor_private_key_rsa()	288
gsk_factor_public_key()	289
gsk_factor_public_key_rsa()	290
gsk_fips_state_query()	291
gsk_fips_state_set()	292
gsk_free_attributes_signers()	294
gsk_free_buffer()	295
gsk_free_certificate()	296
gsk_free_certificates()	297
gsk_free_certificate_extension()	298
gsk_free_certification_request()	299
gsk_free_content_info()	300
gsk_free_crl()	301
gsk_free_crls()	302
gsk_free_decoded_extension()	303
gsk_free_issuer_and_serial_number()	304
gsk_free_name()	305
gsk_free_oid()	306
gsk_free_private_key()	307
gsk_free_private_key_info()	308
gsk_free_public_key()	309
gsk_free_public_key_info()	310
gsk_free_record()	311
gsk_free_records()	312
gsk_free_revocation_source()	313
gsk_free_signer_identifier()	314
gsk_free_string()	315
gsk_free_strings()	316
gsk_generate_key_agreement_pair()	317
gsk_generate_key_pair()	319
gsk_generate_key_parameters()	322
gsk_generate_random_bytes()	324
gsk_generate_secret()	325
gsk_get_certificate_algorithms()	326
gsk_get_certificate_info()	327
gsk_get_cms_vector()	329
gsk_get_content_type_and_cms_version()	331
gsk_get_default_key()	332
gsk_get_default_label()	333
gsk_get_directory_certificates()	334
gsk_get_directory_crls()	336
gsk_get_directory_enum()	339
gsk_get_directory_numeric_value()	341
gsk_get_ec_parameters_info()	342
gsk_get_record_by_id()	343
gsk_get_record_by_index()	344
gsk_get_record_by_label()	345
gsk_get_record_by_subject()	346

gsk_get_record_labels()	347
gsk_get_update_code()	348
gsk_import_certificate()	349
gsk_import_key()	352
gsk_make_content_msg()	355
gsk_make_data_content()	356
gsk_make_data_msg()	357
gsk_make_encrypted_data_content()	358
gsk_make_encrypted_data_msg()	360
gsk_make_enveloped_data_content()	362
gsk_make_enveloped_data_content_extended()	365
gsk_make_enveloped_data_msg()	368
gsk_make_enveloped_data_msg_extended()	371
gsk_make_enveloped_private_key_msg()	374
gsk_make_signed_data_content()	377
gsk_make_signed_data_content_extended()	380
gsk_make_signed_data_msg()	384
gsk_make_signed_data_msg_extended()	387
gsk_make_wrapped_content()	391
gsk_mktime()	392
gsk_modify_pkcs11_key_label()	393
gsk_name_compare()	395
gsk_name_to_dn()	396
gsk_open_database()	398
gsk_open_database_using_stash_file()	400
gsk_open_directory()	402
gsk_open_keyring()	403
gsk_perform_kat()	405
gsk_query_crypto_level()	406
gsk_query_database_label()	407
gsk_query_database_record_length()	408
gsk_rdttime()	409
gsk_read_content_msg()	410
gsk_read_data_content()	411
gsk_read_data_msg()	412
gsk_read_encrypted_data_content()	413
gsk_read_encrypted_data_msg()	415
gsk_read_enveloped_data_content()	417
gsk_read_enveloped_data_content_extended()	419
gsk_read_enveloped_data_msg()	421
gsk_read_enveloped_data_msg_extended()	423
gsk_read_signed_data_content()	425
gsk_read_signed_data_content_extended()	428
gsk_read_signed_data_msg()	431
gsk_read_signed_data_msg_extended()	434
gsk_read_wrapped_content()	438
gsk_receive_certificate()	439
gsk_replace_record()	440
gsk_set_default_key()	443
gsk_set_directory_enum()	445
gsk_set_directory_numeric_value()	447
gsk_sign_certificate()	449
gsk_sign_crl()	452
gsk_sign_data()	455
gsk_validate_certificate()	458
gsk_validate_certificate_mode()	464
gsk_validate_extended_key_usage()	472
gsk_validate_hostname()	474
gsk_validate_server()	476
gsk_verify_certificate_signature()	477
gsk_verify_crl_signature()	480

gsk_verify_data_signature()	483
-----------------------------	-----

Chapter 9. Deprecated Secure Socket Layer (SSL) APIs 487

gsk_free_memory()	488
gsk_get_cipher_info()	489
gsk_get_dn_by_label()	490
gsk_initialize()	491
gsk_secure_soc_close()	496
gsk_secure_soc_init()	497
gsk_secure_soc_read()	505
gsk_secure_soc_reset()	508
gsk_secure_soc_write()	509
gsk_srb_initialize()	511
GSKSRBRD	513
GSKSRBWT	514
gsk_uninitialize()	515
gsk_user_set()	516

Chapter 10. Certificate/Key management. 519

Introduction	519
x.509 certificate revocation	520
gskkyman Overview	521
Setting up the environment to run gskkyman.	522
Key database files	523
z/OS PKCS #11 tokens	523
gskkyman interactive mode descriptions	524
Database menu	524
Key/Token management	527
gskkyman interactive mode examples	536
Starting gskkyman	537
Creating, opening, and deleting a key database file	538
Changing a key database password	541
Storing an encrypted key database password	542
Creating, opening, and deleting a z/OS PKCS #11 token	543
Creating a self-signed server or client certificate	547
Creating a certificate request	550
Sending the certificate request	553
Receiving the signed certificate or renewal certificate	553
Managing keys and certificates	554
Importing a certificate from a file as a trusted CA certificate	571
Importing a certificate from a file with its private key	573
Using gskkyman to be your own certificate authority (CA)	574
Migrating from key database files to z/OS PKCS #11 token	577
Migrating key database files to RACF key rings	577
gskkyman command line mode syntax	577
gskkyman	577
gskkyman command line mode examples	580
gskkyman command line mode displays	582

Chapter 11. SSL started task 587

GSKSRVR environment variables	587
-------------------------------	-----

Configuring the SSL started task	588
Server operator commands	589
Sysplex session cache support	590
Component trace support	590
Hardware cryptography failure notification	590

Chapter 12. Obtaining diagnostic information 591

Obtaining System SSL trace information	591
Capturing trace data through environment variables	591
Component trace support	592
Capturing component trace data	592
Displaying the trace data	594
Event trace records for System SSL	594
Capturing component trace data without an external writer	596

Chapter 13. Messages and codes . . . 599

SSL function return codes	599
Deprecated SSL function return codes	618
ASN.1 status codes (014CExxx)	629
CMS status codes (03353xxx)	633
SSL started task messages (GSK01nnn)	656

Utility messages (GSK00nnn)	665
---------------------------------------	-----

Appendix A. Environment variables 667

Appendix B. Sample C++ SSL files 685

Appendix C. Cipher suite definitions 687

Appendix D. Object identifiers 697

Appendix E. Accessibility 699

Accessibility features	699
Consult assistive technologies	699
Keyboard navigation of the user interface	699
Dotted decimal syntax diagrams	699

Notices 703

Policy for unsupported hardware.	704
Minimum supported hardware	705
Trademarks	705

Index 707

Figures

1. Sockets programming model using System SSL	9
2. Database menu	525
3. Key Management Menu	528
4. Token Management Menu	528
5. Key and Certificate Menus	528
6. Token Key and Certificate Menu	528
7. Certificate Menu	531
8. Token Certificate Menus	531
9. Request Menu	532
10. Token Certificate Request Menu	532
11. Starting Menu for gskkyman	538
12. Creating a New Key Database	538
13. Key Management Menu for gskkyman	539
14. Opening an Existing Key Database File	540
15. Key Management Menu	540
16. Deleting an Existing Key Database	541
17. Changing a Key Database Password	542
18. Key Management Menu	543
19. Creating a New z/OS PKCS #11 Token	543
20. Opening a z/OS PKCS #11 Token from token name	544
21. Opening a z/OS PKCS #11 Token from token list	545
22. Token Management Menu	545
23. Deleting an existing z/OS PKCS #11 Token	546
24. Deleting an existing z/OS PKCS #11 Token	546
25. Creating a Self-Signed Certificate-Key Management Menu	548
26. Creating a Self-Signed Certificate-Token Management Menu	548
27. Creating a Self-Signed Certificate	549
28. Creating a certificate request-Key Management Menu	550
29. Creating a certificate request-Token Management Menu	550
30. Creating a Certificate Request	551
31. Specifying subject alternate names	552
32. Contents of certreq.arm after Certificate Request Generation	553
33. Receiving a Certificate Issued for your Request-Key Management Menu	554
34. Receiving a Certificate Issued for your Request-Token Management Menu	554
35. Key and Certificate List	555
36. Token Key and Certificate List	555
37. Key and Certificate Menu	555
38. Token Key and Certificate Menu	555
39. Certificate Information	556
40. Certificate extensions list.	556
41. Key usage information	557
42. Key information menu	557
43. Token key information menu of a certificate with a secure private key	557
44. Token key information menu of a certificate with a clear private key	558
45. Marking a certificate (and private key) as the default certificate-Key and Certificate Menu	558
46. Marking a certificate (and private key) as the default certificate-Token Key and Certificate Menu	558
47. Copying a Certificate Without its Private Key	559
48. Copying a Certificate and Private key to a Different Key Database-Export File Format.	560
49. Copying a Certificate and Private key to a Different Key Database-Export File Format.	560
50. Copying a Certificate with its Private Key to a Key Database on the Same System.	561
51. Copying a Certificate with its Private Key to a z/OS PKCS #11 Token on the Same System	562
52. Delete Certificate and Key-Key and Certificate Menu	563
53. Delete Certificate and Key-Token Key and Certificate Menu	563
54. Changing a Certificate Label-Key and Certificate Menu	563
55. Changing a Certificate Label-Token and Certificate Menu	563
56. Select 10 to Create a Signed Certificate and Key-Key and Certificate Menu	564
57. Select 10 to Create a Signed Certificate and Key-Token Key and Certificate Menu	564
58. Enter Certificate Details	565
59. Subject Alternate Name Type	566
60. Selecting the ECC Key Type.	567
61. Selecting the ECC Curve Type	568
62. Creating a key parameter file to be used with Diffie-Hellman	569
63. Creating a certificate to be used with Diffie_Hellman	570
64. Select 11 to Create a Certificate Renewal Request-Key and Certificate Menu	571
65. Select 11 to Create a Certificate Renewal Request-Token Key and Certificate Menu	571
66. Certificate List (part 1)	572
67. Certificate List (part 2)	572
68. Certificate List (part 3)	573
69. Importing a Certificate from a File-Key Management Menu	573
70. Importing a Certificate from a File-Token Management Menu	573
71. Importing a Certificate and Private Key from a File-Key Management Menu	574
72. Importing a Certificate and Private Key from a File-Token Management Menu	574

Tables

1. Hardware cryptographic functions used by System SSL	13	15. SAF access levels	524
2. Recommended digest sizes for ECDSA signature key sizes	15	16. SSL-Specific environment variables	667
3. Default EC named curves for specified key sizes	15	17. System environment variables used by SSL	683
4. CSFSERV resources required for hardware support through ICSF callable services	17	18. Cipher suite definitions for SSL V2	687
5. CSFSERV resources required for ICSF PKCS #11 callable services support	17	19. 2-character and 4-character cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2	687
6. Algorithm support: FIPS and non-FIPS	19	20. Cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 by supported protocol, symmetric algorithm, and message authentication algorithm	691
7. Server communicating with clients by way of a socket	36	21. Cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 by key-exchange method and signing certificate	693
8. Using the select() routine	37	22. Supported elliptic curve definitions for TLS V1.0, TLS V1.1, and TLS V1.2	695
9. Suite B supported cipher suites	45	23. Signature algorithm pair definitions for TLS V1.2 and OCSP request signing	695
10. Supported curves	45	24. System SSL supported object identifiers (OIDs)	697
11. gskdb_extended_directory_source parameters	217		
12. gskdb_ocsp_source structure parameters	218		
13. gskdb_ocsp_source structure parameters	219		
14. DN attribute names	262		

About this document

This information supports z/OS® (5650-ZOS) and contains information about Cryptographic Services Integrated Cryptographic Service Facility.

This document 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, TLS V1.1, and TLS V1.2) which can be used by C/C++ applications to communicate securely across an open communications network. The other set of APIs (Certificate Management) provide the ability to use 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.

Intended audience

This document is intended to assist system administrators in setting up the system to use System SSL support and for 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 cryptographic features with System SSL," on page 11 describes System SSLs use of cryptographic features on z/OS.

Chapter 4, "System SSL and FIPS 140-2," on page 19 describes how to execute System SSL securely in a mode designed to meet FIPS 140-2 criteria.

Chapter 5, "Writing and building a z/OS System SSL application," on page 29 describes how to write a System SSL source program and build the System SSL application.

Chapter 6, "Migrating from deprecated SSL interfaces," on page 55 describes how to migrate an existing application which uses the deprecated SSL interfaces to the latest SSL interfaces.

Preface

Chapter 7, “API reference,” on page 57 describes the System SSL program interfaces.

Chapter 8, “Certificate Management Services (CMS) API reference,” on page 153 describes the Certificate Management Services (CMS) program interfaces.

Chapter 9, “Deprecated Secure Socket Layer (SSL) APIs,” on page 487 describes the deprecated System SSL program interfaces.

Chapter 10, “Certificate/Key management,” on page 519 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 587 provides sysplex session cache support and dynamic trace support.

Chapter 12, “Obtaining diagnostic information,” on page 591 provides debugging information.

Chapter 13, “Messages and codes,” on page 599 contains various messages and codes you might encounter using System SSL.

Appendix A, “Environment variables,” on page 667 lists the environment variables used by System SSL.

Appendix B, “Sample C++ SSL files,” on page 685 describes the sample set of files shipped to provide an example of what is needed to build a C++ System SSL application.

Appendix C, “Cipher suite definitions,” on page 687 describes supported cipher suite definitions.

Appendix D, “Object identifiers,” on page 697 describes object identifiers (OIDs) supported by System SSL.

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>**.

Where to find more information

When possible, this information uses cross-document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS V2R2 Information Roadmap*.

To find the complete z/OS library, including the z/OS Information Center, see z/OS Internet Library (<http://www.ibm.com/systems/z/os/zos/bkserv/>).

Internet sources

The following resources are available through the internet to provide additional information about the z/OS library and other security-related topics:

- **Online library**

To view and print online versions of the z/OS publications, use this address:

<http://www.ibm.com/systems/z/os/zos/bkserv/>

- **Redbooks®**

The documents known as IBM® Redbooks that are produced by the International Technical Support Organization (ITSO) are available at the following address:

<http://www.redbooks.ibm.com>

Preface

How to send your comments to IBM

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

Use one of the following methods to send your comments:

1. Send an email to mhvrcfs@us.ibm.com.
2. Send an email from the "Contact us" web page for z/OS (<http://www.ibm.com/systems/z/os/zos/webqs.html>).

Include the following information:

- Your name and address.
- Your email address.
- Your telephone or fax number.
- The publication title and order number:
z/OS V2R2 System SSL Programming
SC14-7495-01
- The topic and page number that is related to your comment.
- The text of your comment.

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

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

If you have a technical problem

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

- Contact your IBM service representative.
- Call IBM technical support.
- Visit the IBM Support Portal at z/OS Support Portal (<http://www-947.ibm.com/systems/support/z/zos/>).

Summary of changes

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

Summary of changes for z/OS Version 2 Release 2 (V2R2)

The following changes are made for z/OS Version 2 Release 2 (V2R2).

New

- The following Certificate Management Services (CMS) APIs are added:
 - “gsk_create_revocation_source()” on page 216
 - “gsk_decode_issuer_and_serial_number()” on page 255
 - “gsk_decode_signer_identifier()” on page 259
 - “gsk_encode_issuer_and_serial_number()” on page 274
 - “gsk_encode_signer_identifier()” on page 279
 - “gsk_free_issuer_and_serial_number()” on page 304
 - “gsk_free_oid()” on page 306
 - “gsk_free_revocation_source()” on page 313
 - “gsk_free_signer_identifier()” on page 314
 - “gsk_get_content_type_and_cms_version()” on page 331
 - “gsk_get_directory_numeric_value()” on page 341
 - “gsk_set_directory_numeric_value()” on page 447
 - “gsk_validate_extended_key_usage()” on page 472
- “x.509 certificate revocation” on page 520 is added.
- The following SSL function return codes are added (See “SSL function return codes” on page 599):
 - 473
 - 474
 - 475
 - 476
 - 477
 - 478
 - 479
 - 480
 - 481
 - 482
 - 484
 - 485
 - 486
 - 487
 - 488
 - 489

- 491
- 492
- 493
- 494
- 495
- The following deprecated SSL function return codes are added (See “Deprecated SSL function return codes” on page 618):
 - -112
 - -125
- The following CMS status codes (03353xxx) are added (See “CMS status codes (03353xxx)” on page 633):
 - 03353094
 - 03353095
 - 03353096
 - 03353097
 - 03353098
 - 03353099
 - 0335309A
 - 0335309B
 - 0335309C
 - 0335309D
 - 0335309E
 - 0335309F
 - 033530A0
 - 033530A1
 - 033530A2
 - 033530A3
 - 033530A4
 - 033530A5
 - 033530A6
 - 033530A7
 - 033530A8
 - 033530AA
 - 033530AB
 - 033530AC
 - 033530AD
 - 033530AE
 - 033530AF
 - 033530B0
 - 033530B1
 - 033530B2
- New environment variables have been added to Appendix A, “Environment variables,” on page 667.

Changed

- The following SSL APIs are modified:
 - “gsk_attribute_get_buffer()” on page 60
 - “gsk_attribute_get_enum()” on page 72
 - “gsk_attribute_get_numeric_value()” on page 79
 - “gsk_attribute_set_buffer()” on page 82
 - “gsk_attribute_set_enum()” on page 92
 - “gsk_attribute_set_numeric_value()” on page 102
 - “gsk_environment_open()” on page 114
 - “gsk_secure_socket_init()” on page 133
 - “gsk_secure_socket_misc()” on page 142
- The following Certificate Management Services (CMS) APIs are modified:
 - “gsk_decode_certificate_extension()” on page 248
 - “gsk_encode_certificate_extension()” on page 265
 - “gsk_get_cms_vector()” on page 329
 - “gsk_get_directory_certificates()” on page 334
 - “gsk_get_directory_crls()” on page 336
 - “gsk_get_directory_enum()” on page 339
 - “gsk_make_enveloped_data_content()” on page 362
 - “gsk_make_enveloped_data_content_extended()” on page 365
 - “gsk_make_enveloped_data_msg()” on page 368
 - “gsk_make_enveloped_data_msg_extended()” on page 371
 - “gsk_make_signed_data_content()” on page 377
 - “gsk_make_signed_data_content_extended()” on page 380
 - “gsk_make_signed_data_msg()” on page 384
 - “gsk_make_signed_data_msg_extended()” on page 387
 - “gsk_open_directory()” on page 402
 - “gsk_set_directory_enum()” on page 445
 - “gsk_validate_certificate()” on page 458
 - “gsk_validate_certificate_mode()” on page 464
- The following Deprecated Secure Socket Layer (SSL) APIs are modified:
 - “gsk_initialize()” on page 491
 - “gsk_secure_soc_init()” on page 497
- The following SSL function return codes are modified (See “SSL function return codes” on page 599):
 - 402
 - 436
- The following CMS status codes (03353xxx) are modified (See “CMS status codes (03353xxx)” on page 633):
 - 03353060
 - 0335306F

Deleted

No content was removed from this information.

Summary of changes for z/OS Version 2 Release 1

See the following publications for all enhancements to z/OS Version 2 Release 1 (V2R1):

- *z/OS V2R2 Migration*
- *z/OS Planning for Installation*
- *z/OS Summary of Message and Interface Changes*
- *z/OS V2R2 Introduction and Release Guide*

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 including server and client authentication that is 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, TLS V1.1, and TLS V1.2 protocols. TLS V1.2 is the latest version of the secure sockets layer protocol that is supported by System SSL.

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 that are required for applications to establish secure sockets communications.

In addition to providing the API interfaces to use 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, use certificates that are 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**. The **gskkyman** utility allows for the management of certificates that are stored in a key database file or z/OS PKCS #11 token.

System SSL is designed to meet the Federal Information Processing Standard - FIPS 140-2 criteria. See Chapter 4, "System SSL and FIPS 140-2," on page 19 for more information.

Software dependencies

Cryptographic Services System SSL (FMID HCPT420)

System SSL Version 2 Release 2 is part of the Cryptographic Services Base element of z/OS. (The System SSL Base members are installed in the PDSE *pdsname.SIEALNKE* and the PDS *pdsname.SGSKSAMP*.)

Cryptographic Services Security Level 3 (FMID JCPT421)

When you order the Cryptographic Services Security Level 3 support, GSKSUS31, GSKSUS64, GSKC31F, GSKC64F, GSKS31F, and GSKS64F are installed as members of the *pdsname.SIEALNKE* PDSE. *pdsname.SIEALNKE* is the PDSE in which the System SSL Cryptographic Services Base members are installed.

Japanese (FMID JCPT42J)

Contains Japanese message text files for the **gskkyman** utility. The **gskmsgs.cat** file is installed in the `/usr/lpp/gskssl/lib/nls/msg/Ja_JP.IBM-939` directory.

Appendix C, “Cipher suite definitions,” on page 687 provides information about the encryption capabilities (ciphers) for each protocol and FMID.

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 Version 2 Release 2 Server Pack, you do not need to install the System SSL Cryptographic Services Base element separately. If you choose the z/OS Custom-Build Product Delivery Offering (CBPDO), you can install the System SSL Cryptographic Services Base element using SMP/E. The *z/OS Program Directory* contains the directions for installing the System SSL Cryptographic Services Base element using SMP/E.

The System SSL base FMID HCPT420 provides cryptographic support up to 56-bit in strength. In order to exploit cryptographic support greater than 56-bit (for example, 3DES, AES), the System SSL Security Level 3 FMID JCPT421 must be ordered and installed. For more information about installing the Security Level 3 FMID, see the *z/OS Program Directory*.

System SSL parts shipped in the UNIX System Services file system

- `/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 including 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

pdsename.**SIEALNKE** PDSE contains members **GSKSSL**, **GSKCMS31**, **GSKSRBRD**, **GSKSRBWT**, **GSKKYMAN**, **GSKSCTSS**, **GSKSRVR**, **GSKCMS64**, **GSKS31**, **GSKS64**, **GSKC31**, **GSKC64** and **GSKSSL64** when the base FMID

HCPT420 is installed. When FMID JCPT421 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 HCPT420 is installed.

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

pdsname.**SIEAMIGE** PDS contains member **GSKSCTFT**.

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

Note:

1. The DLLs are shipped in PDSE form so the DLLs can be called from UNIX System Services file-system-based or 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 data sets are not supported in LPALSTxx. The DLLs can be added to the dynamic LPA by adding them to a PROGxx member.
3. The DLLs cannot be added to the LPA if System SSL is to be used in FIPS mode.

System SSL is designed to meet the National Institute of Standards and Technology (NIST) FIPS 140-2 criteria. For more information about enabling applications and running System SSL FIPS enabled applications, see Chapter 4, "System SSL and FIPS 140-2," on page 19.

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 that is 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 are 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 are signed by a trusted certificate authority (CA) that is known to the server application.

An encrypted SSL connection requires all information that is 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 that is 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 that are 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.

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 9 describes the basic structure of the elements that are 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 that are 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 set all variables, issue this function call to complete the initialization of the SSL environment. When complete, you can open and close SSL connections.

Now, 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. When 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.

gsk_attribute_set...()

This function call sets 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 that are used for the SSL connection.

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

How System SSL works

gsk_attribute_set...()

This function call sets 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 that are 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 complete, 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: **skread()** and **skwrite()** 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.

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 is a z/OS UNIX System Services file that is built and managed using the **gskkyman** utility. In addition to key database files, a PKCS #12 file, a SAF key ring or a z/OS PKCS #11 token can be utilized for certificate information. For more information about key databases, see Chapter 10, "Certificate/Key management," on page 519.

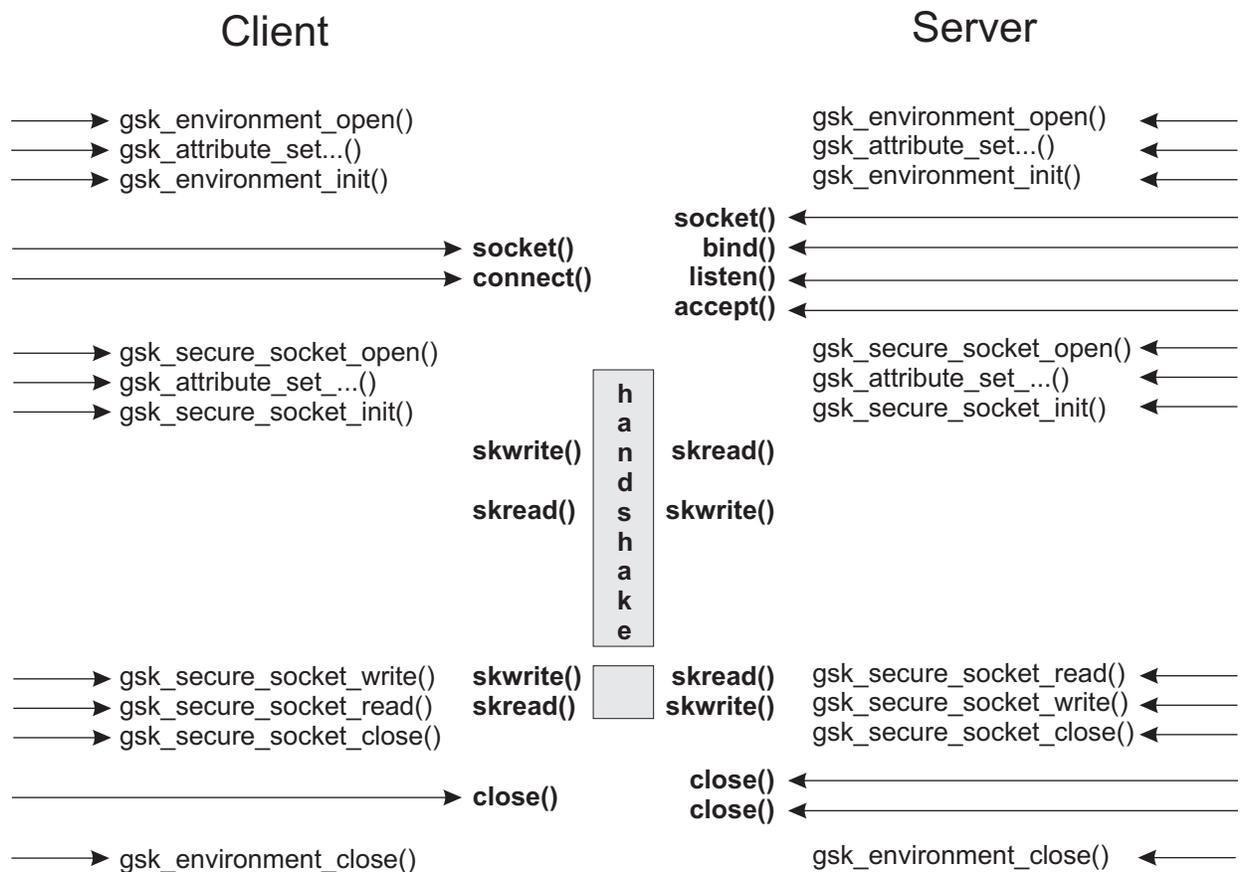


Figure 1. Sockets programming model using System SSL

Chapter 3. Using cryptographic features with System SSL

System SSL uses cryptographic features available on z/OS to offer a comprehensive range of cryptographic support. In addition to software cryptographic processing performed by System SSL, services offered by the Integrated Cryptographic Service Facility (ICSF) and the CP Assist for Cryptographic Function (CPACF) are employed to enhance System SSL with hardware cryptographic support for commonly used algorithms. ICSF also provides support for Elliptic Curve Cryptography (ECC).

In order for System SSL to use cryptographic support provided through ICSF, the ICSF started task must be running and the application user ID must be authorized for the appropriate resources in the RACF[®] CSFSERV class (when the class is active), either explicitly or through a generic resource profile. See “RACF CSFSERV resource requirements” on page 16 for further details. In addition to the CSFSERV class, the application user ID needs READ access to:

- RACF CSFKEYS class when SAF key rings are being used and the application's certificate keys are stored in ICSF'S PKDS. This access is not required if the CSFKEYS class is not active or the RACF resource is not defined.
- RACF resource USER.token-name within the CRYPTOZ class when either SAF key rings or PKCS #11 tokens are being used and the application's certificate keys are stored as secure keys in an ICSF PKCS #11 token. The CRYPTOZ class must be active and the RACF resource must exist, otherwise access is not granted.

For more information about access to CSFKEYS, see the RACDCERT command in *z/OS Security Server RACF Command Language Reference*. For more information about the CRYPTOZ class, see *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*.

Guidelines for using hardware cryptographic features

System SSL handshake processing uses the RSA and digital signature functions that are expensive functions when performed in software. For installations that have high volumes of SSL handshake processing, using the capabilities of the hardware provides maximum performance and throughput. For example, on z10, z196, zEC12, or z13, having a Crypto Express Coprocessor and/or Accelerator results 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 with SSL handshakes, moving the encrypt/decrypt processing to hardware (CPACF) provides maximum performance. The encryption algorithm is determined by the SSL cipher value. To use hardware, the ciphers symmetric algorithm must be available in hardware. For example, on z10, z196, zEC12, or z13, an application encrypting or decrypting data using the symmetric algorithm 3DES or AES would benefit from the processing being done in the hardware.

For maximum performance and throughput, it is recommended that hardware is used for both the SSL handshake and data encrypt/decrypt.

For information about the types of hardware cryptographic features supported by ICSF, see *z/OS Cryptographic Services ICSF Overview*. For information about

Using cryptographic features with System SSL

configuring and using ICSF, see *z/OS Cryptographic Services ICSF Administrator's Guide* and *z/OS Cryptographic Services ICSF System Programmer's Guide*.

Several products use System SSL. See the specific product publications to see if there is information about 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, see the topic about controlling who can use cryptographic keys and services in *z/OS Cryptographic Services ICSF Administrator's Guide*.

Overview of hardware cryptographic features and System SSL

System SSL might use ICSF or the CPACF for cryptographic hardware support, if they are available. Cryptographic hardware support provides performance benefits over software processing and might be used for particular cryptographic algorithms instead of the System SSL software algorithms. System SSL also uses ICSF for cryptographic algorithms that are not supported within the software of System SSL (for example, Elliptic Curve Cryptography). For algorithms for which System SSL has software versions, System SSL checks for hardware support during its runtime initialization and uses the support if available, unless the application specifies otherwise. See Appendix A, "Environment variables," on page 667 for information about the GSK_HW_CRYPTO environment variable (which specifies whether the hardware cryptographic support is used).

When using a secure key (a key stored either in the ICSF PKDS or a PKCS #11 token) or an algorithm that is not supported within System SSL's software, System SSL always uses ICSF for the cryptographic operation. If ICSF is not available, the operation fails.

If the appropriate hardware is available, System SSL uses the CPACF directly for symmetric encryption algorithms DES, 3DES, and AES-CBC, and SHA based digest algorithms. It calls ICSF for RSA signature and encryption operations. If these functions are not available in hardware, System SSL uses internal software implementations of the algorithms.

If a severe ICSF error occurs during a clear key RSA operation, System SSL stops using the hardware support and reverts to using the software algorithms, when applicable. In this event, hardware failure notification is available through the SSL Started Task or SSL trace output, if either facility is enabled. The SSL Started Task outputs 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 appears in the system log only. Any future cryptographic operations for the current SSL application that attempt to use this algorithm is performed in software. When the severe problem with ICSF is resolved, the System SSL application must be restarted to begin using ICSF again.

When using a secure key (a key stored either in the ICSF PKDS or a PKCS #11 token) or an algorithm that is not supported within System SSL's software (ECC and AES-GCM), System SSL always uses ICSF for the cryptographic operation. If ICSF is not available when these algorithms are called upon, the operation fails. Clear key ECC and AES-GCM operations use ICSF PKCS #11 support. For more information about ECC cryptographic support, see "Elliptic Curve Cryptography support" on page 14.

Using cryptographic features with System SSL

Note: System SSL can use secure key support for RSA, DSA, and ECC through ICSF. System SSL does not use secure symmetric keys except for the symmetric key that is used to encrypt the private key being encrypted by the `gsk_make_enveloped_private_key_msg()` API.

Table 1 describes the hardware cryptographic functions that are used by System SSL under different hardware configurations.

To use 4096-bit RSA keys in the hardware, you need one of the following:

- A z10 or higher processor with feature 0863 installed with a Crypto Express3 Coprocessor.
- A z196 (z114) or higher processor with a Crypto Express3 Accelerator with September 2011 or later Licensed Internal Code (LIC).

Table 1. Hardware cryptographic functions used by System SSL

Algorithm	z10			z196/z114			zEC12/zBC12				z13		
	CPACF	CEX2C / CEX3C	CEX2A / CEX3A	CPACF	CEX3C	CEX3A	CPACF	CEX3C/ CEX4C	CEX3A/ CEX4A	CEX4P	CPACF	CEX5C	CEX5A
DES	X			X			X				X		
3DES	X			X			X				X		
AES 128-bit	X			X			X				X		
AES 256-bit	X			X			X				X		
AES-GCM 128-bit				X			X				X		
AES-GCM 256-bit				X			X				X		
SHA-1	X			X			X				X		
SHA-2 (SHA-224)	X			X			X				X		
SHA-2 (SHA-256)	X			X			X				X		
SHA-2 (SHA-384)	X			X			X				X		
SHA-2 (SHA-512)	X			X			X				X		
PKA (RSA) Decrypt (Clear Private Key)		X	X		X	X		X	X			X	X
PKA (RSA) Decrypt (Secure Private Key)		X			X			X		X		X	
PKA (RSA) Encrypt		X	X		X	X		X	X			X	X
Digital Signature Generate (RSA) (Clear and/or Secure Private key)		X			X			X		X Secure key only.		X	
Digital Signature Verify (RSA)		X	X		X	X		X	X			X	X

Using cryptographic features with System SSL

Table 1. Hardware cryptographic functions used by System SSL (continued)

Algorithm	z10			z196/z114			zEC12/zBC12			z13			
	CPACF	CEX2C / CEX3C	CEX2A / CEX3A	CPACF	CEX3C	CEX3A	CPACF	CEX3C/ CEX4C	CEX3A/ CEX4A	CEX4P	CPACF	CEX5C	CEX5A
Digital Signature Generate (ECC) (Clear and/or Secure Private key)		X			X			X		X Secure key only.		X	

Random byte generation support

System SSL supports the generation of random bytes. This support is performed either by calling the ICSF CSFPPRF callable service or through a software implementation within System SSL. If ICSF is available during System SSL's runtime initialization, System SSL calls ICSF. If unavailable, System SSL's software implementation is used. If ICSF terminates or access to the ICSF callable service CSFPPRF is protected by a CSFSERV class profile and the application user is not authorized, the generation of random bytes is performed in software. For more information about the CSFSERV resource class, see "RACF CSFSERV resource requirements" on page 16.

If the System SSL application is FIPS enabled, see "Random byte generation" on page 20 for more information about random bytes generation in FIPS mode.

Elliptic Curve Cryptography support

System SSL uses ICSF callable services for Elliptic Curve Cryptography (ECC) algorithm support. For ECC support through ICSF, ICSF must be initialized with PKCS #11 support. For more information, see *z/OS Cryptographic Services ICSF System Programmer's Guide*. In addition, the application user ID must be authorized for the appropriate resources in the RACF CSFSERV class, either explicitly or through a generic resource profile. See Table 4 on page 17 for the required CSFSERV resources for each ECC function.

If the ICSF started task is not running as required or ECC support is otherwise unavailable, System SSL will fail if an ECC-based operation is required. In this event, notification is available through return or status codes and System SSL trace output.

Current ICSF cryptographic support for ECC can be verified using the DISPLAY CRYPTO function of the SSL Started Task. See Chapter 11, "SSL started task," on page 587 for more information.

ECC public/private keys must be defined over prime finite fields (F_p type fields) only; characteristic two finite fields (F_{2^m} type fields) are not supported. EC domain parameters may be defined using either the specifiedCurve format or the namedCurve format, as described in RFC 5480: *Elliptic Curve Cryptography Subject Public Key Information*. If the EC domain parameters are defined using the specifiedCurve format, then they must match a supported named curve.

The following named curves are supported:

- NIST recommended curves

- secp192r1 – {1.2.840.10045.3.1.1}
- secp224r1 – {1.3.132.0.33}
- secp256r1 – {1.2.840.10045.3.1.7}
- secp384r1 – {1.3.132.0.34}
- secp521r1 – {1.3.132.0.35}
- Brainpool defined curves
 - brainpoolP160r1 – {1.3.36.3.3.2.8.1.1.1}
 - brainpoolP192r1 – {1.3.36.3.3.2.8.1.1.3}
 - brainpoolP224r1 – {1.3.36.3.3.2.8.1.1.5}
 - brainpoolP256r1 – {1.3.36.3.3.2.8.1.1.7}
 - brainpoolP320r1 – {1.3.36.3.3.2.8.1.1.9}
 - brainpoolP384r1 – {1.3.36.3.3.2.8.1.1.11}
 - brainpoolP512r1 – {1.3.36.3.3.2.8.1.1.13}

Note: In FIPS mode, only NIST recommended curves are currently supported. Curves under 224 bits are not recommended.

For data signature generation and verification operations involving ECC-based algorithms, z/OS System SSL supports ECDSA with SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 digest algorithms. When creating signed certificates using the System SSL certificate management utility, **gskkyman**, or through CMS APIs that use a default digest algorithm, the recommended digest for the ECC key size of the signing private key is used (as specified in the following table).

Table 2. Recommended digest sizes for ECDSA signature key sizes

ECC curve type	ECDSA key sizes (bits)	Recommended digest algorithm	Signature algorithm type
x509_ecurve_brainpoolP160r1 x509_ecurve_secp192r1 x509_ecurve_brainpoolP192r1 x509_ecurve_secp224r1 x509_ecurve_brainpoolP224r1 x509_ecurve_secp256r1 x509_ecurve_brainpoolP256r1 x509_ecurve_brainpoolP320r1	160-383	SHA-256	x509_alg_ecdsaWithSha256
x509_ecurve_secp384r1 x509_ecurve_brainpoolP384r1	384-511	SHA-384	x509_alg_ecdsaWithSha384
x509_ecurve_brainpoolP512r1 x509_ecurve_secp521r1	512 and greater	SHA-512	x509_alg_ecdsaWithSha512

System SSL regards certain EC named curves to be the default curve for their key size. For CMS APIs that require ECC key generation and accept a key size parameter only, the default curve for the key size specified is used. These default EC named curves are outlined in the following table.

Table 3. Default EC named curves for specified key sizes

Key size (bits)	Default EC named curve	Named curve OID
160	brainpoolP160r1	1.3.36.3.3.2.8.1.1.1
192	secp192r1	1.2.840.10045.3.1.1
224	secp224r1	1.3.132.0.33

Using cryptographic features with System SSL

Table 3. Default EC named curves for specified key sizes (continued)

Key size (bits)	Default EC named curve	Named curve OID
256	secp256r1	1.2.840.10045.3.1.7
320	brainpoolP320r1	1.3.36.3.3.2.8.1.1.9
384	secp384r1	1.3.132.0.34
512	brainpoolP512r1	1.3.36.3.3.2.8.1.1.13
521	secp521r1	1.3.132.0.35

Diffie-Hellman key agreement

System SSL supports Diffie-Hellman (DH) key agreement group parameters as defined in PKCS #3 (Diffie-Hellman Key Agreement Standard) and RFC 2631: *Diffie-Hellman Key Agreement Method*. The Diffie-Hellman key agreement parameters are the prime P , the base G , and, in non-FIPS mode, the optional subprime Q , and subgroup factor J .

Diffie-Hellman key pairs are the private value X and the public value Y . The private value X is less than $Q-1$ if Q is present in the key parameters, otherwise, the private value X is less than $P-1$.

Multiple Diffie-Hellman key agreement keys can share domain group parameters (P 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 generates temporary Diffie-Hellman values if the group parameters in the client certificate are not the same as the group parameters in the server certificate. DSA keys may also share domain group parameters as Diffie-Hellman keys.

DH keys:

- Can be used only for end user certificates
- Can only be signed using a certificate that contains either an RSA or a DSA key
- Key size when in non-FIPS mode is between 512 and 2048 bits rounded up to a multiple of 64
- Key size in FIPS mode of 2048 bits
- Can only be used for connections where the cipher specification is a fixed Diffie-Hellman key exchange
- When used in fixed Diffie-Hellman key exchange must allow key agreement.

Only an RSA or DSA client certificate can be used in an ephemeral Diffie-Hellman key exchange.

RACF CSFSERV resource requirements

ICSF controls access to cryptographic services through the RACF CSFSERV resource class. An application using System SSL that requires cryptographic support from ICSF must be authorized for the appropriate resources in the class, either explicitly or through a generic resource profile. For more information, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

When the System SSL DLLs are loaded, System SSL determines what hardware is available by using the ICSF Query Algorithm callable service (CSFIQA). For this

Using cryptographic features with System SSL

reason, make sure that the RACF user ID that starts the application can access the CSFIQA resource of the CSFSERV class. If the user ID that starts the SSL application cannot access the CSFIQA resource of the CSFSERV class, System SSL cannot retrieve information by using the CSFIQA callable service, and the informational message ICH408I (which indicates insufficient authorization) may be issued to the console. Although System SSL processing continues, System SSL might not be aware of all the hardware that is currently available.

The following tables summarize the CSFSERV resources required for each ICSF cryptographic function used by System SSL.

Table 4. CSFSERV resources required for hardware support through ICSF callable services

Function	ICSF callable services	z10	z196/z114 and zEC12/zBC12
PKA (RSA) Encrypt	CSNDPKB CSNDPKE	-- CSFPKE	-- CSFPKE
PKA (RSA) Decrypt	CSNDPKB CSNDPKD	-- CSFPKD	-- CSFPKD
RSA Digital Signature Generation	CSNDPKB CSNDPKI CSNDDSG	-- CSFPKI CSFDSG	-- CSFPKI CSFDSG
RSA Digital Signature Verify	CSFDPKB CSNDDSV	-- CSFDSV	-- CSFDSV
ECC Digital Signature Generation (private key in the PKDS)	CSNDDSG		CSFDSG

Table 5. CSFSERV resources required for ICSF PKCS #11 callable services support

Function	ICSF PKCS #11 callable services	CSFSERV resources required
ECC Key Generation	CSFPGKP CSFPGAV CSFPTRD	CSF1GKP CSF1GAV CSF1TRD
RSA/ECC Digital Signature Generation	CSFPTRC CSFPPKS CSFPTRD	CSF1TRC CSF1PKS CSF1TRD
ECC Digital Signature Verify	CSFPTRC CSFPPKV CSFPTRD	CSF1TRC CSF1PKV CSF1TRD
ECDH Derive Key	CSFPTRC CSFPDVK CSFPGAV CSFPTRD	CSF1TRC CSF1DVK CSF1GAV CSF1TRD
Diffie-Hellman in FIPS mode	CSFPTRC CSFPDVK CSFPGKP CSFPGSK CSFPGAV CSFPTRD	CSF1TRC CSF1DVK CSF1GKP CSF1GSK CSF1GAV CSF1TRD
AES-GCM Secret Key Decrypt	CSFPSKD CSFPTRC CSFPTRD	CSF1SKD CSF1TRC CSF1TRD

Using cryptographic features with System SSL

Table 5. CSFSERV resources required for ICSF PKCS #11 callable services support (continued)

Function	ICSF PKCS #11 callable services	CSFSERV resources required
AES-GCM Secret Key Encrypt	CSFPSKE CSFPTRC CSFPTRD	CSF1SKE CSF1TRC CSF1TRD
Random Number Generation	CSFPPRF	CSFRNG
Secure PKCS #7 Make Enveloped Data Message	CSFPTRC CSFPGSK CSFPWPK CSFPTRD	CSF1TRC CSF1GSK CSF1WPK CSF1TRD
Secure PKCS #7 Read Enveloped Data Message	CSFPPKS	CSF1PKS
Secure PKCS #12 Private Key Export	CSFPGSK CSFPWPK CSFPTRC CSFPTRD	CSF1GSK CSF1WPK CSF1TRC CSF1TRD
RSA PKCS #11 Secure Key Decrypt	CSFPPKS	CSF1PKS

PKCS #11 and setting CLEARKEY resource within CRYPTOZ class

The CLEARKEY.token-name resource within the CRYPTOZ class controls the ICSF policy for creating a clear key versus a secure key. When the resource is defined and set to NONE, System SSL's usage of the PKCS #11 callable services to generate keys is restricted to secure keys only. This causes functions within System SSL to fail. System SSL uses both explicit tokens and the SYSTOK-SESSION-ONLY omnipresent token.

The following are examples that can fail in this environment in System SSL:

- The **gskkyman** utility or CMS APIs that create ECC or DH (FIPS mode) keys or certificates.
- Ephemeral DH key exchanges during a SSL/TLS handshake.
- Ephemeral ECDH key exchanges during a SSL/TLS handshake.

PKCS #11 Cryptographic operations using ICSF handles

When executing cryptographic operations against PKCS #11 certificates and keys, System SSL uses the 44-byte handle as defined by ICSF. See *Introducing PKCS #11 and using PKCS #11 callable services in z/OS Cryptographic Services ICSF Application Programmer's Guide* for the definition of this handle. When using System SSL CMS APIs, the ICSF handle may be referred to as a label, for example, the input *private_key_label* to Certificate Management Services (CMS) API **gsk_make_enveloped_private_key_msg()** is the ICSF definition of a handle. The ICSF 44-byte handle is not the same as a PKCS #11 object handle, which is defined as PKCS #11 attribute CK_ULONG.

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 is 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' mode. Applications wanting to execute in FIPS mode must code to the `gsk_fips_state_set()` API. See "Application changes" on page 26 for more information.

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. Hardware cryptographic functions allowed in FIPS mode support clear keys and secure PKCS #11 keys. Secure keys stored in the PKDS are not supported.

Table 6 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 cryptographic features with System SSL," on page 11 for more information about processors, CPACF algorithm availability, and cryptographic card support.

Table 6. Algorithm support: FIPS and non-FIPS

Algorithm	Non-FIPS				FIPS			
	Sizes	System SSL software	Direct calls to CPACF	Support through ICSF	Sizes	System SSL software	Direct calls to CPACF	Support through ICSF
RC2	40 and 128	X						
RC4	40 and 128	X						
DES	56	X	X					
3DES	168	X	X		168	X	X	
AES	128 and 256	X	X		128 and 256	X	X	
AES-GCM	128 and 256			X	128 and 256			X
MD5	48	X						
SHA-1	160	X	X		160	X	X	
SHA-2	224, 256, 384, and 512	X	X		224, 256, 384, and 512	X	X	

System SSL and FIPS 140-2

Table 6. Algorithm support: FIPS and non-FIPS (continued)

Algorithm	Non-FIPS				FIPS			
	Sizes	System SSL software	Direct calls to CPACF	Support through ICSF	Sizes	System SSL software	Direct calls to CPACF	Support through ICSF
RSA	512–4096	X		X	1024–4096	X		X
DSA	512–2048	X			1024–2048	X		
DH	512–2048	X			2048			X
NIST ECC	192–521				192–521			X
ECC Brainpool	160–512			X				

Notes:

- In FIPS mode, only NIST ECC recommended curves are currently supported. Curves under 224 bits are not recommended. Enforcement is the responsibility of the calling application or the system administrator.
- NIST SP800-131 recommended transition algorithm key sizes of RSA \geq 2048, DSA \geq 2048, NIST ECC recommended curves \geq 224, and SHA-1 (usage in digital signatures) are not enforced by System SSL. Brainpool ECC curves are not to be used in FIPS mode. Enforcement is the responsibility of the calling application

Random byte generation

When executing in FIPS mode, System SSL supports the generation of random bytes. System SSL generates random bytes by using ICSF's CSFPPRF callable service. In order for System SSL to call this service, ICSF must be available before System SSL's run time that is being initialized by the application. If access to the CSFPPRF callable service is protected by a CSFSERV class profile, the application's user ID must be authorized to use the service. For more information about the CSFSERV resource class, see "RACF CSFSERV resource requirements" on page 16.

Diffie-Hellman key agreement

When executing in FIPS mode, System SSL uses ICSF's Diffie-Hellman support as documented in *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*. In order for System SSL to be able to use ICSF, ICSF must be available before System SSL's run time is being initialized by the application. If access to the ICSF services is being protected by the CSFSERV class profile, the application user ID must be authorized. For more information about the CSFSERV resource class, see "RACF CSFSERV resource requirements" on page 16.

In FIPS mode, the only Diffie-Hellman key agreement parameters used are the prime P, and the base G.

Diffie-Hellman key size in FIPS mode is 2048 bits.

Certificates

When executing in FIPS mode, System SSL can only use certificates that use the algorithms and key sizes shown in Table 6 on page 19. 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 use the TLS V1.0, TLS V1.1, and TLS V1.2 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 default 2-character specifications string reflects the default order of suites supported:

```
35363738392F303132330A1613100D0915120F0C
```

When executing in non-FIPS mode, if GSK_V3_CIPHERS is set to GSK_V3_CIPHERS_CHAR4 and a cipher specification is not set in GSK_V3_CIPHER_SPECS_EXPANDED, the default cipher specification is set as follows:

```
00350036003700380039002F0030003100320033000A0016
00130010000D000900150012000F000C
```

The algorithm restrictions (see Table 6 on page 19) result in the following default cipher specifications string in FIPS mode:

```
35363738392F303132330A1613100D
```

If using 4-character cipher specifications, the default cipher specifications string in FIPS mode becomes:

```
00350036003700380039002F0030003100320033000A001600130010000D
```

Only the following cipher suites are compatible with the restrictions in Table 6 on page 19 and are therefore supported while executing in FIPS mode:

When using 2-character cipher suites:

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

When using 4-character cipher suites:

```
000A 000D 0010 0013 0016 002F 0030 0031 0032 0033 0035 0036 0037 0038
0039 C003 C004 C005 C008 C009 C00A C00D C00E C00F C012 C013 C014
```

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

For more information about ciphers and their 2-character or 4-character values, see Appendix C, “Cipher suite definitions,” on page 687.

System SSL module verification setup

System SSL requires Security Level 3 FMID JCPT421 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 292.

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 before 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 that 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 - G2 certificate. This certificate is shipped as a default CERTAUTH certificate in the RACF database under the label 'STG Code Signing CA - G2'.

Note: A sample clist, `GSKRACF`, is shipped in `pdsename.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 - G2'))
RACDCERT CERTAUTH ALTER (LABEL('STG Code Signing CA - G2')) TRUST
```

2. Create a key ring to hold the STG Code Signing CA - G2 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. Make sure that the ID is 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 suggested key ring name is `CODE.SIGNATURE.VERIFICATION.KEYRING`.

- Determine if signature verification key ring is already active:

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

The key ring is 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 - G2') USAGE(CERTAUTH))
```

- If a key ring exists, verify that 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 - G2') 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: Because of 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: Because of 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.

- 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, see *z/OS Security Server RACF System Programmer's Guide*.
- b. 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 that 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: Because of 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 GSKKYMAN 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 about using VLF see *VLF considerations for program signature verification* in *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, PKCS #12 file, 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 6 on page 19.

SAF key rings and PKCS #11 tokens

Provided a certificate and its signers chain use only valid algorithms and key sizes, then there are no changes that are 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 if 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 fails. 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.

The **gskkyman** utility 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 later used while executing in FIPS mode.

Key database files

To use a key database in FIPS mode, it must be created as a FIPS mode database. Key databases that are 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 the **gskkyman** utility, see “Creating, opening, and deleting a key database file” on page 538. 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 292).

The following are key points when using FIPS key databases:

- Only certificates that meet the requirements for FIPS (see Table 6 on page 19) 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, the **gskkyman** utility ensures that it is executing in FIPS mode. If an application modifies the key database by using the Certificate Management Services (CMS) APIs, then it too must ensure that 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.

The **gskkyman** utility 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 6 on page 19 may be added to the key database.

PKCS #12 files

To use a PKCS #12 file in FIPS mode, the file must be protected using 3DES. When creating a PKCS #12 file from certificates within a key database file, using the **gskkyman** utility, the key database must be a FIPS key database.

Provided a certificate and its signers chain use only valid algorithms and key sizes, there are no changes that are required if using a PKCS #12 file. A PKCS #12 file may contain certificates with keys sizes or algorithms that are not supported in FIPS mode. While executing in FIPS mode, if an attempt to use a certificate with unsupported key size or algorithms is made, the process fails. The corrective action is to either add or replace certificates with key sizes and algorithms that are valid in FIPS mode, or to execute in non-FIPS mode.

Application changes

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 must request that 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 292). To set FIPS mode, `gsk_fips_state_set()` must be executed before all other System SSL functions except for `gsk_get_cms_vector()`, `gsk_get_ssl_vector()` and `gsk_fips_state_query()`. It is possible to switch to non-FIPS mode later. It is not possible to switch from non-FIPS mode to FIPS mode at any time.

The FIPS mode setting applies to the entire process. Once set, then all threads of the same process execute in FIPS mode. If any thread switches to non-FIPS mode, then all threads in the same process execute in non-FIPS mode.

When executing in FIPS mode and a severe cryptographic problem is encountered, one of the following return codes is 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()`, `gsk_query_crypto_level()`, and `gsk_strerror()` fails.

- 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 an RSA or DSA key pair
- CMSERR_KATPW_FAILED - Failure was encountered by the `gsk_perform_kat()` API when performing known answer tests against the System SSL cryptographic algorithms.
- CMSERR_KATPW_ICSF_FAILED - Failure was encountered by the `gsk_perform_kat()` API when performing known answer tests using ICSF.

The sample files (see Appendix B, “Sample C++ SSL files,” on page 685) `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 before any other System SSL function.

SSL started task

The System SSL started task (GSKSRVR) executes in non-FIPS mode by 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 JCPT421 is not installed), it executes in non-FIPS mode after issuing message GSK01054E (see “SSL started task messages (GSK01nnn)” on page 656).

Sysplex session ID cache

GSKSRVR must be running in FIPS mode to maintain Sysplex Session ID cache entries for SSL server applications executing in FIPS mode. An SSL server application executing in FIPS mode caches 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 that are 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 Version 1 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". It is suggested that new application programs do not use the deprecated SSL interface. For a complete list and descriptions of the suggested APIs, see Chapter 7, "API reference," on page 57. See Chapter 9, "Deprecated Secure Socket Layer (SSL) APIs," on page 487 for more information about deprecated APIs.

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 have a certificate repository available for the application. The certificate repository can be a key database file, PKCS #12 file, PKCS #11 token, or SAF key ring. See Chapter 10, "Certificate/Key management," on page 519 for details about creating and managing key database files or PKCS #11 tokens. For SAF key rings, see the RACDCERT command information in *z/OS Security Server RACF Command Language Reference* for more information.

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 System SSL programming interfaces. See Chapter 7, "API reference," on page 57 for a description of the format of the System SSL programming interfaces.

Before 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 an 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.

`gsk_environment_open()`

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

`gsk_attribute_set...()`

Sets environment attributes such as:

Writing and building a z/OS System SSL application

- The SSL protocol version to be used: SSL Version 2.0, SSL Version 3.0, TLS Version 1.0, TLS Version 1.1, and/or TLS Version 1.2.
- The key database to be used. (key database file, PKCS #12 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 by using a stashed password file. See Chapter 10, "Certificate/Key management," on page 519 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. When using a PKCS #12 file, a stash file must not be specified.

- 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, TLS Version 1.0 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.

```
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);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_SSLV3, GSK_PROTOCOL_SSLV3_OFF);
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_OFF);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1_2, GSK_PROTOCOL_TLSV1_2_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 TLS Version 1.0, TLS Version 1.1, and TLS Version 1.2.

```
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);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_SSLV3, GSK_PROTOCOL_SSLV3_OFF);
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_enum(env_handle, GSK_PROTOCOL_TLSV1_2, GSK_PROTOCOL_TLSV1_2_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: When 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.

When the System SSL program successfully creates the SSL environment, it must now perform the steps that are 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:

```
int server_sock;

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

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

```
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:

```
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:

```
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 is 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.

Writing and building a z/OS System SSL application

- The set of SSL protocol cipher specifications to be allowed for the secure session specified using 4-character cipher specifications. (For example, ciphers utilizing a RSA key exchange with either AES 128/256 or 3DES encryption.) The cipher is selected by the System SSL server program according to the server's order of usage preference.
- The 4-character cipher specification list in `GSK_V3_CIPHER_SPECS_EXPANDED` is used.
- 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.

```
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_enum(soc_handle, GSK_SESSION_TYPE, GSK_SERVER_SESSION);
rc = gsk_attribute_set_buffer(soc_handle, GSK_V3_CIPHER_SPECS_EXPANDED, "0035002F000A",0);
rc = gsk_attribute_set_enum(soc_handle, GSK_V3_CIPHERS, GSK_V3_CIPHERS_CHAR4);
rc = gsk_attribute_set_callback(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 39. Use these function calls, `send()` and `recv()`, to send and receive the application data.

```
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 successfully calls `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:

```
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:

```
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:

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

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

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

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

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

System SSL client program

The socket and System SSL API calls used by the client program are very similar to the calls used by the server program. 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:

```
int sock;

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

Now that the client program socket is created, connect the socket to the server program port using this function call:

```
int rc;
int namelength;
struct sockaddr_in name;
char *ServerHostName;

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 causes 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 specifications to be allowed for the secure session in client-preferred order specified using 4-character cipher specifications. (For example, ciphers utilizing a RSA key exchange with either AES 128/256 or 3DES encryption.)

Note: Although the client is allowed to specify a preference order, an SSL server might not accept the preference.

- The 4-character cipher specification list in GSK_V3_CIPHER_SPECS_EXPANDED is used.
- 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.

```
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);
```

Writing and building a z/OS System SSL application

```
rc = gsk_attribute_set_numeric_value(soc_handle, GSK_FD, sock);
rc = gsk_attribute_set_buffer(soc_handle, GSK_KEYRING_LABEL, "THELABEL",0);
rc = gsk_attribute_set_enum(soc_handle, GSK_SESSION_TYPE, GSK_CLIENT_SESSION);
rc = gsk_attribute_set_buffer(soc_handle, GSK_V3_CIPHER_SPECS_EXPANDED, "0035002F000A",0);
rc = gsk_attribute_set_enum(soc_handle, GSK_V3_CIPHERS, GSK_V3_CIPHERS_CHAR4);
rc = gsk_attribute_set_callback(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 39. Use these function calls, **send()** and **recv()**, to send and receive the application data.

```
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 successfully calls **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:

```
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:

```
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 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:

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

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

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

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

```
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 29).
2. Compile your System SSL source program using the DLL compiler option.
3. Include the `/usr/lib/GSKSSL.x` or `/usr/lib/GSKSSL64.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, create a SAF key ring or PKCS #11 token using the `RACDCERT` command, or utilize an existing PKCS #12 file. The name of the key database file, PKCS #12 file, z/OS PKCS #11 token, or SAF key ring must match the name you specified as the `GSK_KEYRING_FILE` on the `gsk_attribute_set_buffer()` API. For key database files, you need to specify either the password associated with the key file or the stash file name. For PKCS #12 files, you need to specify the password associated with the file. 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 SAF key ring or z/OS PKCS #11 token. Note that the password is case-sensitive. See Chapter 10, “Certificate/Key management,” on page 519 for information about how to create a key database file, SAF 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 certificate repository, you can run the System SSL application. To run the application follow these steps:

1. Ensure that `pdsname.SIEALNKE`, the PDSE that contains the System SSL DLLs, is in the `MVS™` search order. If it is 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:

```
export STEPLIB=$STEPLIB:pdsname.SIEALNKE
```
2. Ensure that the key database file, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token is accessible to the System SSL application.
3. Run the System SSL application.

Note:

1. SSL applications must be run from within a POSIX environment.
2. Once SSL applications call `gsk_initialize()` or `gsk_environment_open()`, they cannot destroy the LE environment.
3. SSL applications must call SSL APIs from a C program, as they are C APIs.

System SSL application programming considerations

When programming System SSL applications, you should consider the following.

- Will the application need to communicate with other applications using non-blocking I/O? The socket connections used for communication between System SSL applications are, by default, blocking. An application attempting to read or write to a socket is blocked until all expected data is received. This might not be desirable, because no other processing can occur while the application is waiting for a read or write to complete.
- Will the application need to prompt the client user to select a certificate from a list during the client authentication process in the SSL handshake? This behavior, if needed, can be accomplished using a registered callback routine that is invoked from inside the `gsk_secure_socket_init()` function call.
- Will the application need to override System SSLs default I/O callback routines to specify I/O behavior? This can be accomplished by specifying your own callback routines for receiving and sending data.

Writing and building a z/OS System SSL application

- Will application-specific data need to be available to the SSL callback routines? If needed, application-specific data can be made available using the `gsk_attribute_set_buffer()` and `gsk_attribute_get_buffer()` function calls.
- Considering both security and performance benefits, how long should SSL sessions be allowed to remain active? 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 timeout values and a larger cache size.
- Will the application need to initiate session renegotiation? If needed, the application can call the `gsk_secure_socket_misc` API to renegotiate the communications session to establish a new session key or have the session cipher reset. Notification callback routines allow the application to take specific actions during a session renegotiation.
- Will the application need to add functionality to the Transport Layer Security (TLS) protocol? Applications can define a TLS extension to the SSL environment or connection by calling the `gsk_attribute_set_tls_extension()` function.
- Will a "Suite B Compliant" TLS V1.2 session be required? System SSL allows TLS client and server applications to specify a profile compliant with Suite B Cryptography as defined in RFC 5430: *Suite B Profile for Transport Layer Security (TLS)*. This profile restricts the cryptographic algorithms used for the session to the set of algorithms supported by Suite B Cryptography.
- Will certificate revocation be needed when validating the partner's certificate? Applications can enable the usage of revocation information obtained through OCSP responses, HTTP CRLs and/or LDAP CRLs.

Non-Blocking I/O

Applications wanting 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 is 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 is 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 set up 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 might try read/write again later.

Non-Blocking socket primer

When a server wants to communicate with clients by using a socket, these routines are used:

Table 7. Server communicating with clients by way of a socket

Routine	Purpose
1) <code>socket()</code>	Create a socket
2) <code>bind()</code>	Register the socket
3) <code>listen()</code>	Indicate willingness to accept connections
4) <code>accept()</code>	Accept a connection request

Table 7. Server communicating with clients by way of a socket (continued)

Routine	Purpose
5) Read request	
6) Write response	
7) Return to step 4	

Once the **accept()** routine is called, the server blocks until data is available for the socket. Problems arise when the server wants to monitor multiple sockets simultaneously or if the server wants 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 38. 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 8. Using the *select()* routine

Routine	Purpose
1) socket()	Create a socket
2) bind()	Register the socket
3) listen()	Indicate willingness to accept connections
4) Set socket as non-blocking	See “Enable/disable non-blocking mode” on page 38
5) select()	Monitor a number of sockets
6) accept()	Accept a connection request
7) Read request	If unable to read all data, return to step 5
8) Write response	If unable to write all data, return to step 5
9) Return to step 4	

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 is configured as non-blocking, any calls to

Writing and building a z/OS System SSL application

`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 is 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 is 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 returns the partial data received or written. Since System SSL processes encrypted data, it is not possible to decrypt a message until the entire message is 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()` returns 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 is not sent to the partner, but the SSL handshake completes. The server decides 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 I/O

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 executes a `recv()` and for write a `send()`. If not using TCP/IP, applications should also consider the specification of the `getpeername` and `setsocketoptions` 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 returns 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 is 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

The SSL protocol has a mechanism built in to allow for faster secure connections between a client/server pair. There is a concept of an 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 entry. 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, so avoiding it improves performance for clients and servers using SSL. A SID cache entry exists for a limited time. Take care when specifying how long an SSL session is allowed to live. Setting the SID cache timeout or number of SID cache entries to ZERO turns 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)

SID caching for the client is done internally within the clients address space, and each SSL environment has its own cache. The server can either cache within its address space per SSL environment or externally through the GSKSRVR for SYSPLEX caching. SYSPLEX caching allows session information to be shared among like servers or processes. See Chapter 11, "SSL started task," on page 587 for more information about Sysplex caching.

Modifying SSL session caching parameters can help tune the security performance characteristics of SSL enabled servers and clients. The contents of the internal client and server caches are controlled by the setting of an expiration lifetime for an SSL session ID entry and the number of entries that can reside concurrently in the cache. Separate caches exist for SSL V2 and SSL V3 (TLS) sessions. The internal SSL SID cache is fixed to a configurable number of entries defined when the SSL environment is being established. By default, the SSL V2 cache size is 256 entries and can be modified through the `GSK_V2_SIDCACHE_SIZE` environment attribute. The default expiration (or timeout) is 100 seconds and can be modified through the `GSK_V2_SESSION_TIMEOUT` environment attribute. By default, the SSL V3 (TLS) cache size is 512 entries and can be modified through the `GSK_V3_SIDCACHE_SIZE` environment attribute. The default expiration (or timeout) is 24 hours and can be modified through the `GSK_V3_SESSION_TIMEOUT` environment attribute. There is no way to remove or to reuse entries for other connections except for repeated connections between the same client/server pair.

Each time a full handshake is performed and caching is active (cache size !=0), a SID cache entry is created and added to the cache. During the add process, detected expired SID entries are removed. If the cache reaches its size limit, an entry is removed from the cache and the newly created SID entry is added.

Session ID cache replacement

The list of options for extending SID caching functionality can become quite long so an external SID cache API was created for those who are more discriminating about managing SID cache data. There are several callbacks used for external SID cache access.

Note that there are probably few applications where using an external SID cache makes sense. Some suggested environments where it might be considered is in a server configuration where multiple instances of a server exist for workload balancing purposes. It might be desirable to have a single SID cache 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:

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

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

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

typedef void                   (*ptgsk_freecache) (
    gsk_data_buffer *
    ssl_session_data);

typedef struct _gsk_sidcache_callback {
    ptgsk_getcache             Get;
    ptgsk_putcache             Put;
    ptgsk_deletcache           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.

Writing and building a z/OS System SSL application

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 calls 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 compatible with earlier versions: 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 supports. TLS extensions may be defined:

- After `gsk_environment_open()` is performed but before the `gsk_environment_init()` call
- After `gsk_secure_socket_open()` is performed but before the `gsk_secure_socket_init()` call

TLS extensions that are defined for an SSL environment applies 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 are 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, including TLS clients that do not support the extensions.

```
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]);

/*
 * 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]);
```

Writing and building a z/OS System SSL application

```
/*
 * Initialize the SSL environment
 */
rc = 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 fails.

```
int rc;
gsk_handle envHandle;
gsk_handle conHandle;

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
 */
rc = 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;
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.clientNameList.setSni = TRUE;
/* enable extension */
tls_extn_con[0].u.clientNameList.unrecognized_name_fatal = TRUE;
/* unrecognized name is fatal */
tls_extn_con[0].u.clientNameList.count = 2;
tls_extn_con[0].u.clientNameList.serverNames = serverNames;
rc = gsk_attribute_set_tls_extension(envHandle, &tls_extn_con[0]);

/*
 * Modify truncated HMAC extension
 */
```

```
memset(&tls_extn_con[1], 0, sizeof(gsk_tls_extension));
tls_extn_con[1].extId = GSK_TLS_EXTID_TRUNCATED_HMAC;
tls_extn_con[1].required = FALSE; /* optional extension */
tls_extn_con[1].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);
```

Suite B cryptography support

System SSL allows TLS client and server applications to specify a profile compliant with Suite B Cryptography as defined in RFC 5430: *Suite B Profile for Transport Layer Security (TLS)*. This profile restricts the cryptographic algorithms that are used for the session to the set of algorithms that are supported by Suite B Cryptography. Communication is possible between TLS clients that require Suite B cryptography and TLS servers that do not explicitly support Suite B cryptography, and vice versa, provided the non-Suite B entity supports the Suite B compliant cryptographic algorithms.

Suite B cryptography does not define cryptographic algorithms. Instead, it specifies the cryptographic algorithms that can be used in a “Suite B Compliant” TLS V1.2 session. Suite B requires the key establishment and authentication algorithms that are used in TLS V1.2 sessions to be based on Elliptic Curve Cryptography, and the encryption algorithm to be AES.

The security levels that are defined in the Suite B profile are:

- 128-bit security level, which corresponds to an elliptic curve size of 256 bits and AES-128
- 192-bit security level, which corresponds to an elliptic curve size of 384 bits and AES-256

Cipher suites that are allowed for the 128-bit and 192-bit Suite B profiles are:

Table 9. Suite B supported cipher suites

Cipher Suite	128-bit security level	192-bit security level
C02B	X	
C023	X	
C02C		X
C024		X

For more information about the cipher suites, see Appendix C, “Cipher suite definitions,” on page 687.

The Suite B standard specifies the elliptic curves that are allowed in a TLS connection. The following is a list of the curves that are allowed for the 128-bit and 192-bit Suite B profiles.

Table 10. Supported curves

Named Curve	128-bit security level	192-bit security level
secp256r1 – {1.2.840.10045.3.1.7}	X	
secp384r1 – {1.3.132.0.34}		X

Server and client certificates that are used to establish a Suite B-compliant connection must be signed with ECDSA.

Writing and building a z/OS System SSL application

- For certificates used at the 128-bit security level, the subject public key must use the secp256r1 curve and be signed with either the secp384r1 curve or the secp256r1 curve.
- For certificates used at the 192-bit security level, the subject public key must use the secp384r1 curve and be signed with the secp384r1 curve.

Whenever a Suite B-compliant client and a Suite B-compliant server establish a TLS V1.2 session, only Suite B algorithms are employed. For more information about the cipher suites, see Appendix C, “Cipher suite definitions,” on page 687.

Note that in a fully Suite B-compliant session, the TLS 1.2 protocol must be used to establish an SSL connection. Therefore, when System SSL is configured to run in a Suite B-compliant mode, any non-TLS 1.2 protocols that are configured for the connection are ignored and the TLS 1.2 protocol is activated, if not already active.

Additionally, Suite B also places restrictions on which cipher suites, elliptical curves, and signature algorithms can be used in a Suite B-compliant session. When System SSL is running in a Suite B mode, any cipher suites, cipher format, elliptical curves, and signature algorithms that are configured are ignored. Only the cipher suites, cipher format, elliptical curves, and signature algorithms for the profile that is chosen for the Suite B session are used by System SSL to establish the connection.

SSL/TLS partner certificate revocation checking

When performing revocation information checking against a partner certificate being used for a secure connection, the SSL/TLS application can be configured to obtain revocation information from any combination of the following revocation sources:

- A dedicated OCSP responder or OCSP responders specified in the Authority Information Access (AIA) extension.
- An HTTP server specified in the CRL Distribution Points (CDP) extension.
- A dedicated LDAP server.

Enabling OCSP support

OCSP revocation information can be obtained through OCSP responders identified within a certificate AIA extension (URI value) or through a dedicated OCSP responder.

To enable the use of the certificate AIA extension, when present in the certificate being validated, the GSK_OCSP_ENABLE setting must be set to ON. Note that in order to use the AIA extension, the extension must contain at least one identifying OCSP responder entry. An OCSP responder entry contains an access method of OCSP and URI access location.

When processing the possible values in the AIA extension, an attempt to contact each OCSP responder is tried and processing stops with the first responder that is able to be successfully contacted. The OCSP response from that OCSP responder is used to determine the revocation status.

To enable the used of a dedicated OCSP responder, GSK_OCSP_URL must be set to the HTTP URL of the OCSP responder. Only one dedicated responder can be specified. For information about specifying a HTTP URL, see the description for GSK_OCSP_URL in “gsk_attribute_set_buffer()” on page 82.

If GSK_OCSP_ENABLE is set ON and the dedicated OCSP responder is specified (GSK_OCSP_URL), both the dedicated OCSP responder and the responders identified in the AIA extension may be used to obtain revocation status. By default, the dedicated OCSP responder is utilized first during validation of a certificate. The default order can be changed through the GSK_OCSP_URL_PRIORITY setting. To have the dedicated OCSP responder (GSK_OCSP_URL) used after the AIA extension, GSK_OCSP_URL_PRIORITY must be set to OFF.

In all cases, once an OCSP responder has returned a response, the response is used to determine the revocation state of the certificate being validated.

If the dedicated OCSP responder (GSK_OCSP_URL) requires the OCSP request to be digitally signed or your security policy requires signing, signing is enabled by specifying the certificate to be used for signing and optionally the signature algorithm. The signing certificate is identified by setting GSK_OCSP_REQUEST_SIGKEYLABEL to the label of the certificate. The optional signature algorithm is specified through GSK_OCSP_REQUEST_SIGALG and defaults to RSA with SHA-256 (0401).

All OCSP responses are digitally signed by the OCSP responder using a CA certificate. The response signature must be verified using the public key of that certificate. The OCSP signing certificate must be found in either the trusted certificate store, the untrusted certificates that are provided through the handshake messages, or the OCSP response. The certificate chain for the OCSP signing certificate must also be validated through the root certificate for the chain and the root certificate must be in the trusted certificate store. Note that revocation checking of the OCSP response signer certificate is not performed if the id-pkix-ocsp-nocheck extension is present in the signing certificate.

The following OCSP request and response characteristics can be tailored to your environment:

HTTP method

The GSK_OCSP_RETRIEVE_VIA_GET setting is used to indicate whether the HTTP GET or the HTTP POST method is to be used when sending OCSP requests to the OCSP responder. By default, the HTTP POST method is used. The HTTP GET method allows for the enablement of HTTP caching on the OCSP responder and can only be used when the OCSP request is less than 255 bytes.

Note: System SSL sends HTTP OCSP GET requests as described in RFC 2560, which includes URL encoding of the base-64 encoding of the DER encoding of the OCSPRequest. However, some HTTP servers are unable to properly handle the URL encoding and may return unexpected HTTP errors while parsing the OCSP GET request. If these problems persist and it is not possible to correct the HTTP server behavior, send the OCSP requests with the HTTP POST method.

HTTP proxy server

To allow the OCSP request and OCSP response to be routed via a proxy HTTP server, GSK_OCSP_PROXY_SERVER_NAME and GSK_OCSP_PROXY_SERVER_PORT must be set.

Nonce The OCSP protocol allows for the OCSP request and corresponding OCSP response to have additional verifying information in the form of a nonce value. The nonce is intended to cryptographically bind the OCSP response to a specific OCSP request. A nonce is sent when

Writing and building a z/OS System SSL application

GSK_OCSP_NONCE_GENERATION_ENABLE is set to ON and checked when GSK_OCSP_NONCE_CHECK_ENABLE is set to ON. The size of the nonce can be specified by GSK_OCSP_NONCE_SIZE setting. Enabling this option improves security, but may cause failures if the OCSP responder does not support nonces.

OCSP response caching

OCSP responses received from an OCSP responder can be cached in the application's address space to save the processing time required to contact an OCSP responder. OCSP caching is enabled by default when OCSP revocation checking is enabled. The maximum number of OCSP responses that are allowed to be stored in the OCSP cache is indicated by the GSK_OCSP_CLIENT_CACHE_SIZE setting, which has a default of 256. To have the OCSP responder contacted for every validation, GSK_OCSP_CLIENT_CACHE_SIZE must be set to 0. With the OCSP cache, it is possible to limit the number of cached responses tied to a specific issuing CA certificate through the GSK_OCSP_CLIENT_CACHE_ENTRY_MAXSIZE setting. By default, this size is set to 0, which indicates there is no limit on the number of cached certificate statuses allowed for a specific issuing CA certificate other than the limit imposed by the overall GSK_OCSP_CLIENT_CACHE_SIZE setting. The value specified for the GSK_OCSP_CLIENT_CACHE_ENTRY_MAXSIZE setting cannot exceed the GSK_OCSP_CLIENT_CACHE_SIZE setting.

OCSP responses are only cached when they contain a nextUpdate time. No nextUpdate time indicates that the OCSP response is only valid at the time the response was provided. The nextUpdate time is used as the expiration time of the OCSP response in the cache. If the HTTP response containing the OCSP response includes the max-age option in the Cache-Control directive, the expiration time of the OCSP response in the cache is either the nextUpdate time or max-age plus the current time, whichever is earlier.

If the HTTP response contains the Expires directive and the max-age option in the Cache-Control directive is not specified, the expiration time of the OCSP response in the cache is either the nextUpdate time or the Expires time, whichever is earlier.

When a new OCSP response is being added to the OCSP cache and the cache reaches its size limit, all expired entries are removed from the cache and then the newly obtained OCSP response is added.

If during the validation process, an attempt to use a cached OCSP response detects the entry is expired, the OCSP response is removed from the cache and the OCSP responder is contacted to obtain a new OCSP response.

Response size

Although OCSP responses should be small in size, a default maximum size of 20480 (20K) has been defined. The size can be overridden through the GSK_OCSP_MAX_RESPONSE_SIZE setting.

Response timeout

When communicating with the OCSP responder, by default, the amount of time to wait for the response is set to 15 seconds. The timeout can be overridden through the GSK_OCSP_RESPONSE_TIMEOUT setting.

Enabling HTTP CDP support

Certificate revocation can be obtained through HTTP servers identified within a certificate's CDP extension. The HTTP server provides the revocation information in the form of a CRL.

To enable the use of the certificate's CDP extension when the extension is present in the certificate being validated and for retrieving CRLs from an HTTP server, the `GSK_HTTP_CDP_ENABLE` setting must be set to ON. Note that in order to use the HTTP CDP extension, the extension must contain at least one identifying HTTP entry.

When processing the possible values in the CDP extension, an attempt to contact each HTTP server is tried and processing stops with the first server that is able to be successfully contacted. The HTTP response contains a CRL that is used to determine the revocation status of the certificate.

The following HTTP server and CRL characteristics can be tailored to your environment:

HTTP CDP CRL caching

CRLs retrieved via HTTP can be cached in the application's address space to save the processing time required to contact an HTTP server. HTTP CDP CRL caching is enabled by default when HTTP CDP CRL revocation checking is enabled. The maximum number of CRLs that are allowed to be stored in the HTTP CDP CRL cache is indicated by the `GSK_HTTP_CDP_CACHE_SIZE` setting, which has a default of 32. To have the HTTP server contacted for every validation, `GSK_HTTP_CDP_CACHE_SIZE` must be set to 0.

CRLs are only cached when they contain a `nextUpdate` time. No `nextUpdate` time indicates the CRL is only valid at the time it was provided. The `nextUpdate` time is used as the expiration time of the CRL in the cache. If the HTTP response includes the `max-age` option in the `Cache-Control` directive, the expiration time of the CRL from the cache is either the `nextUpdate` time or `max-age` plus the current time, whichever is earlier.

If the HTTP response contains the `Expires` directive and the `max-age` option in the `Cache-Control` directive is not specified, the expiration time of the OCSP response in the cache is either the `nextUpdate` time or the `Expires` time, whichever is earlier.

When a new CRL is being added to the HTTP CDP CRL cache, expired CRLs are removed. If the cache reaches its size limit, an entry is removed from the cache and the newly obtained CRL is added. If during the validation process, an attempt to use a cached CRL detects the entry is expired, the CRL is removed from the cache and the HTTP server is contacted to obtain an updated CRL.

If storing large CRLs in the HTTP CDP CRL cache is a concern, the `GSK_HTTP_CDP_CACHE_ENTRY_MAXSIZE` can be used to limit the byte size of these CRLs. By default, the `GSK_HTTP_CDP_CACHE_ENTRY_MAXSIZE` setting is set to 0, which indicates there is no limit on the size of a CRL stored in the HTTP CDP CRL cache.

HTTP proxy server

To allow the HTTP request and response to be routed via a proxy HTTP server, `GSK_HTTP_CDP_PROXY_SERVER_NAME` and `GSK_HTTP_CDP_PROXY_SERVER_PORT` must be set.

Response size

The HTTP response consists of the HTTP response directives and the CRL. CRLs can vary in size from being very small to extremely large depending

Writing and building a z/OS System SSL application

upon how many non-expired certificates have been revoke by the CA. The default maximum HTTP response size is 204800 (200K). The size can be overridden through the `GSK_HTTP_CDP_MAX_RESPONSE_SIZE` setting.

Response timeout

When communicating with the HTTP server, by default, the amount of time to wait for the response is set to 15 seconds. The timeout can be overridden through the `GSK_HTTP_CDP_RESPONSE_TIMEOUT` setting.

Enabling LDAP CRL support

Certificate revocation can be obtained through an LDAP server. The LDAP server provides the revocation information in the form of a CRL.

To enable the use of an LDAP server, `GSK_LDAP_SERVER` must be set to the IP address or the hostname of the LDAP server and optionally, the `GSK_LDAP_PORT` must be set to the port that the LDAP server is listening to for requests. When no port is provided, port 389 is used.

When checking the revocation status of a certificate, the x.500 directory name within the CDP extension is used to identify the CRL to be retrieved from the LDAP server. If the CDP extension is not present or does not contain an x.500 directory name, the issuer name within the certificate is used to identify the CRL. The returned CRL is used to determine the certificate revocation status.

The following LDAP server and CRL characteristics can be tailored to your environment:

Response timeout

The time to wait in seconds for a response from the LDAP server can be overridden by specifying `GSK_LDAP_RESPONSE_TIMEOUT`.

The default is 15 seconds.

LDAP CRL caching

CRLs retrieved from LDAP can be cached in the application's address space to save the processing time required to contact the LDAP server. LDAP CRL caching is enabled by default when LDAP CRL revocation checking is enabled.

When LDAP CRL support is enabled, LDAP basic CRL caching is automatically enabled. LDAP basic CRL caching allows for:

- All valid retrieved CRLs to be placed into the cache.
- An unlimited number of CRLs can reside in the cache.
`GSK_CRL_CACHE_SIZE` can be specified to limit the number of entries allowed in the cache.
- CRLs stay in the cache for a maximum of 24 hours. Every 24 hours after the first CRL is added to the cache, the entire cache is emptied.
`GSK_CRL_CACHE_TIMEOUT` can be specified to change the number of hours that the CRLs remain in the cache.
- Temporary CRLs are placed into the cache when the LDAP server does not contain the CRL. `GSK_CRL_CACHE_TEMP_CRL` can be specified to disable temporary CRL caching.

LDAP extended CRL cache support provides the ability to enhance the CRL caching capabilities. To enable LDAP extended CRL caching support, `GSK_CRL_CACHE_EXTENDED` must be set to ON. LDAP extended caching support allows the following:

- Valid retrieved CRLs are eligible to be cached when they contain a nextUpdate or expiration time that is greater than the current time.
- CRLs are allowed to expire in the cache at different times because the nextUpdate or expiration time controls their expiration time. When the cache is updated with a new CRL, any expired CRLs are removed.
- The default for GSK_CRL_CACHE_CACHE_SIZE is 32 CRLs in the cache at one time. If the size is about to be exceeded, the CRL with the time closest to expiration is removed from the cache.
- By default, temporary CRLs are not added to the cache when the LDAP server does not contain the CRL. GSK_CRL_CACHE_TEMP_CRL can be specified to enable temporary CRL caching. If enabled, the lifetime of these temporary CRL entries is controlled by the GSK_CRL_CACHE_TEMP_CRL_TIMEOUT setting, which defaults to 24 hours.

For both basic and LDAP extended CRL cache support, the GSK_CRL_CACHE_ENTRY_MAXSIZE indicates the maximum size in bytes of a CRL that is allowed to be stored in the cache. The default is 0, which means that the CRL size is unlimited.

If LDAP basic CRL cache support is enabled, LDAP CRL caching can be disabled by setting GSK_CRL_CACHE_SIZE to 0 or GSK_CRL_CACHE_TIMEOUT to 0. If LDAP extended CRL cache support is enabled, LDAP CRL caching can be disabled by setting GSK_CRL_CACHE_SIZE to 0.

Revocation source checking (order of precedence)

If multiple revocation sources are enabled (OCSP, HTTP CDP, and LDAP) in the System SSL environment, there is an order of precedence that is used when checking for certificate revocation information. Use the GSK_OCSP_URL_PRIORITY setting to specify the order that the dedicated OCSP responder and OCSP responders in the AIA extension are checked for certificate revocation information.

- To have the dedicated OCSP responder checked first, set GSK_OCSP_URL_PRIORITY to ON. This is the default.
- To have the AIA extension checked first, set GSK_OCSP_URL_PRIORITY to OFF.

The GSK_AIA_CDP_PRIORITY setting specifies the order that the AIA and CDP extensions are checked for certificate revocation information. To have the OCSP responders (dedicated OCSP responder or OCSP responders in the AIA extension) checked before the HTTP servers specified in the CDP extension, set GSK_AIA_CDP_PRIORITY to ON. To have the HTTP servers specified in the CDP extension checked before the OCSP responders, set GSK_AIA_CDP_PRIORITY to OFF. By default, this setting is set to ON.

Revocation security enforcement

The GSK_REVOCATION_SECURITY_LEVEL setting specifies the level of security to be used when contacting an OCSP responder or an HTTP server specified in the CDP extension. When set to LOW, it indicates that certificate validation does not fail if the OCSP responder or the HTTP server cannot be contacted. When set to MEDIUM, it indicates that certificate validation fails if all OCSP responders and HTTP servers are not contactable, or if they are contactable, a valid OCSP response or CRL must be returned. When set to HIGH, it indicates that certificate validation fails if revocation information cannot be contained from any of the specified sources.

Writing and building a z/OS System SSL application

If GSK_LDAP_SERVER is specified, it is always checked last for certificate revocation information if either OCSP or HTTP CDP is enabled. To enable fallback to LDAP (if the OCSP responders or HTTP servers cannot be contacted), the GSK_REVOCATION_SECURITY_LEVEL option must be set to LOW. This then allows contact to be made to the LDAP server specified in the GSK_LDAP_SERVER option. The GSK_CRL_SECURITY_LEVEL option specifies the level of security to be used when contacting an LDAP server and has three settings: LOW, MEDIUM, and HIGH. For more information about the LDAP security level, see the information about GSK_CRL_SECURITY_LEVEL in Appendix A, "Environment variables," on page 667.

Revocation examples

The following examples depict some typical environments that can be configured in System SSL and indicate how the revocation sources are checked for revocation information.

Example 1

Certificate validation uses revocation information from either a dedicated OCSP responder or OCSP responders specified in the certificate's AIA extension. The dedicated OCSP responder is attempted first and if not contactable, the OCSP responders in the AIA extension are used. If none of the specified OCSP responders can be contacted, certificate validation fails. The dedicated OCSP URL is 127.0.0.1

```
GSK_OCSP_URL=http://127.0.0.1
GSK_OCSP_ENABLE=ON
GSK_REVOCATION_SECURITY_LEVEL=MEDIUM
```

Example 2

Certificate validation uses revocation information provided through the AIA and CDP extensions within the certificate. The AIA contains any OCSP responders to be used and the CDP contains any HTTP servers to be used. The OCSP responders within the AIA extension are checked first. If the OCSP responders cannot be contacted, the HTTP servers within the CDP extension are used. If there are no OCSP responders and HTTP servers or none of them can be contacted, the certificate is considered not revoked.

```
GSK_OCSP_ENABLE=ON
GSK_HTTP_CDP_ENABLE=ON
GSK_AIA_CDP_PRIORITY=ON
GSK_REVOCATION_SECURITY_LEVEL=LOW
```

Example 3

Certificate validation uses revocation information provided through the dedicated OCSP responder and the AIA and CDP extensions within the certificate. The revocation providers are checked in the following order:

1. HTTP servers within the CDP extensions.
2. OCSP responders in the AIA extensions.
3. Dedicated OCSP responder.

If none of the OCSP responders or HTTP servers can be contacted, certificate validation fails.

```
| GSK_OCSP_URL=http://127.0.0.1  
| GSK_OCSP_ENABLE=ON  
| GSK_OCSP_URL_PRIORITY=OFF  
| GSK_HTTP_CDP_ENABLE=ON  
| GSK_AIA_CDP_PRIORITY=OFF  
| GSK_REVOCATION_SECURITY_LEVEL=MEDIUM
```

Example 4

Certificate validation uses revocation information provided by the certificate's CDP extension and a LDAP server. The CDP extension is checked first and if the HTTP server cannot be contacted, the LDAP server is used. If the LDAP server is used, a CRL must be available. If no revocation information is retrieved, the certificate is considered revoked.

```
| GSK_HTTP_CDP_ENABLE=ON  
| GSK_LDAP_SERVER=127.0.0.1  
| GSK_LDAP_USER=cn=admin  
| GSK_LDAP_PASSWORD=secret  
| GSK_CRL_SECURITY_LEVEL=HIGH  
| GSK_REVOCATION_SECURITY_LEVEL=LOW
```

Note: GSK_REVOCATION_SECURITY_LEVEL controls the processing characteristics of the CDP extension. LOW allows processing to continue to the LDAP server. GSK_CRL_SECURITY_LEVEL controls the processing characteristics of the LDAP server.

Writing and building a z/OS System SSL application

Chapter 6. Migrating from deprecated SSL interfaces

In Version 1 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". It is suggested that new application programs do not use the deprecated SSL interface. For application programs that currently use 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.
- 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_secure_socket_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_environment_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()` is issued. You may use `gsk_attribute_get_cert_info()`, if you prefer.
- Note that all of the error return values are renamed and renumbered. Program logic must 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 55 for more information about updating your application programs.

The deprecated APIs included in Chapter 9, “Deprecated Secure Socket Layer (SSL) APIs,” on page 487 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 about X.509 certificates and the Secure Sockets Layer protocol. System SSL only supports the PKCS versions that are indicated in the following list. Make sure that you select the appropriate version of the document on the website.

Note: Copies of ANSI standards can be purchased from the American National Standards Institute (ANSI) web page at www.ansi.org.

- ANSI: *ANSI X9.31 - 1998 Digital Certificates Using Reversible Public Key Cryptography for the Financial Services Industry*
- ANSI: *ANSI X9.62 - Elliptic Curve Digital Signature Algorithm*
- FIPS 186-2: *Digital Signature Standard (DSS) (1024-bit and less)*
- FIPS 186-3: *Digital Signature Standard (DSS) (1024-bit and greater)*
- PKCS #1, Version 2.1: *RSA Encryption Standard*
- PKCS #3, Version 1.4: *Diffie-Hellman Key Agreement Standard*
- PKCS #5, Version 2.0: *Password-based Encryption*
- PKCS #7, Version 1.5 and 1.6: *Cryptographic Message Syntax*
- PKCS #8, Version 1.2: *Private Key Information Syntax*
- PKCS #10, Version 1.7: *Certification Request*
- PKCS #12, Version 1.0: *Personal Information Exchange*
- RFC 2246: *The TLS Protocol Version 1.0*
- RFC 2253: *UTF-8 String Representation of Distinguished Names*
- RFC 2279: *UTF-8, a transformation format of ISO 10646*
- RFC 2459: *Internet x.509 Public Key Infrastructure Certificate and CRL Profile*
- RFC 2560: *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*
- RFC 2587: *PKIX LDAP Version 2 Schema*
- RFC 2631: *Diffie-Hellman Key Agreement Method*
- RFC 3268: *Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)*

- RFC 3280: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*
- RFC 3852: *Cryptographic Message Syntax (CMS) (Key transport only)*
- RFC 4346: *The Transport Layer Security (TLS) Protocol Version 1.1*
- RFC 4366: *Transport Layer Security (TLS) Extensions (Server Name Indication, Maximum Fragment Length and Truncated HMAC)*
- RFC 4492: *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)*
- RFC 5116: *An Interface and Algorithms for Authenticated Encryption*
- RFC 5246: *The Transport Layer Security (TLS) Protocol Version 1.2*
- RFC 5280: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*
- RFC 5288: *AES Galois Counter Mode (GCM) Cipher Suites for TLS*
- RFC 5289: *TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)*
- RFC 5430: *Suite B Profile for Transport Layer Security (TLS)*
- RFC 5746: *Transport Layer Security (TLS) Renegotiation Indication Extension*
- RFC 5480: *Elliptic Curve Cryptography Subject Public Key Information*

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 “gsk_attribute_get_buffer()” on page 60)
- **gsk_attribute_get_cert_info()** (see “gsk_attribute_get_cert_info()” on page 65)
- **gsk_attribute_get_data()** (see “gsk_attribute_get_data()” on page 70)
- **gsk_attribute_get_enum()** (see “gsk_attribute_get_enum()” on page 72)
- **gsk_attribute_get_numeric_value()** (see “gsk_attribute_get_numeric_value()” on page 79)
- **gsk_attribute_set_buffer()** (see “gsk_attribute_set_buffer()” on page 82)
- **gsk_attribute_set_callback()** (see “gsk_attribute_set_callback()” on page 87)
- **gsk_attribute_set_enum()** (see “gsk_attribute_set_enum()” on page 92)
- **gsk_attribute_set_numeric_value()** (see “gsk_attribute_set_numeric_value()” on page 102)
- **gsk_attribute_set_tls_extensions()** (see “gsk_attribute_set_tls_extension()” on page 108)
- **gsk_environment_close()** (see “gsk_environment_close()” on page 111)
- **gsk_environment_init()** (see “gsk_environment_init()” on page 112)
- **gsk_environment_open()** (see “gsk_environment_open()” on page 114)
- **gsk_free_cert_data()** (see “gsk_free_cert_data()” on page 121)
- **gsk_get_all_cipher_suites()** (see “gsk_get_all_cipher_suites()” on page 122)
- **gsk_get_cert_by_label()** (see “gsk_get_cert_by_label()” on page 123)
- **gsk_get_cipher_suites()** (see “gsk_get_cipher_suites()” on page 128)
- **gsk_get_ssl_vector()** (see “gsk_get_ssl_vector()” on page 129)
- **gsk_get_update()** (see “gsk_get_update()” on page 130)
- **gsk_list_free()** (see “gsk_list_free()” on page 131)
- **gsk_secure_socket_close()** (see “gsk_secure_socket_close()” on page 132)
- **gsk_secure_socket_init()** (see “gsk_secure_socket_init()” on page 133)
- **gsk_secure_socket_misc()** (see “gsk_secure_socket_misc()” on page 142)

- **gsk_secure_socket_open()** (see “gsk_secure_socket_open()” on page 144)
- **gsk_secure_socket_read()** (see “gsk_secure_socket_read()” on page 145)
- **gsk_secure_socket_shutdown()** (see “gsk_secure_socket_shutdown()” on page 148)
- **gsk_secure_socket_write()** (see “gsk_secure_socket_write()” on page 150)
- **gsk_strerror()** (see “gsk_strerror()” on page 152)

`gsk_attribute_get_buffer()`

`gsk_attribute_get_buffer()`

Gets the value of an attribute buffer.

Format

```
#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 run time 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_INSUFFICIENT_STORAGE`]

Insufficient storage is available.

[`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 run time and must not be released by the application. The address remains 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_CLIENT_ECURVE_LIST

Returns the list of elliptic curve specifications supported by the client as a string consisting of 4-character decimal values.

GSK_CLIENT_ECURVE_LIST may be specified for an SSL environment or an SSL connection. The elliptic curve specifications are used by the client to guide the server as to which elliptic curves can be used when using cipher suites that use Elliptic Curve Cryptography for the TLS V1.0 or higher protocols. See Table 22 on page 695 for a list of valid 4-character elliptic curve specifications.

GSK_CONNECT_CIPHER_SPEC

Returns the cipher specification selected for an initialized connection.

When using the SSL V2 protocol the cipher specification will be returned as a single character. For other protocols the cipher specification may be returned as either a 2-character or 4-character cipher depending on the setting in GSK_V3_CIPHERS. See Table 18 on page 687 for a list of valid SSL V2 cipher specifications. See Table 19 on page 687 and Table 20 on page 691 for a list of valid 2-character and 4-character cipher specifications for the SSL V3 and TLS protocols.

GSK_CONNECT_SEC_TYPE

Returns the security protocol for an initialized connection. The value will be "SSLV2", "SSLV3", "TLSV1", "TLSV1.1", or "TLSV1.2" depending upon the protocol selected during the SSL handshake. GSK_CONNECT_SEC_TYPE may be specified only for an SSL connection.

GSK_HTTP_CDP_PROXY_SERVER_NAME

Returns the DNS name or IP address of the HTTP proxy server for HTTP CDP CRL retrieval. GSK_HTTP_CDP_PROXY_SERVER_NAME may be specified only for an SSL environment

GSK_KEYRING_FILE

Returns the name of the key database file, PKCS #12 file, SAF key ring or z/OS PKCS #11 token. A key database or PKCS #12 file is used if a database password is defined using either an environment variable or the **gsk_attribute_set_buffer()** routine. When a stash file is defined, a key database file is used.

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 or PKCS #12 file. 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_attribute_get_buffer()

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_OCSP_PROXY_SERVER_NAME

Returns the DNS name or IP address of the OCSP proxy server. GSK_OCSP_PROXY_SERVER_NAME may be specified only for an SSL environment.

GSK_OCSP_REQUEST_SIGALG

Returns the hash and signature algorithm pair to be used to sign OCSP requests as a string consisting of a 4-character value. See Table 23 on page 695 for a list of valid 4-character signature algorithm pairs specifications. GSK_OCSP_REQUEST_SIGALG may be specified only for an SSL environment.

GSK_OCSP_REQUEST_SIGKEYLABEL

Returns the certificate label of the key used to sign OCSP requests. GSK_OCSP_REQUEST_SIGKEYLABEL may be specified only for an SSL environment.

GSK_OCSP_URL

Returns the URL of the OCSP responder. GSK_OCSP_URL may be specified only for an SSL environment.

GSK_PEER_ID

Returns the Base64-encoded version of the cached session peer ID. GSK_PEER_ID may be specified only for an SSL connection and is only applicable for a client SSL V3, TLS V1.0, or higher connection when GSK_ENABLE_CLIENT_SET_PEER_ID is ON.

When the SSL connection is not initialized, the GSK_PEER_ID returned is either the session ID specified on a previous **gsk_attribute_set_buffer()** invocation or NULL.

When the SSL connection is initialized, the GSK_PEER_ID that is returned is either the peer ID data specified on a previous **gsk_attribute_set_buffer()** invocation or the Base64-encoded version of the peer ID data and consists of displayable characters.

When the SSL connection is initialized, the peer ID that is returned can be used as input to the **gsk_attribute_set_buffer()** function to identify the cached session information to be used for a subsequent connection.

For more information about using the GSK_PEER_ID, see Specifying a cached session in the **gsk_secure_socket_init()** usage section.

GSK_SID_VALUE

Returns the Base64-encoded version of the session identifier. GSK_SID_VALUE may be specified only for an SSL connection.

When the SSL connection is not initialized, the GSK_SID_VALUE that is returned is either the session ID specified on a previous **gsk_attribute_set_buffer()** invocation or NULL.

When the SSL connection is initialized, the GSK_SID_VALUE that is returned is either the session ID specified on a previous

gsk_attribute_set_buffer() invocation or the Base64-encoded version of the session identifier and consists of displayable characters.

GSK_SID_VALUE can be used as input to the **gsk_attribute_set_buffer()** function to identify the session information to be used for a subsequent server SSL V3, TLS V1.0, or higher connection.

For more information about using the GSK_SID_VALUE, see Specifying a cached session in the **gsk_secure_socket_init()** usage section.

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 or higher protocols which allow 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 contains a list of server names with each server name preceded by a 1-byte name type and a 2-byte field (in large endian format) containing the length of the server name. The name type always contains X'00' to indicate that 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_SUITE_B_CIPHER_SPECS

Returns the Suite B cipher specifications configured for the environment as a string consisting of 4-character values. GSK_SUITE_B_CIPHER_SPECS may be specified for an SSL environment after the environment has been initialized. See Table 9 on page 45 for a list of valid suite B cipher specifications.

GSK_TLS_SIG_ALG_PAIRS

Returns the list of hash and signature algorithm pairs set by the client or server as a string consisting of 1 or more 4-character values. GSK_TLS_SIG_ALG_PAIRS may be specified for an SSL environment or an SSL connection. The signature algorithm pair specifications are used by the client and server to show which signature/hash algorithm combinations are supported for digital signatures. Signature algorithm pair specification only has relevance for sessions using TLS V1.2 or higher protocols. See Table 23 on page 695 for a list of valid 4-character signature algorithm pairs specifications.

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. See Table 18 on page 687 for a list of valid SSL v2 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. The SSL V3 cipher specifications are used for the SSL V3, TLS V1.0, or higher protocols. See Table 19 on page 687 for a list of valid 2-character cipher specifications.

gsk_attribute_get_buffer()

GSK_V3_CIPHER_SPECS_EXPANDED

Returns the SSL V3 cipher specifications as a string consisting of 4-character values. `GSK_V3_CIPHER_SPECS_EXPANDED` 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 higher protocols. See Table 20 on page 691 for a list of valid 4-character cipher specifications.

Related Topics

`gsk_attribute_set_buffer()` on page 82

`gsk_environment_open()` on page 114

`gsk_secure_socket_open()` on page 144

gsk_attribute_get_cert_info()

Returns certificate information following an SSL handshake.

Format

```
#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_attribute_get_cert_info()

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 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 run time.

- C - Country
- CN - Common name
- DC - Domain component
- DNQUALIFIER - Distinguished name qualifier
- EMAIL - email address
- GENERATIONQUALIFIER - Generation qualifier
- GIVENNAME - Given name
- INITIALS - Initials
- L - Locality
- MAIL - RFC 822 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 email 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 run time.

- C - Country
- CN - Common name
- DC - Domain component
- DNQUALIFIER - Distinguished name qualifier
- EMAIL - email address
- GENERATIONQUALIFIER - Generation qualifier
- GIVENNAME - Given name
- INITIALS - Initials
- L - Locality
- MAIL - RFC 822 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 email address (EMAIL)

CERT_ISSUER_GENERATIONQUALIFIER

Issuer generation qualifier (GENERATIONQUALIFIER)

CERT_ISSUER_GIVENNAME

Issuer given name (GIVENNAME)

gsk_attribute_get_cert_info()

CERT_ISSUER_INITIALS
Issuer initials (INITIALS)

CERT_ISSUER_LOCALITY
Issuer locality (L)

CERT_ISSUER_MAIL
Issuer RFC 822 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 RFC 822 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_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\(\)" on page 133](#)

["gsk_free_cert_data\(\)" on page 121](#)

`gsk_attribute_get_data()`

Returns information related to a certificate request.

Format

```
#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_ERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[GSK_ERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[GSK_ERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[GSK_ERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[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 for certificates signed by a certification authority that is in the list provided by the server. A database entry is included in the list only if it has both a certificate and a private key. The labels are associated with certificates read in from a key database, PKCS #12 file, SAF key ring, or PKCS #12 token when the SSL environment was established (**gsk_environment_init()**). If executing in FIPS mode, the list only includes labels that can be used in FIPS mode. If using the TLS V1.2 protocol, the list includes only those certificates that use the key and signature algorithms supported by the server. The **gsk_list_free()** routine should be called to release the list when it is no longer needed.

GSK_DATA_ID_SERVER_ISSUERS

Returns a list of distinguished names of certification authorities provided by the server in the certificate request. The **gsk_list_free()** routine should be called to release the list when it is no longer needed.

Related Topics

[“gsk_list_free\(\)” on page 131](#)

`gsk_attribute_get_enum()`

Gets an enumerated value.

Format

```
#include <gskssl.h>
```

```
gsk_status gsk_attribute_get_enum (
    gsk_handle          ssl_handle,
    GSK_ENUM_ID        enum_id,
    GSK_ENUM_VALUE *   enum_value)
```

Parameters

ssl_handle

Specifies an SSL environment handle that is returned by `gsk_environment_open()` or an SSL connection handle that is returned by `gsk_secure_socket_open()`.

enum_id

Specifies the enumeration identifier.

enum_value

Returns the enumeration value.

Results

The function return value is 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 is closed or the SSL connection is established.

Usage

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

These enumeration identifiers are supported:

GSK_AIA_CDP_PRIORITY

Returns **GSK_AIA_CDP_PRIORITY_ON** to indicate that the AIA extension and **GSK_OCSP_URL** will be queried before examining the CDP extension.

Returns **GSK_AIA_CDP_PRIORITY_OFF** to indicate that the HTTP URI values specified in the CDP extension will be contacted before attempting to contact the OCSP responders in the AIA extension or the OCSP responder specified in **GSK_OCSP_URL**.

GSK_AIA_CDP_PRIORITY can be specified only for an SSL environment.

GSK_CERT_VALIDATE_KEYRING_ROOT

Returns **GSK_CERT_VALIDATE_KEYRING_ROOT_ON** if SAF key ring certificates must be validated to the root CA certificate. Returns

GSK_CERT_VALIDATE_KEYRING_ROOT_OFF if SAF key ring certificates are only validated to the trust anchor certificate. If a sole intermediate certificate is found in a SAF key ring and the next issuer is not found in the same SAF key ring, the intermediate certificate acts as a trust anchor and the certificate chain is considered complete.

GSK_CERT_VALIDATE_KEYRING_ROOT can only be specified for an SSL environment.

GSK_CERT_VALIDATION_MODE

Returns GSK_CERT_VALIDATION_MODE_2459 if certificate validation is based on the RFC 2459 method, GSK_CERT_VALIDATION_MODE_3280 if certificate validation is based on the RFC 3280 method, and GSK_CERT_VALIDATION_MODE_5280 if certificate validation is based on the RFC 5280 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 is configured to allow client connections where client authentication is requested and the client failed to supply an X.509 certificate. Returns GSK_CLIENT_AUTH_NOCERT_ALERT_ON if the SSL server application is configured to terminate client connections where client authentication is requested and the client failed to supply an X.509 certificate. GSK_CLIENT_AUTH_ALERT can be specified only for an SSL environment.

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_CACHE_TEMP_CRL

Returns GSK_CRL_CACHE_TEMP_CRL_ON if a temporary LDAP CRL cache entry is added to the LDAP CRL cache when the CRL does not exist on the LDAP server.

Returns GSK_CRL_CACHE_TEMP_CRL_OFF if a temporary LDAP CRL cache entry is not added to the LDAP CRL cache when the CRL does not exist on the LDAP server.

GSK_CRL_CACHE_TEMP_CRL can be specified only for an SSL environment.

GSK_CRL_CACHE_EXTENDED

Returns GSK_CRL_CACHE_EXTENDED_ON to indicate that LDAP extended CRL cache support is enabled.

Returns GSK_CRL_CACHE_EXTENDED_OFF to indicate that LDAP basic CRL cache support is enabled.

GSK_CRL_CACHE_EXTENDED can be specified only for an SSL environment.

GSK_CRL_SECURITY_LEVEL

Returns GSK_CRL_SECURITY_LEVEL_LOW if certificate validation does not fail if the LDAP server cannot be contacted.

gsk_attribute_get_enum()

Returns GSK_CRL_SECURITY_LEVEL_MEDIUM if certificate validation requires the LDAP server to be contactable, but does not require a CRL to be defined.

Returns GSK_CRL_SECURITY_LEVEL_HIGH if certificate validation requires CRL revocation information be provided from the LDAP server.

GSK_ENABLE_CLIENT_SET_PEERID

Returns GSK_ENABLE_CLIENT_SET_PEERID_ON if the use of a cached peer ID was enabled for the current SSL environment via a call to **gsk_attribute_set_enum()** for this enum.

Returns GSK_ENABLE_CLIENT_SET_PEERID_OFF if the use of a cached peer ID was not enabled for the current SSL environment.

GSK_ENABLE_CLIENT_SET_PEERID is only valid for a SSL V3, TLS 1.0, or higher secure connection and is only meaningful if specified for a client connection within an SSL environment.

GSK_EXTENDED_RENEGOTIATION_INDICATOR

Returns GSK_EXTENDED_RENEGOTIATION_INDICATOR_OPTIONAL if renegotiation indication is not required during the initial SSL V3 or TLS handshake. This is the default.

Returns GSK_EXTENDED_RENEGOTIATION_INDICATOR_CLIENT if the client initial handshake is allowed to proceed only if the server indicates support for RFC 5746 renegotiation.

Returns GSK_EXTENDED_RENEGOTIATION_INDICATOR_SERVER if the server initial handshake is allowed to proceed only if the client indicates support for RFC 5746 renegotiation.

Returns GSK_EXTENDED_RENEGOTIATION_INDICATOR_BOTH if the server and client initial handshakes are allowed to proceed only if partner indicates support for RFC 5746 renegotiation.

GSK_EXTENDED_RENEGOTIATION_INDICATOR can only be specified for an SSL environment.

GSK_HTTP_CDP_ENABLE

Returns GSK_HTTP_CDP_ENABLE_ON if certificate revocation checking is using the HTTP URI values in the CDP extension to locate an HTTP server.

Returns GSK_HTTP_CDP_ENABLE_OFF if certificate revocation checking is not using the HTTP URI values in the CDP extension.

GSK_HTTP_CDP_ENABLE can be specified only for an SSL environment.

GSK_OCSP_ENABLE

Returns GSK_OCSP_ENABLE_ON if certificate revocation checking is using the HTTP URI values in the AIA extension to locate an OCSP responder.

Returns GSK_OCSP_ENABLE_OFF if certificate revocation checking is not using the HTTP URI values in the AIA extension.

GSK_OCSP_ENABLE can be specified only for an SSL environment.

GSK_OCSP_NONCE_CHECK_ENABLE

Returns GSK_OCSP_NONCE_CHECK_ENABLE_ON if the nonce in the OCSP response will be verified to ensure it matches the nonce sent in the OCSP request.

Returns GSK_OCSP_NONCE_CHECK_ENABLE_OFF if checking of the nonce in the OCSP response is disabled.

GSK_OCSP_NONCE_CHECK_ENABLE can be specified only for an SSL environment.

GSK_OCSP_NONCE_GENERATION_ENABLE

Returns GSK_OCSP_NONCE_GENERATION_ENABLE_ON if OCSP requests will include a generated nonce.

Returns GSK_OCSP_NONCE_GENERATION_ENABLE_OFF if OCSP nonce generation is disabled.

GSK_OCSP_NONCE_GENERATION_ENABLE can be specified only for an SSL environment.

GSK_OCSP_RETRIEVE_VIA_GET

Returns GSK_OCSP_RETRIEVE_VIA_GET_ON if the HTTP GET request should be used when sending an OCSP request.

Returns GSK_OCSP_RETRIEVE_VIA_GET_OFF if the HTTP request will always be sent via an HTTP POST.

GSK_OCSP_RETRIEVE_VIA_GET can be specified only for an SSL environment.

GSK_OCSP_URL_PRIORITY

Returns GSK_OCSP_URL_PRIORITY_ON if the GSK_OCSP_URL defined responder will first be used and then the responders identified in the AIA extension.

Returns GSK_OCSP_URL_PRIORITY_OFF if the responders identified in the AIA extension will be used first and then the GSK_OCSP_URL defined responder.

GSK_OCSP_URL_PRIORITY can be specified only for an SSL environment.

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_TLSV1_2

Returns GSK_PROTOCOL_TLSV1_2_ON if the TLS Version 1.2 protocol is

gsk_attribute_get_enum()

enabled and GSK_PROTOCOL_TLSV1_2_OFF if the TLS Version 1.2 protocol is not enabled. GSK_PROTOCOL_TLSV1_2 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, GSK_PROTOCOL_USED_TLSV1_1 if the TLS Version 1.1 protocol was used to establish the connection, or GSK_PROTOCOL_USED_TLSV1_2 if the TLS Version 1.2 protocol was used to establish the connection. GSK_NULL is returned if a connection is not established. GSK_PROTOCOL_USED can be specified only for an SSL connection.

GSK_RENEGOTIATION

Returns GSK_RENEGOTIATION_NONE if SSL V3 and TLS handshake renegotiation as a server is disabled, while RFC 5746 renegotiation is allowed. This is the default.

Returns GSK_RENEGOTIATION_DISABLED if SSL V3 and TLS handshake renegotiation, including RFC 5746 renegotiation, is disabled.

Returns GSK_RENEGOTIATION_ALL if SSL V3 and TLS handshake renegotiation as a server is enabled.

Returns GSK_RENEGOTIATION_ABBREVIATED if SSL V3 and TLS abbreviated handshake renegotiation for resuming the current session only is permitted as a server. RFC 5746 renegotiation is also allowed.

GSK_RENEGOTIATION can only be specified for an SSL environment.

GSK_RENEGOTIATION_PEER_CERT_CHECK

Returns GSK_RENEGOTIATION_PEER_CERT_CHECK_OFF if an identity check against the peer's certificate is not performed during renegotiation. This is the default.

Returns GSK_RENEGOTIATION_PEER_CERT_CHECK_ON if a comparison is performed against the peer's certificate to ensure that certificate does not change during renegotiation.

GSK_RENEGOTIATION_PEER_CERT_CHECK can only be specified for an SSL environment.

GSK_REQ_CACHED_SESSION

Returns GSK_ENABLE_CLIENT_SET_PEERID_ON if a particular cached session peer ID is desired for an upcoming SSL V3, TLS 1.0, or higher secure connection.

Returns GSK_ENABLE_CLIENT_SET_PEERID_OFF if cached session reuse is not active.

GSK_ENABLE_CLIENT_SET_PEERID is only meaningful if specified for a client connection.

GSK_REVOCAATION_SECURITY_LEVEL

Returns GSK_REVOCAATION_SECURITY_LEVEL_LOW if certificate validation does not fail if the OCSP responder or HTTP server specified in the URI value of the CDP extension cannot be contacted.

Returns GSK_REVOCATION_SECURITY_LEVEL_MEDIUM if certificate validation requires the OCSP responder or the HTTP server specified in the URI value of the CDP extension to be contactable.

Returns GSK_REVOCATION_SECURITY_LEVEL_HIGH if certificate validation requires revocation information be provided by an OCSP responder or HTTP server.

GSK_REVOCATION_SECURITY_LEVEL can be specified only for an SSL environment.

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 is returned if a connection is not established. GSK_SID_FIRST can be specified only for an SSL connection.

GSK_SUITE_B_PROFILE

Returns the Suite B for TLS profile setting. Returns:

- GSK_SUITE_B_PROFILE_128 if the 128-bit Suite B security profile is being applied by the SSL client or server to TLS sessions.
- GSK_SUITE_B_PROFILE_192 if the 192-bit Suite B security profile is being applied by the SSL client or server to TLS sessions.
- GSK_SUITE_B_PROFILE_ALL if either the 128-bit or 192-bit Suite B security profile is allowed by the SSL client or server for TLS sessions.
- GSK_SUITE_B_PROFILE_OFF if there is no Suite B security profile being applied by the SSL client or server to TLS sessions.

GSK_SUITE_B_PROFILE can only be specified for an SSL environment.

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

Returns GSK_T61_AS_LATIN1_ON if the ISO8859-1 character set is used when converting a string tagged as TELETExSTRING or GSK_T61_AS_LATIN1_OFF if the T.61 character set is 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.

GSK_TLS_CBC_PROTECTION_METHOD

Returns GSK_TLS_CBC_PROTECTION_METHOD_NONE to indicate that no CBC protection is enabled.

Returns

GSK_TLS_CBC_PROTECTION_METHOD_ZEROBYTEFRAGMENT to indicate that zero byte record fragmenting is enabled.

gsk_attribute_get_enum()

| Returns GSK_TLS_CBC_PROTECTION_METHOD_ONEBYTEFRAGMENT
| to indicate that one byte record fragmenting is enabled.
|
| GSK_TLS_CBC_PROTECTION_METHOD can only be specified for an SSL
| environment.

GSK_TLSEXT_MFL

Returns GSK_TLSEXT_MFL_OFF if the "Maximum Fragment Length" type TLS extension is not 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 is negotiated, where the returned value reflects the negotiated maximum fragment length. 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 is negotiated and is in use. Returns GSK_TLSEXT_SNI_OFF if the "Server Name Indication" type TLS extension is not negotiated. GSK_TLSEXT_SNI can be specified only for an SSL connection.

GSK_TLSEXT_THMAC

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

GSK_V3_CIPHERS

| Returns GSK_V3_CIPHERS_CHAR2 if 2 character V3 cipher specifications
| are in use. Returns GSK_V3_CIPHERS_CHAR4 if 4 character V3 cipher
| specifications are in use. GSK_V3_CIPHERS can be specified for an SSL
| environment or an SSL connection.

Related Topics

["gsk_attribute_set_enum\(\)" on page 92](#)

["gsk_environment_open\(\)" on page 114](#)

["gsk_secure_socket_open\(\)" on page 144](#)

gsk_attribute_get_numeric_value()

Gets a numeric value.

Format

```
#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_ENTRY_MAXSIZE

Returns the maximum size in bytes of a CRL that is allowed to be stored in the LDAP CRL cache. Any CRLs larger than this size are not cached. `GSK_CRL_CACHE_ENTRY_MAXSIZE` can be specified only for an SSL environment.

GSK_CRL_CACHE_SIZE

Returns the maximum number of CRLs that are allowed to be stored in the LDAP CRL cache. `GSK_CRL_CACHE_SIZE` can be specified only for an SSL environment.

GSK_CRL_CACHE_TEMP_CRL_TIMEOUT

Returns the time in hours that a temporary CRL cache entry resides in the

gsk_attribute_get_numeric_value()

LDAP extended CRL cache. `GSK_CRL_CACHE_TEMP_CRL_TIMEOUT` can be specified only for an SSL environment.

GSK_CRL_CACHE_TIMEOUT

Returns the LDAP basic 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_HTTP_CDP_CACHE_ENTRY_MAXSIZE

Returns the maximum size in bytes of a CRL that is allowed to be stored in the HTTP CDP CRL cache. `GSK_HTTP_CDP_CACHE_ENTRY_MAXSIZE` can be specified only for an SSL environment.

GSK_HTTP_CDP_CACHE_SIZE

Returns the maximum number of CRLs that are allowed to be stored in the HTTP CDP CRL cache. `GSK_HTTP_CDP_CACHE_SIZE` can be specified only for an SSL environment.

GSK_HTTP_CDP_MAX_RESPONSE_SIZE

Returns the maximum size in bytes accepted as a response from an HTTP server when retrieving a CRL. `GSK_HTTP_CDP_MAX_RESPONSE_SIZE` can be specified only for an SSL environment.

GSK_HTTP_CDP_PROXY_SERVER_PORT

Returns the HTTP proxy server port for HTTP CDP CRL retrieval. `GSK_HTTP_CDP_PROXY_SERVER_PORT` can be specified only for an SSL environment.

GSK_HTTP_CDP_RESPONSE_TIMEOUT

Returns the time in seconds to wait for a response from the HTTP server. `GSK_HTTP_CDP_RESPONSE_TIMEOUT` can be specified only for an SSL environment.

GSK_LDAP_RESPONSE_TIMEOUT

Returns the time in seconds to wait for a response from the LDAP server. `GSK_LDAP_RESPONSE_TIMEOUT` can be specified only for an SSL environment.

GSK_LDAP_SERVER_PORT

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

GSK_MAX_SOURCE_REV_EXT_LOC_VALUES

Returns the maximum number of location values that will be contacted for each revocation source when performing validation of a certificate. `GSK_MAX_SOURCE_REV_EXT_LOC_VALUES` can be specified only for an SSL environment.

GSK_MAX_VALIDATION_REV_EXT_LOC_VALUES

Returns the maximum number of location values that will be contacted when performing validation of a certificate. `GSK_MAX_VALIDATION_REV_EXT_LOC_VALUES` can be specified only for an SSL environment.

GSK_OCSP_CLIENT_CACHE_ENTRY_MAXSIZE

Returns the maximum number of OCSP responses or cached certificate statuses that are allowed to be kept in the OCSP response cache for an issuing CA certificate. `GSK_OCSP_CLIENT_CACHE_ENTRY_MAXSIZE` can be specified only for an SSL environment.

| **GSK_OCSP_CLIENT_CACHE_SIZE**

| Returns the maximum number of OCSP responses or cached certificate
| statuses to be kept in the OCSP response cache.
| GSK_OCSP_CLIENT_CACHE_SIZE can only be specified for an SSL
| environment.

| **GSK_OCSP_MAX_RESPONSE_SIZE**

| Returns the maximum size in bytes allowed in a response from an OCSP
| responder. GSK_OCSP_MAX_RESPONSE_SIZE can be specified only for an
| SSL environment.

| **GSK_OCSP_NONCE_SIZE**

| Returns the size in bytes of the nonce that will be sent in OCSP requests.
| GSK_OCSP_NONCE_SIZE can be specified only for an SSL environment.

| **GSK_OCSP_PROXY_SERVER_PORT**

| Returns the port for the OCSP responder proxy server.
| GSK_OCSP_PROXY_SERVER_PORT can be specified only for an SSL
| environment.

| **GSK_OCSP_RESPONSE_TIMEOUT**

| Returns the time in seconds to wait for a response from the OCSP
| responder. GSK_OCSP_RESPONSE_TIMEOUT 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_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\(\)" on page 102](#)

["gsk_environment_open\(\)" on page 114](#)

["gsk_secure_socket_open\(\)" on page 144](#)

`gsk_attribute_set_buffer()`

Sets the value of an attribute buffer.

Format

```
#include <gskssl.h>
```

```
gsk_status 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 is 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 sets 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).

The values set using this service are treated as independent values. They are not validated with other values set using `gsk_attribute_set_buffer()`,

gsk_attribute_set_enum(), or **gsk_attribute_set_tls_extensions()** APIs until used together to perform a SSL/TLS handshake by calling **gsk_secure_socket_init()**.

If the input *buffer_value* is NULL, the target attribute is set to NULL.

These buffer identifiers are supported:

GSK_CLIENT_ECURVE_LIST

Specifies the list of elliptic curves that are supported by the client as a string consisting of 1 or more 4-character decimal values in order of preference for use. GSK_CLIENT_ECURVE_LIST may be specified for an SSL environment or an SSL connection. The list is used by the client to guide the server as to which elliptic curves are preferred when using ECC-based cipher suites for the TLS V1.0 or higher protocols.

Only NIST recommended curves are able to be specified for the attribute. To use Brainpool standard curves for an SSL connection, the buffer must be reinitialized to NULL using either **gsk_attribute_set_buffer()** or the GSK_CLIENT_ECURVE_LIST environment variable. See Table 22 on page 695 for a list of valid 4-character elliptic curve specifications

GSK_HTTP_CDP_PROXY_SERVER_NAME

Specifies the DNS name or IP address of the HTTP proxy server for HTTP CDP CRL retrieval. GSK_HTTP_CDP_PROXY_SERVER_NAME can be specified only for an SSL environment.

GSK_KEYRING_FILE

Specifies the name of the key database file, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. A key database or PKCS #12 file is used if a database password is defined using either an environment variable or the **gsk_attribute_set_buffer()** routine. When a stash file is defined, a key database file is used. 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 user ID is used if the user ID 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.

A z/OS PKCS #11 token name is specified as *TOKEN*/token-name. *TOKEN* indicates a PKCS #11 token is being specified.

Note: Certificate private keys are not available when using a SAF key ring owned by another user, except for SITE certificates where CONTROL authority is given to IRR.DIGTCERT.GENCERT in the FACILITY class or for user certificates where READ or UPDATE authority is given to *ringOwner.ringName.LST* resource in the RDATAALIB class.

GSK_KEYRING_LABEL

Specifies the label of the key that is used to authenticate the application. The default key is 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_attribute_set_buffer()

GSK_KEYRING_PW

Specifies the password for the key database or PKCS #12 file.
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 is changed if it does not have the correct extension. The GSK_KEYRING_PW value is used instead of the GSK_KEYRING_STASH 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 that is 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_OCSP_PROXY_SERVER_NAME

Specifies the DNS name or IP address of the OCSP proxy server.
GSK_OCSP_PROXY_SERVER_NAME can be specified only for an SSL environment.

GSK_OCSP_REQUEST_SIGALG

Specifies the hash and signature algorithm pair to be used to sign OCSP requests as a string consisting of a 4-character value. See Table 23 on page 695 for a list of valid 4-character signature algorithm pair specifications.
Defaults to RSA with SHA256 ('0401').

Only requests sent to the OCSP responder identified via the GSK_OCSP_URL setting are signed. GSK_OCSP_REQUEST_SIGALG can be specified only for an SSL environment.

GSK_OCSP_REQUEST_SIGKEYLABEL

Specifies the label of the key to be used to sign OCSP requests. OCSP requests are only signed when a key label is specified.

Only requests sent to the OCSP responder identified via the GSK_OCSP_URL setting are signed (not ones selected from a certificate AIA extension). GSK_OCSP_REQUEST_SIGKEYLABEL can be specified only for an SSL environment.

GSK_OCSP_URL

Specifies the URL of an OCSP responder. The OCSP responder is used to obtain certificate revocation status during certificate validation. A certificate does not need an AIA extension if a responder URL is configured using this option.

The value must conform to the definition of an HTTP URL:

```
http_URL = "http:" "://" host [ ":" port ] [ abs_path [ "?" query ] ]
```

where host can be an IPv4 or IPv6 address, or a domain name.

If GSK_OCSP_URL is specified, GSK_OCSP_ENABLE is set ON, and GSK_OCSP_URL_PRIORITY is set ON, the order that the responders are used is GSK_OCSP_URL defined responder first and then the responders identified in the AIA extension.

If GSK_OCSP_URL is specified, GSK_OCSP_ENABLE is set ON, and GSK_OCSP_URL_PRIORITY is set OFF, the order that the responders are used is the responders identified in the AIA extension first and then the GSK_OCSP_URL defined responder.

GSK_OCSP_URL can be specified only for an SSL environment.

GSK_PEER_ID

Specify the GSK_PEER_ID to uniquely identify the cached session to be used to resume an existing session or duplicate an existing session. Use **gsk_attribute_get_buffer()** to initially retrieve the GSK_PEER_ID to be used by **gsk_attribute_set_buffer()**.

GSK_PEER_ID may be specified only for an SSL connection and is only applicable for a client SSL V3, TLS V1.0, or higher connection when SSL environment enumerator GSK_ENABLE_CLIENT_SET_PEERID is ON and V3 session caching is enabled (session cache size and timeout are non-zero).

For more information about using the GSK_PEER_ID, see Specifying a cached session in the **gsk_secure_socket_init()** usage section.

GSK_SID_VALUE

Specify the GSK_SID_VALUE to uniquely identify the cached session to be used to resume an existing session. GSK_SID_VALUE may be specified only for an SSL connection and is only applicable for a server connection when V3 session caching is enabled (session cache and timeout are non-zero). Use **gsk_attribute_get_buffer()** to initially retrieve the GSK_SID_VALUE to be used by **gsk_attribute_set_buffer()**.

For more information about using the GSK_SID_VALUE, see Specifying a cached session in the **gsk_secure_socket_init()** usage section.

GSK_TLS_SIG_ALG_PAIRS

Specifies the list of hash and signature algorithm pair specifications that are supported by the client or server as a string consisting of 1 or more 4-character values in order of preference for use.

GSK_TLS_SIG_ALG_PAIRS may be specified for an SSL environment or an SSL connection. The signature algorithm pair specifications are sent by either the client or server to the session partner to indicate which signature/hash algorithm combinations are supported for digital signatures. Signature algorithm pair specification only has relevance for sessions using TLS V1.2 or higher protocols. See Table 23 on page 695 for a list of valid 4-character signature algorithm pair specifications.

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 run time 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_attribute_set_buffer()

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. See Table 18 on page 687 for a list of valid SSL v2 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, or higher protocols. See Table 19 on page 687 for a list of valid 2-character cipher specifications.

GSK_V3_CIPHER_SPECS_EXPANDED

Specifies the SSL V3 cipher specifications as a string consisting of 1 or more 4-character values. GSK_V3_CIPHER_SPECS_EXPANDED 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, or higher protocols. See Table 20 on page 691 for a list of valid 4-character cipher specifications. Applications wanting to use cipher suites that use Elliptic Curve Cryptography must set an appropriate cipher specification in GSK_V3_CIPHER_SPECS_EXPANDED.

Related Topics

“gsk_attribute_get_buffer()” on page 60

“gsk_environment_open()” on page 114

“gsk_environment_init()” on page 112

“gsk_secure_socket_open()” on page 144

“gsk_secure_socket_init()” on page 133

gsk_attribute_set_callback()

Sets an SSL callback.

Format

```
#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 is 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 run time. A callback allows the application to replace the default routine used by the SSL run time. 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 that the application is providing a routine to be used during a full handshake 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

gsk_attribute_set_callback()

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.

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.

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

GSK_IO_CALLBACK

Indicates that 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 overrides 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 error code is returned in the *errno* runtime variable. The default routine uses the **recv()** library routine to read data from the network.

```
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 error code is returned in the *errno* runtime variable. The default routine uses the **send()** library routine to write data to the network.

```
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.

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

The routine specified by the *io_setsocketoptions* 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 error code is returned in the *errno* runtime variable. 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.

```
int io_setsocketoptions (
    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.

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

GSK_SESSION_RESET_CALLBACK

Indicates that the application is providing the routines to be called when a session renegotiation has been initiated or completed 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, or higher 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.

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

The *Reset_Complete* routine is called when a session renegotiation has been completed. If session renegotiation does not successfully complete, for example because of renegotiation not being allowed, then the *Reset_Complete* routine is not invoked even though the *Reset_Init* routine was called at the commencement of renegotiation. The *con_handle* parameter is the handle for the SSL connection.

gsk_attribute_set_callback()

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

GSK_SID_CACHE_CALLBACK

Indicates that 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 run time.

```
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_data_buffer * Put (
    gsk_data_buffer *  ssl_session_data,
    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`).

```
void Delete (
    const unsigned char * session_id,
    unsigned int         session_id_length,
    gsk_sslversion       ssl_version)
```

The routine specified by the *FreeDataBuffer* entry is called to release the data buffer returned by the *Get* routine.

```
void FreeDataBuffer (
    gsk_data_buffer *  ssl_session_data)
```

GSK_SNI_CALLBACK

Indicates that the application is providing the routine to allow a server to interrogate a list of server names supplied by the client and select an appropriate key label for use as the server certificate based on the information received from the client. The selected certificate from the key database, PKCS #12 file, key ring or token will be sent to the client as the server certificate during the handshake process. The callback parameter is the address of this routine. The exit routine can obtain the server name list

provided by the client 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 callback routine cannot enforce the required use of the server name indication extension. The failure to select a key label causes a fatal UNRECOGNIZED_NAME alert. To enforce such actions 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 requested 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.

```
int sni_callback (
    gsk_handle    soc_handle)
```

Related Topics

[“gsk_environment_init\(\)” on page 112](#)

[“gsk_secure_socket_init\(\)” on page 133](#)

gsk_attribute_set_enum()

Sets an enumerated value.

Format

```
#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 that is returned by **gsk_environment_open()** or an SSL connection handle that is returned by **gsk_secure_socket_open()**.

enum_id

Specifies the enumeration identifier.

enum_value

Specifies the enumeration value.

Results

The function return value is 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 sets 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).

The values set using this service are treated as independent values. They are not validated with other values set using **gsk_attribute_set_buffer()**, **gsk_attribute_set_enum()**, or **gsk_attribute_set_tls_extensions()** APIs until used together to perform a SSL/TLS handshake by calling **gsk_secure_socket_init()**.

These enumeration identifiers are supported:

GSK_AIA_CDP_PRIORITY

Specifies the priority order that the AIA and CDP extensions are checked for revocation information.

Specify **GSK_AIA_CDP_PRIORITY_ON** to indicate that the AIA extension is processed prior to the CDP extension during certificate revocation

checking. This means that any OCSP responders specified in the AIA extension or the OCSP responder specified in GSK_OCSP_URL are contacted before attempting to contact the HTTP servers specified in the URI values in the CDP extension. This is the default setting.

Specify GSK_AIA_CDP_PRIORITY_OFF to indicate that the CDP extension is queried prior to the AIA extension. This means that HTTP servers in the URI values in the CDP extension are contacted before attempting to contact the OCSP responders in the AIA extension or the OCSP responder specified in GSK_OCSP_URL.

GSK_AIA_CDP_PRIORITY can be specified only for an SSL environment.

GSK_CERT_VALIDATE_KEYRING_ROOT

Specifies the setting of how certificates in a SAF key ring are validated. Specify GSK_CERT_VALIDATE_KEYRING_ROOT_ON if SAF key ring certificates must be validated to the root CA certificate. Specify GSK_CERT_VALIDATE_KEYRING_ROOT_OFF if SAF key ring certificates are only validated to the trust anchor certificate. If a sole intermediate certificate is found in a SAF key ring and the next issuer is not found in the same SAF key ring, the intermediate certificate acts as a trust anchor and the certificate chain is considered complete. By default, SAF key ring certificates are only validated to the trust anchor certificate. This setting does not affect the validation of SSL key database file, PKCS #12 file, and PKCS #11 token certificates as these certificates are always validated to the root CA certificate.

GSK_CERT_VALIDATE_KEYRING_ROOT can be specified only for an SSL environment.

GSK_CERT_VALIDATION_MODE

Specifies the method of certificate validation. RFC 2459, RFC 3280, and RFC 5280 describe differing methods of certificate validation. Specify GSK_CERT_VALIDATION_MODE_2459 if certificate validation according to the RFC 2459 method is required, GSK_CERT_VALIDATION_MODE_3280 if certificate validation according to the RFC 3280 method is required, or GSK_CERT_VALIDATION_MODE_5280 if certificate validation according to the RFC 5280 method is required.

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

GSK_CERT_VALIDATION_MODE can be specified only for an SSL environment.

GSK_CRL_CACHE_EXTENDED

Specifies that LDAP extended CRL cache support is enabled.

Specify GSK_CRL_CACHE_EXTENDED_ON to indicate that LDAP extended CRL cache support is enabled. LDAP extended CRL cache support enables the following support:

- LDAP CRLs are only cached when there is an expiration time present and it is greater than the current time.
- A limit on the maximum number of CRLs that can be stored in the LDAP cache which can be overridden by specifying GSK_CRL_CACHE_SIZE. The default is 32.
- By default, GSK_CRL_CACHE_TEMP_CRL is disabled.

gsk_attribute_set_enum()

| Specify GSK_CRL_CACHE_EXTENDED_OFF to indicate that LDAP basic
| CRL cache support is enabled. When LDAP basic CRL cache support is
| enabled, retrieved LDAP CRLs are only cached if
| GSK_CRL_CACHE_TIMEOUT is greater than 0 and
| GSK_CRL_CACHE_SIZE is set to a non-zero number. LDAP basic CRL
| cache support is the default.

| **Note:** When set to GSK_CRL_CACHE_EXTENDED_ON, the
| GSK_CRL_CACHE_TIMEOUT value is ignored.

| GSK_CRL_CACHE_EXTENDED can be specified only for an SSL
| environment.

GSK_CRL_CACHE_TEMP_CRL

| Specifies if a temporary LDAP CRL cache entry is added to the LDAP CRL
| cache when the CRL does not reside on the LDAP server.

| Specify GSK_CRL_CACHE_TEMP_CRL_ON if a temporary CRL cache
| entry is to be added to the LDAP CRL cache.

| Specify GSK_CRL_CACHE_TEMP_CRL_OFF if a temporary CRL cache
| entry is not to be added to the LDAP CRL cache.

| If a temporary CRL is cached, it will prevent continual attempts to contact
| the LDAP server and will allow connections to be successful.

| GSK_CRL_CACHE_TEMP_CRL can be specified only for an SSL
| environment.

GSK_CRL_SECURITY_LEVEL

| Specifies the level of security to be used when contacting LDAP servers to
| check CRLs for revoked certificates during certificate validation.

| Three levels of security are available:

GSK_CRL_SECURITY_LEVEL_LOW

| Certificate validation does 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 retrieved. This is the default.

GSK_CRL_SECURITY_LEVEL_HIGH

| Certificate validation requires revocation information to be
| provided by the LDAP server.

| GSK_CRL_SECURITY_LEVEL can be specified only for an SSL
| environment.

GSK_CLIENT_AUTH_ALERT

| Specify GSK_CLIENT_AUTH_NOCERT_ALERT_OFF if the SSL server
| application is to allow client connections where client authentication is
| 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 is 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 is 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 is successful. The application can check for this case by calling the **gsk_attribute_get_cert_info()** routine and checking for a NULL return address.

When a client's certificate is being requested, the client can be required to provide a certificate by setting GSK_CLIENT_AUTH_ALERT to GSK_CLIENT_NOCERT_ALERT_ON. If no certificate is received, the requested handshake fails. See “gsk_attribute_set_enum()” on page 92 for more information about the GSK_CLIENT_AUTH_ALERT setting.

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_ENABLE_CLIENT_SET_PEERID

Specify GSK_ENABLE_CLIENT_SET_PEERID_ON to enable the use of a cached GSK_PEER_ID for SSL V3, TLS 1.0, or higher client connections. GSK_ENABLE_CLIENT_SET_PEERID can be specified only for an SSL environment and is only applicable for client connections. GSK_ENABLE_CLIENT_SET_PEERID_OFF is the default setting.

GSK_ENABLE_CLIENT_SET_PEERID_ON limits the number of full client handshakes that can be cached over the lifetime of the SSL environment to a maximum number of 4.29 billion. If this maximum number is reached, all new SSL connections that are not using a cached GSK_PEER_ID will result in full handshakes and will not add entries into the session cache. Also, if the maximum number is reached, reusing a cached GSK_PEER_ID is allowed as long as the GSK_PEER_ID can still be located in the cache.

GSK_EXTENDED_RENEGOTIATION_INDICATOR

Specify GSK_EXTENDED_RENEGOTIATION_INDICATOR_OPTIONAL to not require the renegotiation indicator during initial handshake. This is the default.

Specify GSK_EXTENDED_RENEGOTIATION_INDICATOR_CLIENT to allow the client initial handshake to proceed only if the server indicates support for RFC 5746 Renegotiation.

Specify GSK_EXTENDED_RENEGOTIATION_INDICATOR_SERVER to allow the server initial handshake to proceed only if the client indicates support for RFC 5746 Renegotiation.

Specify GSK_EXTENDED_RENEGOTIATION_INDICATOR_BOTH to allow the server and client initial handshakes to proceed only if partner indicates support for RFC 5746 Renegotiation.

GSK_EXTENDED_RENEGOTIATION_INDICATOR can be specified only for an SSL environment.

gsk_attribute_set_enum()

GSK_HTTP_CDP_ENABLE

Specifies whether the HTTP URIs within the CDP extension are to be utilized for certificate revocation checking.

Specify **GSK_HTTP_CDP_ENABLE_OFF** to indicate that certificate revocation checking with the HTTP URI values in the CDP is not enabled. This is the default.

Specify **GSK_HTTP_CDP_ENABLE_ON** to indicate that certificate revocation checking with the HTTP URI values in the CDP extension is enabled.

GSK_HTTP_CDP_ENABLE can be specified only for an SSL environment.

GSK_OCSP_ENABLE

Specifies whether the AIA extensions are to be used for certificate revocation checking.

Specify **GSK_OCSP_ENABLE_ON** to activate certificate revocation checking using the HTTP URI values in the certificate's AIA extension.

Specify **GSK_OCSP_ENABLE_OFF** to disable use of the AIA extension. This is the default.

If **GSK_OSCP_URL** is specified, **GSK_OCSP_ENABLE** is set to **ON**, and **GSK_OCSP_URL_PRIORITY** is set to **ON**, then the order the responders are used is **GSK_OSCP_URL** defined responder first and then the responders identified in the AIA extension. If **GSK_OSCP_URL** is specified, **GSK_OCSP_ENABLE** is set to **ON** and **GSK_OCSP_URL_PRIORITY** is set to **OFF**, then the order that responders are used is the responders identified in the AIA extension first and then the **GSK_OSCP_URL** defined responder.

GSK_OCSP_ENABLE can be specified only for an SSL environment.

GSK_OCSP_NONCE_CHECK_ENABLE

Specifies if OCSP response nonce checking is on or off.

Specify **GSK_OCSP_NONCE_CHECK_ENABLE_ON** to have the nonce in the OCSP response verified to ensure it matches the nonce sent in the OCSP request.

Note: Setting **GSK_OCSP_NONCE_CHECK_ENABLE_ON** also implies that **GSK_OCSP_NONCE_GENERATION_ENABLE_ON** is also set to **ON**.

Specify **GSK_OCSP_NONCE_CHECK_ENABLE_OFF** to disable checking of the nonce in the OCSP response. This is the default.

GSK_OCSP_NONCE_CHECK_ENABLE can be specified only for an SSL environment.

GSK_OCSP_NONCE_GENERATION_ENABLE

Specifies whether the OCSP request includes a generated nonce.

Specify **GSK_OCSP_NONCE_GENERATION_ENABLE_ON** to enable nonce generation.

Specify **GSK_OCSP_NONCE_GENERATION_ENABLE_OFF** to disable OCSP nonce generation. This is the default.

GSK_OCSP_NONCE_GENERATION_ENABLE can be specified only for an SSL environment.

GSK_OCSP_RETRIEVE_VIA_GET

Specifies whether the HTTP request to the OCSP responder is sent using the HTTP Get Method or the HTTP Post method.

Specify **GSK_OCSP_RETRIEVE_VIA_GET_ON** to indicate that the HTTP GET method should be used when sending an OCSP request whose total request size after Base64 encoding is less than 255 bytes. This option allows HTTP caching on the OCSP responder when the responder has been enabled for caching.

Specify **GSK_OCSP_RETRIEVE_VIA_GET_OFF** to indicate the HTTP request should always be sent via an HTTP Post method. This is the default.

GSK_OCSP_RETRIEVE_VIA_GET can be specified only for an SSL environment.

GSK_OCSP_URL_PRIORITY

Specifies the order of precedence for contacting the OCSP responder locations if both **GSK_OCSP_URL** and **GSK_OCSP_ENABLE** are active.

Specify **GSK_OCSP_URL_PRIORITY_ON** to indicate that the **GSK_OCSP_URL** defined responder will be used first and then the responders identified in the AIA extension. This is the default.

Specify **GSK_OCSP_URL_PRIORITY_OFF** to indicate that the responder identified in the AIA extension will be used first and then the **GSK_OCSP_URL** defined responder.

GSK_OCSP_URL_PRIORITY can be specified only for an SSL environment.

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 and TLS protocols provide 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 is not used. Enabling this protocol has 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 is not used. Enabling this protocol has 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 is not used. Enabling this protocol has 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_attribute_set_enum()

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_PROTOCOL_TLSV1_2

Specify GSK_PROTOCOL_TLSV1_2_ON to enable the TLS Version 1.2 protocol or GSK_PROTOCOL_TLSV1_2_OFF to disable the TLS Version 1.2 protocol.

GSK_PROTOCOL_TLSV1_2 can be specified for an SSL environment or an SSL connection.

GSK_RENEGOTIATION

Specify GSK_RENEGOTIATION_NONE to disable SSL V3 and TLS handshake renegotiation as a server and allow RFC 5746 renegotiation. This is the default.

Specify GSK_RENEGOTIATION_DISABLED to disable SSL V3 and TLS handshake renegotiation as a server and also disable RFC 5746 renegotiation.

Specify GSK_RENEGOTIATION_ALL to allow SSL V3 and TLS handshake renegotiation as a server while also allowing RFC 5746 renegotiation.

Specify GSK_RENEGOTIATION_ABBREVIATED to allow SSL V3 and TLS abbreviated handshake renegotiation as a server for resuming the current session only, while disabling SSL V3 and TLS full handshake renegotiation as a server. With this enumeration value set, the System SSL session ID cache is not checked when resuming the current session. RFC 5746 renegotiation is allowed.

GSK_RENEGOTIATION can be specified only for an SSL environment.

GSK_RENEGOTIATION_PEER_CERT_CHECK

Specify GSK_RENEGOTIATION_PEER_CERT_CHECK_OFF to not perform an identity check against the peer's certificate during renegotiation. This allows the peer certificate to change during renegotiation. This is the default.

Specify GSK_RENEGOTIATION_PEER_CERT_CHECK_ON to perform a comparison against the peer's certificate to ensure that certificate does not change during renegotiation.

GSK_RENEGOTIATION_PEER_CERT_CHECK can be specified only for an SSL environment.

GSK_REQ_CACHED_SESSION

Specify GSK_REQ_CACHED_SESSION_ON to require the cached session identified by GSK_PEER_ID to be used for an upcoming SSL V3, TLS 1.0, or higher secure connection. If either a cached or full handshake is allowed, specify GSK_REQ_CACHED_SESSION_OFF.

GSK_REQ_CACHED_SESSION_OFF is the default setting.

GSK_REQ_CACHED_SESSION can be specified only for an SSL environment and is only applicable for client connections.

GSK_REVOCATION_SECURITY_LEVEL

Specifies the level of security to be used when contacting an OCSP responder or an HTTP server specified in a URI value of the CDP extension.

Three levels of security are available:

GSK_REVOCATION_SECURITY_LEVEL_LOW

Certificate validation does not fail if the OCSP responder or HTTP server specified in the URI value of the CDP extension cannot be contacted.

GSK_REVOCATION_SECURITY_LEVEL_MEDIUM

Certificate validation requires the OCSP responder or the HTTP server in a URI value in the CDP extension to be contactable. For an OCSP responder, it must be able to provide a valid certificate revocation status. If the certificate status is revoked or unknown, certificate validation fails. For an HTTP server in a CDP extension, it must be contactable and able to provide a CRL. This is the default setting.

GSK_REVOCATION_SECURITY_LEVEL_HIGH

Certificate validation requires revocation information to be provided by the OCSP responder or HTTP server. If OCSP revocation checking via the AIA extension is enabled, there must be HTTP URI values present in the certificate that are able to be contactable and able to provide a valid certificate revocation status. If HTTP CRL checking is enabled, there must be HTTP URI values in the CDP extension that are able to be contactable and able to provide a CRL.

`GSK_REVOCATION_SECURITY_LEVEL` can be specified only for an SSL environment.

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_SUITE_B_PROFILE

Specifies the Suite B profile that an SSL server or client applies to TLS sessions. RFC 5430 defines the cipher suites that are valid for use when using the compliant Suite B profile for TLS. Specify:

- `GSK_SUITE_B_PROFILE_128` if only the 128-bit Suite B security profile is required.
- `GSK_SUITE_B_PROFILE_192` if only the 192-bit Suite B security profile is required.
- `GSK_SUITE_B_PROFILE_ALL` if both the 128-bit and 192-bit Suite B security profiles are required.
- `GSK_SUITE_B_PROFILE_OFF` if the Suite B security profile is not to be applied to any TLS sessions.

`GSK_SUITE_B_PROFILE` can be specified only for an SSL environment.

Because this setting affects the cipher suites that are allowed, this also has an implicit effect on the Elliptic Curves and Certificates that can be used.

gsk_attribute_set_enum()

Suite B Cryptography requires that key establishment and authentication algorithms that are used in TLS sessions be based on Elliptic Curve Cryptography, and that the encryption algorithm be AES.

For more information about the cipher suites, elliptic curves, and certificates that are allowed by Suite B, see “Suite B cryptography support” on page 45.

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: 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.

GSK_TLS_CBC_PROTECTION_METHOD

|
| Specifies an optional SSL V3.0 or TLS V1.0 CBC IV protection method
| when writing application data.

|
| Specify `GSK_TLS_CBC_PROTECTION_METHOD_NONE` to indicate that
| no CBC protection is enabled. This is the default.

|
| Specify `GSK_TLS_CBC_PROTECTION_METHOD_ZEROBYTEFRAGMENT`
| to indicate that zero byte record fragmenting is enabled. When specified, a
| zero byte record fragment is sent before the application data records are
| sent.

|
| Specify `GSK_TLS_CBC_PROTECTION_METHOD_ONEBYTEFRAGMENT`
| to indicate that one byte record fragmenting is enabled. When specified,
| the first record is sent in two record fragments with the first record
| fragment containing only one byte of application data. The rest of the
| application data from the first record is sent in the second record fragment.
| All the following records are written whole. For example, a write of 256
| bytes of data that is broken into 64 byte record fragments would be written
| as:

|
| 1 byte, 63 bytes, 64 bytes, 64 bytes, 64 bytes

|
| `GSK_TLS_CBC_PROTECTION_METHOD` can only be specified for an SSL
| environment.

GSK_V3_CIPHERS

Specify `GSK_V3_CIPHERS_CHAR2` if the cipher specification is specified using 1 or more 2-character values in `GSK_V3_CIPHER_SPECS`. Specify `GSK_V3_CIPHERS_CHAR4` if the cipher specification is specified using 1 or more 4-character values in `GSK_V3_CIPHER_SPECS_EXPANDED`. `GSK_V3_CIPHERS` can be specified for an SSL environment or an SSL connection.

Related Topics

["gsk_attribute_get_enum\(\)" on page 72](#)

["gsk_environment_open\(\)" on page 114](#)

["gsk_environment_init\(\)" on page 112](#)

["gsk_secure_socket_open\(\)" on page 144](#)

["gsk_secure_socket_init\(\)" on page 133](#)

`gsk_attribute_set_numeric_value()`

`gsk_attribute_set_numeric_value()`

Sets a numeric value.

Format

```
#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 sets 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_ENTRY_MAXSIZE

Sets the maximum size in bytes of a CRL that is allowed to be stored in the LDAP CRL cache. Any CRLs larger than this size are not cached. The valid cache entry sizes are 0 through 2147483647. The default is 0, which means there is no limit on the size of the CRL stored in the LDAP CRL cache.

`GSK_CRL_CACHE_ENTRY_MAXSIZE` can be specified only for an SSL environment.

GSK_CRL_CACHE_SIZE

Sets the maximum number of CRLs that are allowed to be stored in the LDAP CRL cache. The valid cache sizes are -1 through 32000. If LDAP extended CRL cache support is enabled, the default is 32 and CRLs are only cached if they contain an expiration time that is later than the current time. If LDAP basic CRL cache support is enabled, the default is -1 (which is unlimited) and caching only occurs if GSK_CRL_CACHE_TIMEOUT is set to a value greater than 0. A value of 0 for GSK_CRL_CACHE_SIZE means that LDAP CRL caching is not enabled.

GSK_CRL_CACHE_SIZE can be specified only for an SSL environment.

GSK_CRL_CACHE_TEMP_CRL_TIMEOUT

Sets the time in hours that a temporary CRL cache entry resides in the LDAP CRL cache. A temporary LDAP CRL cache entry is added to the LDAP CRL cache when the CRL does not reside on the LDAP server. The range is 1-720 hours and defaults to 24 hours.

Note: This support is only available when LDAP extended CRL cache support is activated and caching of temporary CRLs is enabled.

GSK_CRL_CACHE_TEMP_CRL_TIMEOUT can be specified only for an SSL environment.

GSK_CRL_CACHE_TIMEOUT

Sets the LDAP basic CRL cache timeout. This is the number of hours that a cached CRL remains valid in the LDAP basic CRL cache. The range is 0-720 and defaults to 24. A value of 0 disables LDAP CRL caching.

GSK_CRL_CACHE_TIMEOUT can be specified only for an SSL environment.

GSK_FD

Sets the socket descriptor for network operations. GSK_FD can be specified only for an SSL connection. The socket must not be closed until the **gsk_secure_socket_close()** routine has been called to terminate the secure connection.

GSK_HTTP_CDP_CACHE_ENTRY_MAXSIZE

Sets the maximum size in bytes of a CRL that is allowed to be stored in the HTTP CDP CRL cache. Any CRLs larger than this size are not cached. The valid sizes are 0 through 2147483647. The default is 0, which means there is no limit on the size of the CRL stored in the HTTP CDP CRL cache.

GSK_HTTP_CDP_CACHE_ENTRY_MAXSIZE can be specified only for an SSL environment.

GSK_HTTP_CDP_CACHE_SIZE

Sets the maximum number of CRLs that are allowed to be stored in the HTTP CDP CRL cache. The valid sizes are 0 through 32000. The default is 32 and a value of 0 means that HTTP CDP CRL caching is disabled.

GSK_HTTP_CDP_CACHE_SIZE can be specified only for an SSL environment.

GSK_HTTP_CDP_MAX_RESPONSE_SIZE

Sets the maximum size in bytes accepted as a response from an HTTP server when retrieving a CRL. The valid sizes are 0 through 2147483647. A value of 0 will disable checking the size and allow a CRL of any size. Setting the maximum response size too small could implicitly disable HTTP CRL support. The default is 204800 (200K).

gsk_attribute_set_numeric_value()

GSK_HTTP_CDP_MAX_RESPONSE_SIZE can be specified only for an SSL environment.

GSK_HTTP_CDP_PROXY_SERVER_PORT

Sets the HTTP proxy server port for HTTP CDP CRL retrieval. The port must be between 1 and 65535. Port 80 is used if no HTTP proxy server port is set.

GSK_HTTP_CDP_PROXY_SERVER_PORT can be specified only for an SSL environment.

GSK_HTTP_CDP_RESPONSE_TIMEOUT

Sets the time in seconds to wait for a complete response from the HTTP server. The valid time limits are 0 through 43200 seconds (12 hours). The default is 15 seconds and a value of 0 means there is no time limit for HTTP CRL retrievals.

GSK_HTTP_CDP_RESPONSE_TIMEOUT can be specified only for an SSL environment.

GSK_LDAP_RESPONSE_TIMEOUT

Sets the time in seconds to wait for a response from the LDAP server. The valid time limits are 0 through 43200 seconds (12 hours). The default is 15 seconds and a value of 0 means that there is no time limit for LDAP CRL retrievals.

GSK_LDAP_RESPONSE_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 LDAP server port is set.

GSK_LDAP_SERVER_PORT can be specified only for an SSL environment.

GSK_MAX_SOURCE_REV_EXT_LOC_VALUES

Sets the maximum number of locations values that will be contacted per HTTP CDP or AIA extension when attempting validation of a certificate. The locations for revocation information are specified by the accessLocation in the AIA certificate extension for OCSP and the distributionPoint in the CDP extension for HTTP CRLs. When an HTTP URI is present in an AIA or CDP extension, validation will attempt to contact the remote HTTP server to obtain revocation information. Both of these extensions can contain multiple location values and therefore have the potential to impact performance when there are a very large number of locations present. The valid values are 0 through 256. The default value is 10 and a value of 0 indicates there is no limit on the number of locations contacted.

GSK_MAX_SOURCE_REV_EXT_LOC_VALUES can be specified only for an SSL environment.

GSK_MAX_VALIDATION_REV_EXT_LOC_VALUES

Sets the maximum number of location values that will be contacted when performing validation of a certificate. The locations for revocation information are specified by the accessLocation in the AIA certificate extension for OCSP and the distributionPoint in the CDP extension for HTTP CRLs. When an HTTP URI is present in an AIA or CDP extension, validation will attempt to contact the remote HTTP server to obtain revocation information. Both of these extensions can contain multiple location values and therefore has the potential to negatively impact performance when there are a very large number of locations present. The

valid values are 0 through 1024. The default value for this option is 100 and a value of 0 indicates there is no limit on the number of locations contacted.

GSK_MAX_VALIDATION_REV_EXT_LOC_VALUES can be specified only for an SSL environment.

GSK_OCSP_CLIENT_CACHE_ENTRY_MAXSIZE

Sets the maximum number of OCSP responses or cached certificate statuses that are allowed to be kept in the OCSP response cache for an issuing CA certificate. The valid sizes are 0 through 32000 and must be less than or equal to the size specified for GSK_OCSP_CLIENT_CACHE_SIZE. By default, the size is set to 0 which means there is no limit on the number of cached certificate statuses allowed for a specific issuing CA certificate other than the limit imposed by GSK_OCSP_CLIENT_CACHE_SIZE.

Note: GSK_OCSP_CLIENT_CACHE_SIZE specifies the total number of cached certificate statuses allowed in the entire OCSP cache. If this count is exceeded, any expired certificate statuses are first removed. If there are no expired certificate statuses that have the same issuing CA certificate, the certificate status that is closest to the expiration time is removed first. This cache size is rounded up to the nearest multiple of 16 with a minimum size of 16.

GSK_OCSP_CLIENT_CACHE_ENTRY_MAXSIZE can be specified only for an SSL environment.

GSK_OCSP_CLIENT_CACHE_SIZE

Sets the maximum number of OCSP responses or cached certificate statuses to be kept in the OCSP response cache. The valid cache sizes are 0 through 32000 and defaults to 256. The OCSP response cache will be disabled if 0 is specified. The OCSP response cache will be allocated using the requested size rounded up to the nearest multiple of 16 with a minimum size of 16.

GSK_OCSP_CLIENT_CACHE_SIZE can be specified only for an SSL environment.

GSK_OCSP_MAX_RESPONSE_SIZE

Sets the maximum size in bytes allowed in a response from an OCSP responder. The valid response sizes are 0 through 2147483647 and defaults to 20480. A value of 0 will disable checking the size and allows an OCSP response of any size. Setting the maximum response size too small could implicitly disable OCSP support.

GSK_OCSP_MAX_RESPONSE_SIZE can be specified only for an SSL environment.

GSK_OCSP_NONCE_SIZE

Sets the size in bytes for the value of the nonce to be sent in OCSP requests.

The valid nonce sizes are 8 through 256 and defaults to 8.

GSK_OCSP_NONCE_SIZE can be specified only for an SSL environment.

GSK_OCSP_PROXY_SERVER_PORT

Sets the OCSP responder port for the proxy. The port must be between 1 and 65535. Port 80 is used if no OCSP proxy server port is set.

GSK_OCSP_PROXY_SERVER_PORT can be specified only for an SSL environment.

gsk_attribute_set_numeric_value()

GSK_OCSP_RESPONSE_TIMEOUT

Sets the time in seconds to wait for a complete response from the OCSP responder. The valid time limits are 0 through 43200 seconds (12 hours) and defaults to 15 seconds. A value of 0 indicates there is no time limit for the retrieval of the OCSP response.

GSK_OCSP_RESPONSE_TIMEOUT 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 remembers 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 is performed. Session identifiers are not 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 is removed when the cache is full to add a new entry. The range is 0-32000 and defaults to 256. Session identifiers are not remembered if a value of 0 is specified. The session identifier cache is 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, or higher 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 remembers 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 are not 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 to add a new entry. The range is 0-64000 and defaults to 512. Session identifiers are not remembered if a value of 0 is specified. The SSL V3 session cache is used for the SSL V3, TLS V1.0, or higher protocols. The session identifier cache is allocated by 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\(\)" on page 79](#)

["gsk_environment_open\(\)" on page 114](#)

["gsk_environment_init\(\)" on page 112](#)

"gsk_secure_socket_init()" on page 133

"gsk_secure_socket_open()" on page 144

`gsk_attribute_set_tls_extension()`

`gsk_attribute_set_tls_extension()`

Defines a TLS extension to the SSL environment or connection.

Format

```
#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 defines 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()` is not called). TLS extensions that are defined for an SSL environment applies 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 or higher connections only.

The application must prime the TLS extension structure with the appropriate TLS extension data before calling the routine, including the TLS extension type identifier and the specific data that is 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 enforces support requirements of the specific extension type on the communicating partner. If the partner indicates that it does not support the extension, the connection is rejected. An optional setting allows the connection to continue without support for that particular extension type if the communicating partner indicates that it does not support the TLS extension type.

Note:

1. Setting an extension as required for a server means that all clients connecting to the server must have the extension enabled. Failure for a client to do so results in the server rejecting the connection request from the client. It is recommended that for maximum interoperability, that the required field is not 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 zeros; any non-zero data results 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 prevents the SSL V2 protocol from being used.

The values set by using this service are treated as independent values. They are not validated with other values set using `gsk_attribute_set_buffer()`, `gsk_attribute_set_enum()`, or `gsk_attribute_set_tls_extensions()` APIs until used together to perform a SSL/TLS handshake by calling `gsk_secure_socket_init()`.

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 with the server name details received from the client to determine which certificate from the key database, PKCS #12 file, key ring or token is sent to the client as the servers 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 deactivates 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 clients server name list, the server sends 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, are effective when running as a server; it is ignored for clients.

Note:

1. It is recommended that the `gsk_sni_server_labels` structure to be included in the `gsk_tls_extension` data be initialized with binary zeros before setting the required server label data. This ensures future application compatibility when additional bits within the `gsk_sni_server_labels` structure are used.
2. System SSL only supports server names that contain US-ASCII characters.

GSK_TLS_EXTID_SNI_CLIENT_SNAMES

Specifies the server name (or list of server names) that the client sends to the server in a 'Server Name Indication' type TLS extension to indicate

gsk_attribute_set_tls_extension()

with which server the client wants to communicate. The list of server names 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 deactivates 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 clients server name list, the server sends 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, are effective when running as a client; it is ignored for servers.

Note:

1. It is recommended that the *gsk_sni_client_snames* structure to be included in the *gsk_tls_extension* data be initialized with binary zeros before setting the required server label data. This will ensure future application compatibility when additional bits within the *gsk_sni_client_snames* structure are used.
2. System SSL only supports server names that contain US-ASCII characters.

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 deactivates 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 deactivates a previously registered GSK_TLS_EXTID_CLIENT_MFL type TLS extension setting.

GSK_TLS_EXTID_TRUNCATED_HMAC

Specifies whether the TLS server or client supports the 'Truncated HMAC' type TLS extension. Set *truncateHmac* to TRUE to enable the extension. A *truncateHmac* setting of FALSE deactivates a previously registered GSK_TLS_EXTID_TRUNCATED_HMAC type TLS extension setting.

gsk_environment_close()

Closes an SSL environment.

Format

```
#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 closes an environment created by the **gsk_environment_open()** routine. The storage that is allocated for the environment is not released until all connections created using the environment are closed. The SSL environment cannot be used to create new connections upon completion of the close.

Related Topics

[“gsk_environment_open\(\)” on page 114](#)

[“gsk_environment_init\(\)” on page 112](#)

[“gsk_secure_socket_init\(\)” on page 133](#)

[“gsk_secure_socket_close\(\)” on page 132](#)

`gsk_environment_init()`

Initializes an SSL environment.

Format

```
#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_NOT_AVAILABLE]

The key database, PKCS #12 file, key ring or token does not contain any certificates.

[GSK_ERR_BAD_KEYFILE_PASSWORD]

The key database or PKCS #12 file password is not correct.

[GSK_ERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[GSK_ERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[GSK_ERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[GSK_ERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[GSK_ERR_LDAP]

Unable to initialize the LDAP client.

[GSK_ERR_LDAP_NOT_AVAILABLE]

The LDAP server is not available.

[GSK_ERR_PERMISSION_DENIED]

Not authorized to access key database, PKCS #12 file, SAF key ring or z/OS PKCS #11 token.

[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 open state.

[GSK_KEYFILE_INVALID_FORMAT]

The database is not a key database.

[GSK_KEYFILE_IO_ERR]

An input/output error occurred while reading the key database, PKCS #12 file, key ring or token.

[GSK_KEYFILE_PASSWORD_EXPIRED]

The key database password is expired.

[GSK_KEYRING_OPEN_ERROR]

Unable to open the key database, PKCS #12 file, key ring or token.

[GSK_NO_KEYFILE_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\(\)” on page 114](#)

[“gsk_environment_close\(\)” on page 111](#)

[“gsk_secure_socket_open\(\)” on page 144](#)

`gsk_environment_open()`

Creates an 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:

- TLS V1.0 is enabled (SSL V2, SSL V3, TLS V1.1 and TLS V1.2 are 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 performed.
- The default callback routines will be used.

- The SSL V2 cipher specification is set to "713642" if United States only encryption is enabled (System SSL Security Level 3 FMID installed) and "642" otherwise.
- 2-character cipher definitions in GSK_V3_CIPHER_SPECS will be used for SSL V3 cipher values.
- The SSL V3 cipher specification is set to "35363738392F303132330A1613100D0915120F0C" if United States only encryption is enabled (System SSL Security Level 3 FMID installed) and "0915120F0C" otherwise.
- The supported elliptic curve list is set to "00210023002400250019".
- The signature algorithm pair list is set to "0601060305010503040104030402030103030201020302020101".
- No TLS extensions are initialized.
- Suite B is disabled.
- OCSP support is disabled (OCSP URL is not defined and AIA extensions are not enabled).
- HTTP CDP CRL support is disabled.
- LDAP CRL support is disabled.

When executing in FIPS mode, the following default values are set:

- TLS V1.0 is enabled (SSL V2, SSL V3, TLS V1.1 and TLS V1.2 are 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.
- 2-character cipher definitions in GSK_V3_CIPHER_SPECS will be used for SSL V3 cipher values.
- The cipher specification is set to "35363738392F303132330A1613100D".
- The supported elliptic curve list is set to "00210023002400250019".
- The signature algorithm pair list is set to "060106030501050304010403040203010303020102030202".
- Suite B is disabled.
- OCSP support is disabled (OCSP URL is not defined and AIA extensions are not enabled).
- HTTP CDP CRL support is disabled.
- LDAP CRL support is disabled.

See Table 18 on page 687 for a list of supported SSL V2 cipher specifications.

See Table 19 on page 687 for a list of supported 2-character SSL V3 cipher specifications.

See Table 20 on page 691 for a list of supported 4-character SSL V3 cipher specifications.

gsk_environment_open()

See Table 22 on page 695 for a list of supported 4-character elliptic curve specifications.

Applications wanting to use cipher suites that use elliptic curve certificates must set an appropriate cipher specification in `GSK_V3_CIPHER_SPECS_EXPANDED`. If an application requires an SSL V3, TLS V1.0, or higher session to use the 4-character cipher suites specified in `GSK_V3_CIPHER_SPECS_EXPANDED` then it must explicitly call `gsk_attribute_set_enum()` and set the enumeration identifier `GSK_V3_CIPHERS` to have a value of `GSK_V3_CIPHERS_CHAR4`.

If an application has indicated it is using the 4-character cipher specifications by setting `GSK_V3_CIPHERS` to `GSK_V3_CIPHERS_CHAR4`, but does not set a cipher specification in `GSK_V3_CIPHER_SPECS_EXPANDED`, the default cipher specification will be set as follows:

- Executing in non-FIPS mode with United States only encryption enabled (System SSL Security Level 3 FMID installed):
"00350036003700380039002F0030003100320033000A001600130010000D000900150012000F000C"
- Executing in non-FIPS mode with United States only encryption disabled (System SSL Security Level 3 FMID is not installed):
"000900150012000F000C"
- Executing in FIPS mode:
"00350036003700380039002F0030003100320033000A001600130010000D"

If an application has indicated it will be running in Suite B compatibility mode by setting `GSK_SUITE_B_PROFILE` to a value other than `GSK_SUITE_B_PROFILE_OFF`, the cipher specification will be set based on the values for `GSK_SUITE_B_PROFILE` as follows:

- Executing with `GSK_SUITE_B_PROFILE_128` "C02BC023"
- Executing with `GSK_SUITE_B_PROFILE_192` "C02CC024"
- Executing with `GSK_SUITE_B_PROFILE_ALL` "C02CC024C02BC023"

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

- When using 2-character cipher suites:

0A 0D 10 13 16 2F 30 31 32 33 35 36 37 38 39 3C 3D 3E 3F 40 67 68 69
6A 6B 9C 9D 9E 9F A0 A1 A2 A3 A4 A5

- When using 4-character cipher suites:

000A 000D 0010 0013 0016 002F 0030 0031 0032 0033 0035 0036 0037 0038
0039 003C 003D 003E 003F 0040 0067 0068 0069 006A 006B 009C 009D 009E
009F 00A0 00A1 00A2 00A3 00A4 00A5 C003 C004 C005 C008 C009 C00A
C00D C00E C00F C012 C013 C014 C023 C024 C025 C026 C027 C028 C029
C02A C02B C02C C02D C02E C02F C030 C031 C032

If using the TLS V1.1 or higher protocols, export ciphers are not supported. The 40-bit ciphers (cipher specifications "03" and "06" or "0003" and "0006") will be ignored if specified.

If using the TLS V1.2 or higher protocols the 56-bit DES cipher suites "09", "0C", "0F", "12" and "15" (or "0009", "000C", "000F", "0012" and "0015") will be ignored if specified.

These environment variables are processed (See Appendix A, “Environment variables,” on page 667 for information about environment variables):

GSK_AIA_CDP_PRIORITY

Specifies the priority order that the AIA and the CDP extensions are checked for certificate revocation information.

GSK_CERT_VALIDATION_KEYRING_ROOT

Specifies how certificates in a SAF key ring are validated.

GSK_CERT_VALIDATION_MODE

Specifies which internet standard is used for certificate validation.

GSK_CLIENT_AUTH_NOCERT_ALERT

Specifies whether the SSL server application accepts a connection from a client where client authentication is requested and the client fails to supply an X.509 certificate.

GSK_CLIENT_ECURVE_LIST

Specifies the list of elliptic curves that are supported by the client.

GSK_CRL_CACHE_ENTRY_MAXSIZE

Specifies the maximum size in bytes of a CRL to be kept in the LDAP CRL cache.

GSK_CRL_CACHE_EXTENDED

Specifies that LDAP extended CRL cache support is enabled.

GSK_CRL_CACHE_SIZE

Specifies the maximum number of CRLs that are allowed to be stored in the LDAP CRL cache.

GSK_CRL_CACHE_TIMEOUT

Specifies the number of hours that a cached LDAP CRL remains valid.

GSK_CRL_CACHE_TEMP_CRL

Specifies if a temporary LDAP CRL cache entry is added to the LDAP CRL cache when the CRL does not reside on the LDAP server.

GSK_CRL_CACHE_TEMP_CRL_TIMEOUT

Specifies the time in hours that a temporary CRL cache entry resides in the LDAP CRL cache.

GSK_CRL_SECURITY_LEVEL

Specifies the level of security used when contacting LDAP servers to check CRLs for revoked certificates.

GSK_EXTENDED_RENEGOTIATION_INDICATOR

Specifies the level of enforcement of renegotiation indication.

GSK_HTTP_CDP_CACHE_ENTRY_MAXSIZE

Specifies the maximum size in bytes of a CRL that can be stored in the HTTP CDP CRL cache.

GSK_HTTP_CDP_CACHE_SIZE

Specifies the maximum number of CRLs that are allowed to be stored in the HTTP CDP CRL cache.

GSK_HTTP_CDP_ENABLE

Specifies if certificate revocation checking with the HTTP URI values in the CDP extension is enabled.

gsk_environment_open()

| **GSK_HTTP_CDP_MAX_RESPONSE_SIZE**
| Specifies the maximum size in bytes accepted as a response from an HTTP
| server when retrieving a CRL.

| **GSK_HTTP_CDP_PROXY_SERVER_NAME**
| Specifies the DNS name or IP address of the HTTP proxy server.

| **GSK_HTTP_CDP_PROXY_SERVER_PORT**
| Specifies the HTTP proxy server port.

| **GSK_HTTP_CDP_RESPONSE_TIMEOUT**
| Specifies the time in seconds to wait for a response from the HTTP server.

| **GSK_KEY_LABEL**
| Specifies the label of the key that used to authenticate the application.

| **GSK_KEYRING_FILE**
| Specifies the name of the key database file, PKCS #12 file, SAF key ring, or
| z/OS PKCS #11 token.

| **GSK_KEYRING_PW**
| Specifies the password for the key database or PKCS #12 file.

| **GSK_KEYRING_STASH**
| Specifies the name of the key database password stash file.

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

| **GSK_LDAP_PORT**
| Specifies the LDAP server port.

| **GSK_LDAP_RESPONSE_TIMEOUT**
| Specifies the time in seconds to wait for a response from the LDAP server.

| **GSK_LDAP_SERVER**
| Specifies one or more blank-separated LDAP server host names.

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

| **GSK_MAX_SOURCE_REV_EXT_LOC_VALUES**
| Specifies the maximum number of location values that will be contacted
| per data source when attempting validation of a certificate.

| **GSK_MAX_VALIDATION_REV_EXT_LOC_VALUES**
| Specifies the maximum number of location values that will be contacted
| when performing validation of a certificate.

| **GSK_OCSP_CLIENT_CACHE_ENTRY_MAXSIZE**
| Specifies the maximum number of OCSP responses or cached certificate
| statuses that are allowed to be kept in the OCSP response cache for an
| issuing CA certificate.

| **GSK_OCSP_CLIENT_CACHE_SIZE**
| Specifies the maximum number of OCSP responses or cached certificate
| statuses to be kept in the OCSP response cache.

| **GSK_OCSP_ENABLE**
| Specifies whether the AIA extensions are to be used for revocation
| checking.

GSK_OCSP_MAX_RESPONSE_SIZE

Specifies the maximum size in bytes allowed in a response from an OCSP responder.

GSK_OCSP_NONCE_CHECK_ENABLE

Specifies if OCSP response nonce checking is on or off.

GSK_OCSP_NONCE_GENERATION_ENABLE

Specifies whether an OCSP request will contain a generated nonce.

GSK_OCSP_NONCE_SIZE

Specifies the size in bytes for the value of the nonce to be sent in OCSP requests.

GSK_OCSP_PROXY_SERVER_NAME

Specifies the DNS name or IP address of the OCSP proxy server.

GSK_OCSP_PROXY_SERVER_PORT

Specifies the OCSP responder port for the proxy.

GSK_OCSP_REQUEST_SIGALG

Specifies the hash and signature algorithm pair to be used to sign OCSP requests.

GSK_OCSP_REQUEST_SIGKEYLABEL

Specifies the label of the key to be used to sign OCSP requests.

GSK_OCSP_RESPONSE_TIMEOUT

Specifies the time in seconds to wait for a complete response from the OCSP responder.

GSK_OCSP_RETRIEVE_VIA_GET

Specifies whether the HTTP request to the OCSP responder is sent using the HTTP Get Method or the HTTP Post method.

GSK_OCSP_URL

Specifies the URI of an OCSP responder.

GSK_OCSP_URL_PRIORITY

Specifies the order of precedence for contacting OCSP responder locations if both GSK_OCSP_URL and GSK_OCSP_ENABLE are active.

GSK_PROTOCOL_SSLV2

Specifies whether the SSL V2 protocol is supported.

GSK_PROTOCOL_SSLV3

Specifies whether the SSL V3 protocol is supported.

GSK_PROTOCOL_TLSV1

Specifies whether the TLS V1.0 protocol is supported.

GSK_PROTOCOL_TLSV1_1

Specifies whether the TLS V1.1 protocol is supported.

GSK_PROTOCOL_TLSV1_2

Specifies whether the TLS V1.2 protocol is supported.

GSK_REVOCATION_SECURITY_LEVEL

Specifies the level of security to be used when contacting an OCSP responder or an HTTP server specified in a URI value of the CDP extension.

GSK_RENEGOTIATION

Specifies the type of session renegotiation allowed for an SSL environment.

gsk_environment_open()

	GSK_RENEGOTIATION_PEER_CERT_CHECK
	Specifies if the peer certificate is allowed to change during renegotiation.
	GSK_SUITE_B_PROFILE
	Specifies the Suite B profile to be applied to TLS V1.2 sessions.
	GSK_SYSPLEX_SIDCACHE
	Specifies whether sysplex session caching is supported.
	GSK_TLS_CBC_PROTECTION_METHOD
	Specifies an optional SSL V3.0 or TLS V1.0 CBC IV protection method when writing application data.
	GSK_TLS_SIG_ALG_PAIRS
	Specifies the list of TLS V1.2 hash and signature algorithm pair specifications that are supported by the client or server.
	GSK_V2_CIPHER_SPECS
	Specifies the SSL V2 cipher specifications in order of preference.
	GSK_V2_SESSION_TIMEOUT
	Specifies the session timeout value in seconds for the SSL V2 protocol.
	GSK_V2_SIDCACHE_SIZE
	Specifies the number of session identifiers that can be contained in the SSL V2 cache.
	GSK_V3_CIPHER_SPECS
	Specifies the SSL V3 cipher specifications in order of preference (2-character values).
	GSK_V3_CIPHER_SPECS_EXPANDED
	Specifies the SSL V3 cipher specifications in order of preference (4-character values).
	GSK_V3_SESSION_TIMEOUT
	Specifies the session timeout value in seconds for the SSL V3, TLS V1.0, and higher protocols.
	GSK_V3_SIDCACHE_SIZE
	Specifies the number of session identifiers that can be contained in the SSL V3 cache.

Related Topics

[“gsk_environment_init\(\)” on page 112](#)

[“gsk_environment_close\(\)” on page 111](#)

gsk_free_cert_data()

Releases the storage allocated for a certificate data array.

Format

```
#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 releases the storage allocated for an array of certificate data elements.

Related Topics

“`gsk_attribute_get_cert_info()`” on page 65

“`gsk_get_cert_by_label()`” on page 123

`gsk_get_all_cipher_suites()`

Returns the available SSL cipher suites.

Format

```
#include <gskssl.h>

gsk_status gsk_get_all_cipher_suites (
    gsk_all_cipher_suites * cipher_suites)
```

Parameters

`cipher_suites`

Returns the runtime version, release, security level, and cipher suites.

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_STRUCTURE_TOO_SMALL]

Size specified for supplied structure is too small.

Usage

The `gsk_get_all_cipher_suites()` routine returns the System SSL runtime version, release, security level, and available cipher suites. The current System SSL run time is Version 4 Release 2. The cipher suites are static null-terminated character strings which must not be modified or freed by the application. The available cipher suites for protocols SSL V3.0, TLS V1.0, and higher are returned in both 2-character and 4-character formats.

The cipher lists include all supported ciphers. As new ciphers are added, the lists will be modified to contain the newly added ciphers. The adding of ciphers may cause cipher selection to be modified as new ciphers are added, and different ciphers to be selected if the lists are being used as the cipher list strings.

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 114.

The application must initialize the size field in the `gsk_all_cipher_suites` structure to the size of the `gsk_all_ciphers_suites` structure before using this function.

gsk_get_cert_by_label()

Gets certificate information for a record label.

Format

```
#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 depends upon the contents of the certificate. These element identifiers are currently provided:

gsk_get_cert_by_label()

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 run time.

- C - Country
- CN - Common name
- DC - Domain component
- DNQUALIFIER - Distinguished name qualifier
- EMAIL - email address
- GENERATIONQUALIFIER - Generation qualifier
- GIVENNAME - Given name
- INITIALS - Initials
- L - Locality
- MAIL - RFC 822 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 email 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 run time.

- C - Country
- CN - Common name
- DC - Domain component
- DNQUALIFIER - Distinguished name qualifier
- EMAIL - email address
- GENERATIONQUALIFIER - Generation qualifier
- GIVENNAME - Given name
- INITIALS - Initials
- L - Locality
- MAIL - RFC 822 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 email 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 RFC 822 style address (MAIL)

gsk_get_cert_by_label()

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 RFC 822 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.

Related Topics

["gsk_environment_init\(\)" on page 112](#)

["gsk_secure_socket_init\(\)" on page 133](#)

gsk_get_cipher_suites()

gsk_get_cipher_suites()

Returns the available SSL cipher suites.

Format

```
#include <gskssl.h>
```

```
void gsk_get_cipher_suites ( gsk_cipher_suites * cipher_suites)
```

Parameters

cipher_suites

Returns the runtime version, release, security level, and cipher suites.

Usage

The **gsk_get_cipher_suites()** routine returns the System SSL runtime version, release, security level, and available cipher suites. The current System SSL run time is Version 4 Release 2. 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 114.

gsk_get_ssl_vector()

Obtain the address of the Secure Socket Layer function vector.

Format

```
#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 sidedeck file is specified as input to the binder. This causes all SSL functions to be resolved at bind time and causes 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 are defined:

GSKSSL_API_LVL1

SSL functions provided as part of z/OS Version 1 Release 6 are available.

GSKSSL_API_LVL2

SSL functions provided as part of z/OS Version 1 Release 11 are available.

GSKSSL_API_LVL3

SSL functions provided as part of z/OS Version 1 Release 13 are available.

`gsk_get_update()`

`gsk_get_update()`

Checks for a key database, PKCS #12 file, SAF key ring or z/OS PKCS #11 token update.

Format

```
#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, PKCS #12 file, 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_ERR_PERMISSION_DENIED]

Not authorized to access key database, PKCS #12 file, SAF key ring or z/OS PKCS #11 token.

[GSK_INVALID_HANDLE]

The environment handle is not valid.

[GSK_INVALID_STATE]

The environment is not in the initialized state.

[GSK_KEYRING_OPEN_ERROR]

Unable to open the key database, PKCS #12 file, key ring or token.

Usage

The `gsk_get_update()` routine tests if the key database, PKCS #12 file, 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\(\)” on page 114](#)

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\(\)” on page 70](#)

`gsk_secure_socket_close()`

Closes a secure socket connection.

Format

```
#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 closes 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\(\)” on page 144](#)

[“gsk_secure_socket_init\(\)” on page 133](#)

[“gsk_secure_socket_read\(\)” on page 145](#)

[“gsk_secure_socket_write\(\)” on page 150](#)

gsk_secure_socket_init()

Initializes a secure socket connection.

Format

```
#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_CERTIFICATE_NOT_AVAILABLE]

No certificates available.

[GSK_ERR_BAD_CERT]

Certificate is not valid.

[GSK_ERR_BAD_DATE]

Certificate is not valid yet or is expired.

[GSK_ERR_BAD_EC_PARAMS]

EC parameters not supplied.

[GSK_ERR_BAD_HTTP_RESPONSE]

HTTP response is not valid.

[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_OCSP_RESPONSE]

OCSP response is not valid.

[GSK_ERR_BAD_PEER]

Peer application has violated the SSL protocol.

[GSK_ERR_BAD_PEER_ID]

Specified peer identifier is not valid.

[GSK_ERR_BAD_SID_VALUE]

Specified session identifier is not valid.

gsk_secure_socket_init()

- [GSK_ERR_BAD_SIG_ALG_PAIR]
Signature algorithm pairs list is not valid.
- [GSK_ERR_BAD_V2_CIPHER]
SSL V2 cipher is not valid.
- [GSK_ERR_BAD_V3_CIPHER]
SSL V3 cipher is not valid.
- [GSK_ERR_BAD_V3_EXPANDED_CIPHER]
SSL V3 expanded cipher is not valid.
- [GSK_ERR_CERT_VALIDATION]
Certificate validation error.
- [GSK_ERR_CERTIFICATE_REVOKED]
Peer certificate is revoked.
- [GSK_ERR_CLIENT_AND_SERVER_SID_NOT_EQUAL]
Client session identifier does not match the server session identifier.
- [GSK_ERR_CRYPTO]
Cryptographic error detected.
- [GSK_ERR_EC_PARAMETERS_NOT_SUPPLIED]
EC parameters not supplied.
- [GSK_ERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.
- [GSK_ERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.
- [GSK_ERR_HOSTNAME_NOT_VALID]
HTTP server hostname is not valid.
- [GSK_ERR_HTTP_IO_ERROR]
HTTP server communication error.
- [GSK_ERR_HTTP_RESPONSE_TIMEOUT]
HTTP response not received within configured time limit.
- [GSK_ERR_INCOMPATIBLE_KEY]
Certificate key is not compatible with cipher suite.
- [GSK_ERR_ICSF_CLEAR_KEY_SUPPORT_NOT_AVAILABLE]
Clear key support not available due to ICSF key policy.
- [GSK_ERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.
- [GSK_ERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.
- [GSK_ERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.
- [GSK_ERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.
- [GSK_ERR_INCORRECT_KEY_ATTRIBUTE]
TKDS Private Key attributes do not support digital signature or RSA operation.
- [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.

[GSK_ERR_LDAP_NOT_AVAILABLE]
The LDAP server is not available.

| [GSK_ERR_LDAP_RESPONSE_TIMEOUT]
| LDAP response not received within configured time limit.

| [GSK_ERR_MAX_RESPONSE_SIZE_EXCEEDED]
| Maximum response size exceeded.

[GSK_ERR_MISSING_KEY_ALGORITHM]
Certificate key algorithm is not in signature algorithm pairs list.

[GSK_ERR_MISSING_SIGNATURE_ALGORITHM]
Signature algorithm is not in signature algorithm pairs list.

[GSK_ERR_MULTIPLE_DEFAULT]
Multiple keys are marked as the default.

[GSK_ERR_MULTIPLE_LABEL]
Multiple certificates exist for label.

| [GSK_ERR_NO_CACHE_SESSION]
| No cached session entry exists.

[GSK_ERR_NO_CERTIFICATE]
No certificate received from partner.

[GSK_ERR_NO_CIPHERS]
No cipher specifications.

[GSK_ERR_NO_PRIVATE_KEY]
Certificate does not contain a private key or the private key is unusable.

[GSK_ERR_NON_SUITE_B_CERTIFICATE]
Certificate does not meet Suite B requirements.

| [GSK_ERR_OCSP_EXPIRED]
| OCSP response is expired.

| [GSK_ERR_OCSP_NONCE_CHECK_FAILED]
| Nonce in OCSP response does not match value in OCSP request.

| [GSK_ERR_OCSP_REQ_ERROR]
| Error creating OCSP request.

| [GSK_ERR_OCSP_RESPONDER_ERROR]
| OCSP request failed with internal responder error.

| [GSK_ERR_OCSP_RESPONSE_TIMEOUT]
| OCSP response not received within configured time limit.

| [GSK_ERR_OCSP_SIG_REQUIRED]
| OCSP responder requires a signed request.

| [GSK_ERR_PEER_ID_ATTRIBUTES_CONFLICT]
| Client session cache attributes do not agree.

| [GSK_ERR_REQ_CERT_BC_EXT_MISSING]
| The required basic constraints certificate extension is missing.

gsk_secure_socket_init()

	[GSK_ERR_REVINFO_NOT_YET_VALID]
	Revocation information is not yet valid.
	[GSK_ERR_RNG]
	Error encountered when generating random bytes.
	[GSK_ERR_SECURE_LABEL_OPERATION_UNSUPPORTED]
	A secure private key cannot be used with a fixed ECDH key exchange.
	[GSK_ERR_SELF_SIGNED]
	A self-signed certificate cannot be validated.
	[GSK_ERR_SIGNATURE_NOT_SUPPLIED]
	Signature not supplied.
	[GSK_ERR_SOCKET_CLOSED]
	Socket connection closed by peer application.
	[GSK_ERR_UNKNOWN_CA]
	A certification authority certificate is missing.
	[GSK_ERR_UNRECOGNIZED_NAME]
	The requested server name is not recognized.
	[GSK_ERR_UNSUPPORTED]
	SSL protocol or certificate type is not supported.
	[GSK_ERR_UNSUPPORTED_CERTIFICATE_TYPE]
	The certificate type is not supported by System SSL.
	[GSK_ERR_UNSUPPORTED_REQUIRED_EXTENSION]
	A required TLS extension has been rejected.
	[GSK_ERR_UNSUPPORTED_EXTENSION]
	An unrequested TLS Extension has been encountered.
	[GSK_INSUFFICIENT_STORAGE]
	Insufficient storage is available.
	[GSK_INVALID_HANDLE]
	The connection handle is not valid.
	[GSK_INVALID_STATE]
	The connection is not in the open state or a previous initialization request has failed.
	[GSK_RSA_TEMP_KEY_PAIR]
	Unable to generate temporary RSA public/private key pair.
	[GSK_WOULD_BLOCK_READ]
	An attempt to read a handshake message failed with EWOULDBLOCK.
	[GSK_WOULD_BLOCK_WRITE]
	An attempt to write a handshake message failed with EWOULDBLOCK.

Usage

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.

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.

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()** to have the SSL handshake processed in non-blocking mode (the default **io_setsocketoptions()** routine places the socket into blocking mode during the handshake processing).

In FIPS mode, only DSA certificates with domain parameters that conform to FIPS 186-3: *Digital Signature Standard (DSS)* are supported. In non-FIPS mode, if the key size is less than 1024 bits, then domain parameters that conform to FIPS 186-2 are supported. In non-FIPS mode, if the key size is greater than or equal to 1024 bits, the domain parameters must conform to FIPS 186-3, with the exception that parameters that have a prime modulus (p) of 2048 bits and a prime divisor (q) of 160 bits are also tolerated.

Be sure a **gsk_secure_socket_shutdown()** call is issued before a **gsk_secure_socket_close()** call.

Protocol Selection

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 attempts 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 V1.2 is used if it is enabled on both the client and the server
- If TLS V1.2 cannot be used and TLS V1.1 is enabled, negotiations drop back to TLS V1.1
- If TLS V1.1 cannot be used and TLS V1.0 is enabled, negotiations drop back to TLS V1.0
- If TLS V1.0 cannot be used and SSL V3 is enabled, negotiations drop back to SSL V3
- If SSL V3 cannot be used, TLS V1.2 was not enabled on the client or server, and SSL V2 is enabled, negotiations drop back to SSL V2

Note:

1. SSL V2 and SSL V3 are not as secure as TLS and should be disabled whenever possible to avoid attacks that force the client and server to drop back to SSL V2 or SSL V3 even though they are capable of using TLS V1.0, TLS V1.1, or TLS V1.2.

`gsk_secure_socket_init()`

2. When TLS extensions are defined for a client and any of the TLS protocols are enabled for the connection, SSL V2 is not negotiated even if it is enabled.
3. If TLS V1.2 is enabled on the client, establishment of SSL sessions with SSL V2 servers is not supported.

Cipher selection

The client sends a list of ciphers it supports during the SSL handshake. The server application uses this list, and the defined ciphers that are 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 servers cipher list for a match in the clients list. The first matching cipher is used.

When building the server's list of cipher suites for comparison with the list sent by the client, the server might omit some ciphers from the list as follows:

- When executing in an export level cryptographic environment, any ciphers that are not permitted for use in an export level environment.
- When executing in FIPS mode, any cipher suites that are not valid for use in FIPS mode.
- Any cipher suites that specify a *key algorithm* that is not supported for use with the server certificate's key. For example, if the cipher requires an RSA key algorithm but the server certificate uses a DSA key algorithm.
- When using protocol SSL V3.0 or lower, any cipher suites that specify Elliptic Curve Cryptography.
- When using protocol TLS V1.1 or lower, any cipher suites that specify:
 - A *sign key algorithm* that is not supported for use with the server certificate's key. For example, if the cipher requires a Diffie-Hellman certificate signed with an RSA signature, but the server certificate is a Diffie-Hellman certificate that is signed with a DSA signature.
 - SHA-2 message authentication.
 - AES-GCM encryption.
- When using protocol TLS V1.1 and higher, any cipher suites that specify 40-bit export encryption.
- When using protocol TLS V1.2 and higher, any cipher suites that specify:
 - 56-bit DES encryption.
 - A key algorithm that is not specified in the signature algorithm pairs list that is supplied by the client.

Note: For protocols TLS V1.1 and above, export cipher suites cannot be used. 40-bit ciphers is ignored if TLS V1.1 or above is negotiated as the security protocol. If TLS V1.1 or above is the intended protocol and only 40-bit ciphers are available, the connection fails with `GSK_ERR_NO_CIPHERS`.

Server certificate

The server certificate can use either RSA, DSA, Diffie-Hellman, or ECDSA as the public/private key algorithm.

In FIPS mode, the RSA or DSA key size must be at least 1024 bits, the Diffie-Hellman key size must be at least 2048 bits, and the ECC key size must be at least 192 bits and use a NIST-approved named curve.

An RSA certificate can be used with an RSA, ephemeral Diffie-Hellman, or ephemeral ECDH key exchange. A DSA certificate can be used with an ephemeral Diffie-Hellman key exchange. A Diffie-Hellman certificate can be used in a fixed Diffie-Hellman key exchange. An ECDSA certificate can be used with a fixed ECDH or ephemeral ECDH key exchange.

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.
- Diffie-Hellman certificates that are used in fixed Diffie-Hellman key exchange must allow key agreement.
- ECDSA certificates that are used in fixed ECDH key exchange must allow key agreement.
- ECDSA certificates that are used in ephemeral ECDH key exchange must allow digital signature.
- RSA certificates that are used in ephemeral ECDH key exchange must allow digital signature.
- DSA certificates using ephemeral Diffie-Hellman key exchange must allow digital signature.
- Other RSA certificates must allow key encipherment.

System SSL does not accept VeriSign Global Server ID certificates. When specified, System SSL uses these certificates as any other certificate when determining the encryption cipher to be used for the SSL session.

When using TLS V1.2 as the SSL session protocol, the client may pass to the server a list of signature algorithm pairs as part of the TLS handshake. The key algorithm and signature algorithm of the server certificate must be present in this list of signature algorithm pairs. In addition, any peer certificates in the server certificate chain must also be signed using a signature algorithm present in the list.

The signature algorithm pair list under the TLS V1.2 protocol may allow some TLS ciphers to operate using certificates that were previously incompatible with the cipher specification. In previous versions of TLS, these ciphers (primarily ciphers that use a fixed Diffie-Hellman or fixed ECDH key exchange) required the server certificate to be signed with a specific signature key algorithm. Under TLS V1.2, the signature algorithm pairs list allows the cipher to be used if the signature algorithm is specified in the list.

Client certificate

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 that is provided by the server. Server authentication requires that the provided certificate be validated. The validation process involves building the partner's certificate chain, using trusted certificates from the trusted certificate store and untrusted certificates that are provided through the handshake messages, as well as checking for certificate revocation information (LDAP CRLs, HTTP CRLs and/or OCSP responses). See "gsk_validate_certificate_mode()" on page 464 for a description how the certificate and revocation information is provided through data sources and the steps that are performed during certificate validation. Depending upon the server handshake type, the server may ask the client to

gsk_secure_socket_init()

provide its certificate. The key label that is stored in the connection is used to retrieve the certificate from the key database, PKCS #12 file, key ring, or token. The default key is used if no label is set. The key record must contain both an X.509 certificate and a private key. See “gsk_validate_certificate_mode()” on page 464 for a description of the steps that are performed during certificate validation.

The client certificate can use either RSA, Digital Signature Standard algorithm (DSA), ECDSA, or Diffie-Hellman as the public/private key algorithm. The type of client certificate that can be used depends on the key exchange method being used for the session cipher that is selected by the server, as detailed in the following list:

- RSA key exchange - RSA or DSA
- Fixed Diffie-Hellman key exchange - Diffie-Hellman
- Ephemeral Diffie-Hellman key exchange - RSA or DSA
- Fixed ECDH key exchange - RSA, DSA, or ECDSA
- Ephemeral ECDH key exchange - RSA, DSA, or ECDSA

Client certificates that are used in a fixed Diffie-Hellman or fixed ECDH key exchange where the client certificate is used to send the client's public key to the server must support key agreement. This means the certificate key usage extension (if any) must allow key agreement.

In all other cases the client certificate must support digital signatures. This means the certificate key usage extension (if any) must allow digital signature.

When using TLS V1.2 as the SSL session protocol, the server may pass to the client a list of signature algorithm pairs as part of the TLS handshake. The key algorithm and signature algorithm of the client certificate must be present in this list of signature algorithm pairs. In addition, any peer certificates in the client certificate chain must also be signed using a signature algorithm present in the list.

Specifying a cached session

When acting as a client, if the client intends to resume a newly created SSL V3, TLS V1.0, or higher session at a later time, immediately following the successful **gsk_secure_socket_init()** invocation to establish the connection, a call to **gsk_attribute_get_buffer()** needs to be performed in order to retrieve the peer ID (GSK_PEER_ID) for the newly generated connection. GSK_PEER_ID requires the enablement of the peer ID support by specifying environment attribute GSK_ENABLE_CLIENT_SET_PEERID to ON during the establishment of the SSL environment.

The GSK_PEER_ID value is used to locate a session entry in the client session cache so that it can be used to resume the session with the server if one exists and is not expired. If a matching session entry cannot be located by the server, a full handshake is performed.

The peer ID value provides information to locate the session information within the client session cache. Later, if the client wants to resume this session with the same server, as long as the SSL environment has not been closed, the client can set the peer ID to be used through a call to **gsk_attribute_set_buffer()** before calling **gsk_secure_socket_init()**. If the peer ID's cache entry is located, the following criteria is verified in order for this cache entry to be used:

- The requested cipher must be included within the session's cipher list.

- The requested protocol must be included within the session's supported protocol list.
- The certificate labels must match.

If a cached session is required, set the `GSK_REQ_CACHE_SESSION` attribute to `ON` prior to calling `gsk_secure_socket_init()`. `gsk_secure_socket_init()` fails if the cached entry cannot be found. If the session ID value from a server hello does not match the cached session ID, `gsk_secure_socket_init()` fails.

If a cached session is not required, ensure that the `GSK_REQ_CACHE_SESSION` attribute is `OFF` (the default) prior to calling `gsk_secure_socket_init()`. This results in a full handshake if the cached entry cannot be found.

When acting as a server, if the next SSL V3, TLS V1.0, or higher connection from a particular client needs to utilize a previously established session, after the call to `gsk_secure_socket_init()` to establish the connection, call `gsk_attribute_get_buffer()` to retrieve the session ID (`GSK_SID_VALUE`) information for the newly generated connection. Before calling `gsk_secure_socket_init()` for the next connection, call `gsk_attribute_set_buffer()` to set the session ID value.

If the cached session is required, a server application does not require setting attribute `GSK_REQ_CACHE_SESSION` to `ON` to enforce the reusing of a cached session. By setting the `GSK_SID_VALUE` prior to calling `gsk_secure_socket_init()`, the server is required to locate that cached session.

The `GSK_SID_VALUE` value is used to locate a session entry in the server session cache that can be used to resume the session with a client if one exists and has not expired. `gsk_secure_socket_init()` fails if the cached entry cannot be found.

If the session ID value from a client hello does not match the cached session ID, `gsk_secure_socket_init()` fails.

Related Topics

[“gsk_environment_init\(\)” on page 112](#)

[“gsk_secure_socket_write\(\)” on page 150](#)

[“gsk_secure_socket_read\(\)” on page 145](#)

[“gsk_secure_socket_misc\(\)” on page 142](#)

[“gsk_secure_socket_close\(\)” on page 132](#)

`gsk_secure_socket_misc()`

Performs miscellaneous secure connection functions.

Format

```
#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_NO_NEGOTIATION`]

An attempt was made to renegotiate a session when renegotiation is disabled.

[`GSK_ERR_NOT_SSLV3`]

The session is not using the SSL V3, TLS V1.0, or higher 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 higher 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. If using session ID caching, specifying GSK_RESET_SESSION causes the cache entry to be deleted for this session.

Note: Caution should be taken when specifying GSK_RESET_SESSION and reusing cached sessions.

- Specifying GSK_RESET_CIPHER causes a new cache entry to be created from the session if one did not already exist.
- For a client application with multiple connections reusing a cached session entry, the connection using **gsk_secure_socket_misc()** as well as current reused connections will continue to function properly. However, a new connection requiring the use of the cached entry will fail if cache reuse is required and the cache entry no longer exists. A new connection not requiring the use of the cached entry will result in a full handshake.
- For a server application with multiple connections reusing a cached session entry, the connection utilizing **gsk_secure_socket_misc()** as well as current reused connections will continue to function properly. However, a new connection requiring the use of the cached entry will fail if cache reuse is required and the cache entry no longer exists.
- For a server application, specify GSK_RENEGOTIATION_ABBREVIATED to ensure successful GSK_RESET_CIPHER when GSK_SID_VALUE is specified.

Related Topics

["gsk_secure_socket_open\(\)" on page 144](#)

["gsk_secure_socket_read\(\)" on page 145](#)

["gsk_secure_socket_write\(\)" on page 150](#)

gsk_secure_socket_open()

gsk_secure_socket_open()

Creates a secure socket connection.

Format

```
#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\(\)” on page 132](#)

[“gsk_secure_socket_init\(\)” on page 133](#)

gsk_secure_socket_read()

Reads data using a secure socket connection.

Format

```
#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_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[GSK_ERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[GSK_ERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[GSK_ERR_IO]

I/O error communicating with peer application.

gsk_secure_socket_read()

[GSK_ERR_NO_NEGOTIATION]

An attempt was made to renegotiate a session when renegotiation is disabled or the peer rejected an attempted session renegotiation.

[GSK_ERROR_RENEGOTIATION_INDICATION]

Peer did not signal support for TLS Renegotiation Indication.

[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]

A complete SSL record is not available.

[GSK_WOULD_BLOCK_WRITE]

An SSL handshake is in progress but data cannot be written to the socket.

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 does not 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()** if 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\(\)" on page 150](#)

["gsk_secure_socket_init\(\)" on page 133](#)

`gsk_secure_socket_shutdown()`

Shuts down a secure socket connection.

Format

```
#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 higher 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_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 higher 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 run time 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\(\)" on page 132](#)

["gsk_secure_socket_open\(\)" on page 144](#)

["gsk_secure_socket_read\(\)" on page 145](#)

["gsk_secure_socket_write\(\)" on page 150](#)

gsk_secure_socket_write()

Writes data using a secure socket connection.

Format

```
#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_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[GSK_ERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[GSK_ERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[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 because of an EWOULDBLOCK condition.

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()** if 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()" on page 145

"gsk_secure_socket_init()" on page 133

gsk_strerror()

gsk_strerror()

Return a text string for an SSL error code

Format

```
#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 is always a valid text string address even when the error code is not recognized (the return value is 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.

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 RFC 5280: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, RFC 3280: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile* and RFC 2459: *Internet x.509 Public Key Infrastructure Certificate and CRL Profile*. RFC 5280 obsoletes RFC 3280 which obsoletes RFC 2459.

Note: You can use the `gsk_strerror()` routine to return a text string describing a CMS error code. See “`gsk_strerror()`” on page 152 for more information.

This is a list of the Certificate Management Services (CMS) APIs:

- `gsk_add_record()` (see “`gsk_add_record()`” on page 159)
- `gsk_change_database_password()` (see “`gsk_change_database_password()`” on page 162)
- `gsk_change_database_record_length()` (see “`gsk_change_database_record_length()`” on page 164)
- `gsk_close_database()` (see “`gsk_close_database()`” on page 165)
- `gsk_close_directory()` (see “`gsk_close_directory()`” on page 166)
- `gsk_construct_certificate()` (see “`gsk_construct_certificate()`” on page 167)
- `gsk_construct_private_key()` (see “`gsk_construct_private_key()`” on page 171)
- `gsk_construct_private_key_rsa()` (see “`gsk_construct_private_key_rsa()`” on page 173)
- `gsk_construct_public_key()` (see “`gsk_construct_public_key()`” on page 175)
- `gsk_construct_public_key_rsa()` (see “`gsk_construct_public_key_rsa()`” on page 177)
- `gsk_construct_renewal_request()` (see “`gsk_construct_renewal_request()`” on page 178)
- `gsk_construct_self_signed_certificate()` (see “`gsk_construct_self_signed_certificate()`” on page 181)
- `gsk_construct_signed_certificate()` (see “`gsk_construct_signed_certificate()`” on page 184)
- `gsk_copy_attributes_signers()` (see “`gsk_copy_attributes_signers()`” on page 188)
- `gsk_copy_buffer()` (see “`gsk_copy_buffer()`” on page 189)
- `gsk_copy_certificate()` (see “`gsk_copy_certificate()`” on page 190)
- `gsk_copy_certificate_extension()` (see “`gsk_copy_certificate_extension()`” on page 191)
- `gsk_copy_certification_request()` (see “`gsk_copy_certification_request()`” on page 192)
- `gsk_copy_content_info()` (see “`gsk_copy_content_info()`” on page 193)

- **gsk_copy_crl()** (see “gsk_copy_crl()” on page 194)
- **gsk_copy_name()** (see “gsk_copy_name()” on page 195)
- **gsk_copy_private_key_info()** (see “gsk_copy_private_key_info()” on page 196)
- **gsk_copy_public_key_info()** (see “gsk_copy_public_key_info()” on page 197)
- **gsk_copy_record()** (see “gsk_copy_record()” on page 198)
- **gsk_create_certification_request()** (see “gsk_create_certification_request()” on page 199)
- **gsk_create_database()** (see “gsk_create_database()” on page 203)
- **gsk_create_database_renewal_request()** (see “gsk_create_database_renewal_request()” on page 205)
- **gsk_create_database_signed_certificate()** (see “gsk_create_database_signed_certificate()” on page 208)
- **gsk_create_renewal_request()** (see “gsk_create_renewal_request()” on page 214)
- **gsk_create_revocation_source()** (see “gsk_create_revocation_source()” on page 216)
- **gsk_create_self_signed_certificate()** (see “gsk_create_self_signed_certificate()” on page 222)
- **gsk_create_signed_certificate()** (see “gsk_create_signed_certificate()” on page 226)
- **gsk_create_signed_certificate_record()** (see “gsk_create_signed_certificate_record()” on page 229)
- **gsk_create_signed_certificate_set()** (see “gsk_create_signed_certificate_set()” on page 234)
- **gsk_create_signed_crl()** (see “gsk_create_signed_crl()” on page 239)
- **gsk_create_signed_crl_record()** (see “gsk_create_signed_crl_record()” on page 242)
- **gsk_decode_base64()** (see “gsk_decode_base64()” on page 246)
- **gsk_decode_certificate()** (see “gsk_decode_certificate()” on page 247)
- **gsk_decode_certificate_extension()** (see “gsk_decode_certificate_extension()” on page 248)
- **gsk_decode_certification_request()** (see “gsk_decode_certification_request()” on page 250)
- **gsk_decode_crl()** (see “gsk_decode_crl()” on page 251)
- **gsk_decode_import_certificate()** (see “gsk_decode_import_certificate()” on page 252)
- **gsk_decode_import_key()** (see “gsk_decode_import_key()” on page 253)
- **gsk_decode_issuer_and_serial_number()** (see “gsk_decode_issuer_and_serial_number()” on page 255)
- **gsk_decode_name()** (see “gsk_decode_name()” on page 256)
- **gsk_decode_private_key()** (see “gsk_decode_private key()” on page 257)
- **gsk_decode_public_key()** (see “gsk_decode_public key()” on page 258)
- **gsk_decode_signer_identifier()** (see “gsk_decode_signer_identifier()” on page 259)
- **gsk_delete_record()** (see “gsk_delete_record()” on page 260)
- **gsk_dn_to_name()** (see “gsk_dn_to_name()” on page 261)
- **gsk_encode_base64()** (see “gsk_encode_base64()” on page 264)
- **gsk_encode_certificate_extension()** (see “gsk_encode_certificate_extension()” on page 265)

- **gsk_encode_ec_parameters()** (see “gsk_encode_ec_parameters()” on page 267)
- **gsk_encode_export_certificate()** (see “gsk_encode_export_certificate()” on page 268)
- **gsk_encode_export_key()** (see “gsk_encode_export_key()” on page 270)
- **gsk_encode_export_request()** (see “gsk_encode_export_request()” on page 273)
- **gsk_encode_issuer_and_serial_number()** (see “gsk_encode_issuer_and_serial_number()” on page 274)
- **gsk_encode_name()** (see “gsk_encode_name()” on page 275)
- **gsk_encode_private_key()** (see “gsk_encode_private_key()” on page 276)
- **gsk_encode_public_key()** (see “gsk_encode_public_key()” on page 277)
- **gsk_encode_signature()** (see “gsk_encode_signature()” on page 278)
- **gsk_encode_signer_identifier()** (see “gsk_encode_signer_identifier()” on page 279)
- **gsk_export_certificate()** (see “gsk_export_certificate()” on page 280)
- **gsk_export_certification_request()** (see “gsk_export_certification_request()” on page 282)
- **gsk_export_key()** (see “gsk_export_key()” on page 284)
- **gsk_factor_private_key()** (see “gsk_factor_private_key()” on page 287)
- **gsk_factor_private_key_rsa()** (see “gsk_factor_private_key_rsa()” on page 288)
- **gsk_factor_public_key()** (see “gsk_factor_public_key()” on page 289)
- **gsk_factor_public_key_rsa()** (see “gsk_factor_public_key_rsa()” on page 290)
- **gsk_fips_state_query()** (see “gsk_fips_state_query()” on page 291)
- **gsk_fips_state_set()** (see “gsk_fips_state_set()” on page 292)
- **gsk_free_attributes_signers()** (see “gsk_free_attributes_signers()” on page 294)
- **gsk_free_buffer()** (see “gsk_free_buffer()” on page 295)
- **gsk_free_certificate()** (see “gsk_free_certificate()” on page 296)
- **gsk_free_certificates()** (see “gsk_free_certificates()” on page 297)
- **gsk_free_certificate_extension()** (see “gsk_free_certificate_extension()” on page 298)
- **gsk_free_certification_request()** (see “gsk_free_certification_request()” on page 299)
- **gsk_free_content_info()** (see “gsk_free_content_info()” on page 300)
- **gsk_free_crl()** (see “gsk_free_crl()” on page 301)
- **gsk_free_crls()** (see “gsk_free_crls()” on page 302)
- **gsk_free_decoded_extension()** (see “gsk_free_decoded_extension()” on page 303)
- **gsk_free_issuer_and_serial_number()** (see “gsk_free_issuer_and_serial_number()” on page 304)
- **gsk_free_name()** (see “gsk_free_name()” on page 305)
- **gsk_free_oid()** (see “gsk_free_oid()” on page 306)
- **gsk_free_private_key()** (see “gsk_free_private_key()” on page 307)
- **gsk_free_private_key_info()** (see “gsk_free_private_key_info()” on page 308)
- **gsk_free_public_key()** (see “gsk_free_public_key()” on page 309)
- **gsk_free_public_key_info()** (see “gsk_free_public_key_info()” on page 310)
- **gsk_free_record()** (see “gsk_free_record()” on page 311)
- **gsk_free_records()** (see “gsk_free_records()” on page 312)
- **gsk_free_revocation_source()** (see “gsk_free_revocation_source()” on page 313)

- **gsk_free_signer_identifier()** (see “gsk_free_signer_identifier()” on page 314)
- **gsk_free_string()** (see “gsk_free_string()” on page 315)
- **gsk_free_strings()** (see “gsk_free_strings()” on page 316)
- **gsk_generate_key_agreement_pair()** (see “gsk_generate_key_agreement_pair()” on page 317)
- **gsk_generate_key_pair()** (see “gsk_generate_key_pair()” on page 319)
- **gsk_generate_key_parameters()** (see “gsk_generate_key_parameters()” on page 322)
- **gsk_generate_random_bytes()** (see “gsk_generate_random_bytes()” on page 324)
- **gsk_generate_secret()** (see “gsk_generate_secret()” on page 325)
- **gsk_get_certificate_algorithms()** (see “gsk_get_certificate_algorithms()” on page 326)
- **gsk_get_certificate_info()** (see “gsk_get_certificate_info()” on page 327)
- **gsk_get_cms_vector()** (see “gsk_get_cms_vector()” on page 329)
- **gsk_get_content_type_and_cms_version()** (see “gsk_get_content_type_and_cms_version()” on page 331)
- **gsk_get_default_key()** (see “gsk_get_default_key()” on page 332)
- **gsk_get_default_label()** (see “gsk_get_default_label()” on page 333)
- **gsk_get_directory_certificates()** (see “gsk_get_directory_certificates()” on page 334)
- **gsk_get_directory_crls()** (see “gsk_get_directory_crls()” on page 336)
- **gsk_get_directory_enum()** (see “gsk_get_directory_enum()” on page 339)
- **gsk_get_directory_numeric_value()** (see “gsk_get_directory_numeric_value()” on page 341)
- **gsk_get_ec_parameters_info()** (see “gsk_get_ec_parameters_info()” on page 342)
- **gsk_get_record_by_id()** (see “gsk_get_record_by_id()” on page 343)
- **gsk_get_record_by_index()** (see “gsk_get_record_by_index()” on page 344)
- **gsk_get_record_by_label()** (see “gsk_get_record_by_label()” on page 345)
- **gsk_get_record_by_subject()** (see “gsk_get_record_by_subject()” on page 346)
- **gsk_get_record_labels()** (see “gsk_get_record_labels()” on page 347)
- **gsk_get_update_code()** (see “gsk_get_update_code()” on page 348)
- **gsk_import_certificate()** (see “gsk_import_certificate()” on page 349)
- **gsk_import_key()** (see “gsk_import_key()” on page 352)
- **gsk_make_content_msg()** (see “gsk_make_content_msg()” on page 355)
- **gsk_make_data_content()** (see “gsk_make_data_content()” on page 356)
- **gsk_make_data_msg()** (see “gsk_make_data_msg()” on page 357)
- **gsk_make_encrypted_data_content()** (see “gsk_make_encrypted_data_content()” on page 358)
- **gsk_make_encrypted_data_msg()** (see “gsk_make_encrypted_data_msg()” on page 360)
- **gsk_make_enveloped_data_content()** (see “gsk_make_enveloped_data_content()” on page 362)
- **gsk_make_enveloped_data_content_extended()** (see “gsk_make_enveloped_data_content_extended()” on page 365)
- **gsk_make_enveloped_data_msg()** (see “gsk_make_enveloped_data_msg()” on page 368)

- **gsk_make_enveloped_data_msg_extended()** (see “gsk_make_enveloped_data_msg_extended()” on page 371)
- **gsk_make_enveloped_private_key_msg()** (see “gsk_make_enveloped_private_key_msg()” on page 374)
- **gsk_make_signed_data_content()** (see “gsk_make_signed_data_content()” on page 377)
- **gsk_make_signed_data_content_extended()** (see “gsk_make_signed_data_content_extended()” on page 380)
- **gsk_make_signed_data_msg()** (see “gsk_make_signed_data_msg()” on page 384)
- **gsk_make_signed_data_msg_extended()** (see “gsk_make_signed_data_msg_extended()” on page 387)
- **gsk_make_wrapped_content()** (see “gsk_make_wrapped_content()” on page 391)
- **gsk_mktime()** (see “gsk_mktime()” on page 392)
- **gsk_modify_pkcs11_key_label()** (see “gsk_modify_pkcs11_key_label()” on page 393)
- **gsk_name_compare()** (see “gsk_name_compare()” on page 395)
- **gsk_name_to_dn()** (see “gsk_name_to_dn()” on page 396)
- **gsk_open_database()** (see “gsk_open_database()” on page 398)
- **gsk_open_database_using_stash_file()** (see “gsk_open_database_using_stash_file()” on page 400)
- **gsk_open_directory()** (see “gsk_open_directory()” on page 402)
- **gsk_open_keyring()** (see “gsk_open_keyring()” on page 403)
- **gsk_perform_kat()** (see “gsk_perform_kat()” on page 405)
- **gsk_query_crypto_level()** (see “gsk_query_crypto_level()” on page 406)
- **gsk_query_database_label()** (see “gsk_query_database_label()” on page 407)
- **gsk_query_database_record_length()** (see “gsk_query_database_record_length()” on page 408)
- **gsk_rdttime()** (see “gsk_rdttime()” on page 409)
- **gsk_read_content_msg()** (see “gsk_read_content_msg()” on page 410)
- **gsk_read_data_content()** (see “gsk_read_data_content()” on page 411)
- **gsk_read_data_msg()** (see “gsk_read_data_msg()” on page 412)
- **gsk_read_encrypted_data_content()** (see “gsk_read_encrypted_data_content()” on page 413)
- **gsk_read_encrypted_data_msg()** (see “gsk_read_encrypted_data_msg()” on page 415)
- **gsk_read_enveloped_data_content()** (see “gsk_read_enveloped_data_content()” on page 417)
- **gsk_read_enveloped_data_content_extended()** (see “gsk_read_enveloped_data_content_extended()” on page 419)
- **gsk_read_enveloped_data_msg()** (see “gsk_read_enveloped_data_msg()” on page 421)
- **gsk_read_enveloped_data_msg_extended()** (see “gsk_read_enveloped_data_msg_extended()” on page 423)
- **gsk_read_signed_data_content()** (see “gsk_read_signed_data_content()” on page 425)
- **gsk_read_signed_data_content_extended()** (see “gsk_read_signed_data_content_extended()” on page 428)
- **gsk_read_signed_data_msg()** (see “gsk_read_signed_data_msg()” on page 431)

- **gsk_read_signed_data_msg_extended()** (see “gsk_read_signed_data_msg_extended()” on page 434)
- **gsk_read_wrapped_content()** (see “gsk_read_wrapped_content()” on page 438)
- **gsk_receive_certificate()** (see “gsk_receive_certificate()” on page 439)
- **gsk_replace_record()** (see “gsk_replace_record()” on page 440)
- **gsk_set_default_key()** (see “gsk_set_default_key()” on page 443)
- **gsk_set_directory_enum()** (see “gsk_set_directory_enum()” on page 445)
- **gsk_set_directory_numeric_value()** (see “gsk_set_directory_numeric_value()” on page 447)
- **gsk_sign_certificate()** (see “gsk_sign_certificate()” on page 449)
- **gsk_sign_crl()** (see “gsk_sign_crl()” on page 452)
- **gsk_sign_data()** (see “gsk_sign_data()” on page 455)
- **gsk_validate_certificate()** (see “gsk_validate_certificate()” on page 458)
- **gsk_validate_certificate_mode()** (see “gsk_validate_certificate_mode()” on page 464)
- **gsk_validate_extended_key_usage()** (see “gsk_validate_extended_key_usage()” on page 472)
- **gsk_validate_hostname()** (see “gsk_validate_hostname()” on page 474)
- **gsk_validate_server()** (see “gsk_validate_server()” on page 476)
- **gsk_verify_certificate_signature()** (see “gsk_verify_certificate_signature()” on page 477)
- **gsk_verify_crl_signature()** (see “gsk_verify_crl_signature()” on page 480)
- **gsk_verify_data_signature()** (see “gsk_verify_data_signature()” on page 483)

gsk_add_record()

Adds a record to a key or request database.

Format

```
#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_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]

The record type is not supported for the database type.

gsk_add_record()

[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.

[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.

gsk_add_record()

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.

Except for 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

```
#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.

[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

```
#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()

gsk_close_directory()

Closes an LDAP directory.

Format

```
#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 closes an LDAP directory opened by the **gsk_open_directory()** routine. The *directory_handle* is not valid upon return from the **gsk_close_directory()** routine.

gsk_construct_certificate()

Constructs a signed certificate and returns it to the caller.

Format

```
#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_public_key_info *   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_EC_PARAMS]

Elliptic Curve parameters are not valid.

[CMSERR_BAD_KEY_SIZE]

The key size is not valid.

gsk_construct_certificate()

[CMSERR_BAD_SUBJECT_NAME]

The subject name is not valid.

[CMSERR_CA_NOT_SUPPLIED]

Signing Certificate Authority Certificate not supplied.

[CMSERR_DUPLICATE_EXTENSION]

Supplied extensions contain a duplicate extension.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_EXPIRED]

The signer certificate is expired.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[CMSERR_INCORRECT_KEY_TYPE]

Incorrect key algorithm.

[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_MISMATCH]

The signer certificate key cannot be used to sign a certificate or the key type is not supported for the requested signature algorithm.

[CMSERR_NO_MEMORY]

Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]

The signer certificate does not have a private key.

[CMSERR_SUBJECT_IS_CA]

The requested subject name is the same as the signer name.

Usage

The **gsk_construct_certificate()** routine will construct an X.509 certificate as described in RFC 5280: *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.

- If the supplied *public_key* contains a Diffie-Hellman key, the *issuer_certificate* must contain either an RSA or a DSA key.
- If the supplied *public_key* is an ECC key, the *issuer_certificate* cannot contain a DSA key.

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.
- An ECC key can be used for authentication, digital signature and key agreement. An ECC key can be used for both CA certificates and 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}

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_dsaWithSha224

Digital Signature Standard with SHA-224 digest – {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256

Digital Signature Standard with SHA-256 digest – {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1

Elliptic Curve Digital Signature Algorithm with SHA-1 digest – {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224

Elliptic Curve Digital Signature Algorithm with SHA-224 digest – {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256

Elliptic Curve Digital Signature Algorithm with SHA-256 digest – {1.2.840.10045.4.3.2}

gsk_construct_certificate()

x509_alg_ecdsaWithSha384

Elliptic Curve Digital Signature Algorithm with SHA-384 digest –
{1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512

Elliptic Curve Digital Signature Algorithm with SHA-512 digest –
{1.2.840.10045.4.3.4}

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()

Constructs a private key from its component values.

Format

```
#include <gskcms.h>

gsk_status gsk_construct_private_key (
    gsk_private_key *      private_key_factors,
    pkcs_private_key_info * private_key)
```

Parameters

private_key_factors

Specifies the private key structure containing the key algorithm type and private key components.

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:

[CMSERR_ALG_NOT_SUPPORTED]

Cryptographic algorithm is not supported.

[CMSERR_BASE_NOT_SUPPLIED]

Base not supplied.

[CMSERR_COEFFICIENT_NOT_SUPPLIED]

CRT Coefficient not supplied.

[CMSERR_EC_PARAMETERS_NOT_SUPPLIED]

EC parameters not supplied.

[CMSERR_MODULUS_NOT_SUPPLIED]

Modulus not supplied.

[CMSERR_PRIME_EXPONENT1_NOT_SUPPLIED]

First prime exponent not supplied.

[CMSERR_PRIME_EXPONENT2_NOT_SUPPLIED]

Second prime exponent not supplied.

[CMSERR_PRIME_NOT_SUPPLIED]

Prime not supplied.

[CMSERR_PRIME1_NOT_SUPPLIED]

First prime not supplied.

[CMSERR_PRIME2_NOT_SUPPLIED]

Second prime not supplied.

[CMSERR_PRIVATE_EXPONENT_NOT_SUPPLIED]

Private exponent not supplied.

[CMSERR_PRIVATE_KEY_INFO_NOT_SUPPLIED]

Private key information not supplied.

[CMSERR_PRIVATE_KEY_NOT_SUPPLIED]

Private key structure not supplied.

gsk_construct_private_key()

[CMSERR_PRIVATE_VALUE_NOT_SUPPLIED]

Private value not supplied.

[CMSERR_PUBLIC_EXPONENT_NOT_SUPPLIED]

Public exponent not supplied.

[CMSERR_STRUCTURE_TOO_SMALL]

Size specified for supplied structure is too small.

[CMSERR_SUB_PRIME_NOT_SUPPLIED]

Sub-prime not supplied.

Usage

The **gsk_construct_private_key()** function constructs the `pkcs_private_key_info` from the supplied private key components. The format of the supplied components is as stored in ICSF PKCS #11 tokens.

Before calling the function, the application must initialize the size field in *private_key_factors* to the size of the `gsk_private_key` structure. It must also prime *private_key_factors* with the `x509_algorithm_identifier`, including appropriate private key components for the private key type being constructed.

The `x509_algorithm_identifier` in *private_key* is set with the appropriate value for the private key type when returned.

gsk_construct_private_key_rsa()

Constructs an RSA private key from its component values.

Note: This function is deprecated. Use `gsk_construct_private_key()` instead.

Format

```
#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 $q^{-1} \bmod p$.

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` structures `x509_algorithm_identifier` is set with `x509_alg_rsaEncryption`, while version

gsk_construct_private_key_rsa()

specifies 0.

gsk_construct_public_key()

Constructs a public key from its component values

Format

```
#include <gskcms.h>

gsk_status gsk_construct_public_key(
    gsk_public_key *      public_key_factors,
    x509_public_key_info * public_key)
```

Parameters

public_key_factors

Specifies the public key structure containing the key algorithm type and public key components.

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:

[CMSERR_ALG_NOT_SUPPORTED]

Cryptographic algorithm not supported

[CMSERR_BASE_NOT_SUPPLIED]

Base not supplied

[CMSERR_EC_PARAMETERS_NOT_SUPPLIED]

EC parameters not supplied.

[CMSERR_MODULUS_NOT_SUPPLIED]

Modulus not supplied

[CMSERR_PRIME_NOT_SUPPLIED]

Prime not supplied

[CMSERR_PUBLIC_EXPONENT_NOT_SUPPLIED]

Public exponent not supplied

[CMSERR_PUBLIC_KEY_INFO_NOT_SUPPLIED]

Public key information not supplied

[CMSERR_PUBLIC_KEY_NOT_SUPPLIED]

Public key structure not supplied

[CMSERR_PUBLIC_VALUE_NOT_SUPPLIED]

Public value not supplied

[CMSERR_STRUCTURE_TOO_SMALL]

Size specified for supplied structure is too small

[CMSERR_SUB_PRIME_NOT_SUPPLIED]

Sub-prime not supplied

gsk_construct_public_key()

Usage

The **gsk_construct_public_key()** function constructs the `x509_public_key_info` from the supplied public key components. The format of the supplied components is as stored in ICSF PKCS #11 tokens.

Before calling the function, the application must initialize the size field in *public_key_factors* to the size of the `gsk_public_key` structure. It must also prime *public_key_factors* with the `x509_algorithm_identifier`, including appropriate public key components for the public key type being constructed.

The `x509_algorithm_identifier` in *public_key* is set with the appropriate value for the public key type when returned.

gsk_construct_public_key_rsa()

Constructs an RSA public key from its component values.

Note: This function is deprecated. Use `gsk_construct_public_key()` instead.

Format

```
#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` structures `x509_algorithm_identifier` is set with `x509_alg_rsaEncryption`.

`gsk_construct_renewal_request()`

Constructs a certification renewal request as described in PKCS #10, Version 1.7: *Certification Request*.

Format

```
#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()` routine constructs a certification renewal request and returns the constructed request in the `pkcs_cert_request` structure *request*.

The `gsk_encode_export_request()` routine 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}

x509_alg_dsaWithSha224

Digital Signature Standard with SHA-224 digest - {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256

Digital Signature Standard with SHA-256 digest - {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1

Elliptic Curve Digital Signature Algorithm with SHA-1 digest - {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224

Elliptic Curve Digital Signature Algorithm with SHA-224 digest - {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256

Elliptic Curve Digital Signature Algorithm with SHA-256 digest - {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384

Elliptic Curve Digital Signature Algorithm with SHA-384 digest - {1.2.840.10045.4.3.3}

gsk_construct_renewal_request()

x509_alg_ecdsaWithSha512

Elliptic Curve Digital Signature Algorithm with SHA-512 digest -
{1.2.840.10045.4.3.4}

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

```
#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_EC_PARAMS]

Elliptic Curve parameters are not valid.

[CMSERR_BAD_KEY_SIZE]

The key size is not valid.

gsk_construct_self_signed_certificate()

[CMSERR_BAD_SUBJECT_NAME]

The subject name is not valid.

[CMSERR_DUPLICATE_EXTENSION]

Supplied extensions contain a duplicate extension.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[CMSERR_KEY_MISMATCH]

The signer key cannot be used to sign a certificate or the key type is not supported for the requested signature algorithm.

[CMSERR_NO_MEMORY]

Insufficient storage is available.

Usage

The **gsk_construct_self_signed_certificate()** routine will construct an X.509 certificate as described in RFC 5280: *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}

x509_alg_dsaWithSha224

Digital Signature Standard with SHA-224 digest - {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256

Digital Signature Standard with SHA-256 digest - {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1

Elliptic Curve Digital Signature Algorithm with SHA-1 digest - {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224

Elliptic Curve Digital Signature Algorithm with SHA-224 digest - {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256

Elliptic Curve Digital Signature Algorithm with SHA-256 digest - {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384

Elliptic Curve Digital Signature Algorithm with SHA-384 digest - {1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512

Elliptic Curve Digital Signature Algorithm with SHA-512 digest - {1.2.840.10045.4.3.4}

When executing in non-FIPS mode, an RSA key size must be between 512 and 4096 bits. A DSA key size must be between 512 and 2048 bits. An ECC key can be a NIST recommended named curve with a key between 192 and 521 or a Brainpool curve with a key between 160 and 512.

When executing in FIPS mode:

- Signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.
- An RSA key size must be between 1024 and 4096 bits.
- A DSA key size must be either 1024 bits or 2048 bits.
- An NIST ECC key must be a NIST recommended named curve with a key size between 192 and 521.

For more information about FIPS supported algorithms and key sizes, see “Algorithms and key sizes” on page 19.

A DSA key size of 1024 or less should specify signature algorithm x509_alg_dsaWithSha1, while a key size of 2048 bits should specify either x509_alg_dsaWithSha224 or x509_alg_dsaWithSha256 as the signature algorithm.

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 before 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. However, 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

```
#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 DERencoded 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_EC_PARAMS]

Elliptic Curve parameters are 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.

[CMSERR_DUPLICATE_EXTENSION]

Supplied extensions contain a duplicate extension.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_EXPIRED]

The signer certificate is expired.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[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_MISMATCH]

The signer certificate key cannot be used to sign a certificate or the key type is not supported for the requested signature algorithm.

[CMSERR_NO_MEMORY]

Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]

The signer certificate does not have a private key.

[CMSERR_REQUEST_NOT_SUPPLIED]

Certificate request not supplied.

[CMSERR_SUBJECT_IS_CA]

The requested subject name is the same as the signer name.

Usage

The **gsk_construct_signed_certificate()** routine will construct an X.509 certificate as described in RFC 5280: *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

gsk_construct_signed_certificate()

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}

x509_alg_dsaWithSha224

Digital Signature Standard with SHA-224 digest - {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256

Digital Signature Standard with SHA-256 digest - {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1

Elliptic Curve Digital Signature Algorithm with SHA-1 digest –
{1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224

Elliptic Curve Digital Signature Algorithm with SHA-224 digest –
{1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256

Elliptic Curve Digital Signature Algorithm with SHA-256 digest –
{1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384

Elliptic Curve Digital Signature Algorithm with SHA-384 digest –
{1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512

Elliptic Curve Digital Signature Algorithm with SHA-512 digest –
{1.2.840.10045.4.3.4}

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()

gsk_copy_attributes_signers()

Copies a `gsk_attributes_signers` structure.

Format

```
#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

```
#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()

gsk_copy_certificate()

Copies an X.509 certificate.

Format

```
#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()

gsk_copy_certification_request()

Copies a PKCS #10 certification request.

Format

```
#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()

Copies PKCS #7 content information.

Format

```
#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.

gsk_copy_crl()

gsk_copy_crl()

Copies an X.509 certificate revocation list.

Format

```
#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

```
#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()

gsk_copy_private_key_info()

Copies the private key information.

Format

```
#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

```
#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()

gsk_copy_record()

Copies a database record.

Format

```
#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 as described in PKCS #10, Version 1.7: *Certification Request*.

Format

```
#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.

gsk_create_certification_request()

- [CMSERR_FIPS_KEY_PAIR_CONSISTENCY]
FIPS mode key generation failed pair-wise consistency check.
- [CMSERR_ICSF_CLEAR_KEY_SUPPORT_NOT_AVAILABLE]
Clear key support not available due to ICSF key policy.
- [CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.
- [CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.
- [CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.
- [CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.
- [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 PKCS #10 certification request. 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}
- x509_alg_dsaWithSha224**
Digital Signature Standard with SHA-224 digest - {2.16.840.1.101.3.4.3.1}
- x509_alg_dsaWithSha256**
Digital Signature Standard with SHA-256 digest - {2.16.840.1.101.3.4.3.2}
- x509_alg_ecdsaWithSha1**
Elliptic Curve Digital Signature Algorithm with SHA-1 digest -
{1.2.840.10045.4.1}
- x509_alg_ecdsaWithSha224**
Elliptic Curve Digital Signature Algorithm with SHA-224 digest -
{1.2.840.10045.4.3.1}
- x509_alg_ecdsaWithSha256**
Elliptic Curve Digital Signature Algorithm with SHA-256 digest -
{1.2.840.10045.4.3.2}
- x509_alg_ecdsaWithSha384**
Elliptic Curve Digital Signature Algorithm with SHA-384 digest -
{1.2.840.10045.4.3.3}
- x509_alg_ecdsaWithSha512**
Elliptic Curve Digital Signature Algorithm with SHA-512 digest -
{1.2.840.10045.4.3.4}

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 2048 bits. Key sizes of between 512 and 1024 bits are rounded up to a multiple of 64, key size 2048 must be explicitly specified as such. A key size of 1024 or less should specify signature algorithm `x509_alg_dsaWithSha1`, while a key size of 2048 bits should specify either `x509_alg_dsaWithSha224` or `x509_alg_dsaWithSha256` as the signature algorithm.

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 either 1024 bits or 2048 bits. A key size of 1024 bits should specify signature algorithm `x509_alg_dsaWithSha1`, while a key size of 2048 bits should specify either `x509_alg_dsaWithSha224` or `x509_alg_dsaWithSha256` as the signature algorithm.

For an ECC key the key size will determine the default named curve that will be used for the public/private key pair, as specified in Table 3 on page 15. In FIPS mode, only NIST recommended that curves are supported. To specify a specific supported elliptic curve, use `gsk_construct_renewal_request()` to create a certificate request.

gsk_create_certification_request()

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

```
#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 that 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.

gsk_create_database()

[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_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_EC_PARAMS]

Elliptic Curve parameters are not valid.

[CMSERR_BAD_HANDLE]

The database handle is not valid.

gsk_create_database_renewal_request()

[CMSERR_BAD_KEY_SIZE]

The key size is not valid.

[CMSERR_BAD_LABEL]

The record label is not valid.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[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_PRIVATE_KEY_INFO_NOT_SUPPLIED]

Private key information not supplied.

[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, Version 1.7: *Certification Request*. 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

gsk_create_database_renewal_request()

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}

x509_alg_dsaWithSha1

Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_dsaWithSha224

Digital Signature Standard with SHA-224 digest - {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256

Digital Signature Standard with SHA-256 digest - {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1

Elliptic Curve Digital Signature Algorithm with SHA-1 digest - {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224

Elliptic Curve Digital Signature Algorithm with SHA-224 digest - {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256

Elliptic Curve Digital Signature Algorithm with SHA-256 digest - {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384

Elliptic Curve Digital Signature Algorithm with SHA-384 digest - {1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512

Elliptic Curve Digital Signature Algorithm with SHA-512 digest - {1.2.840.10045.4.3.4}

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.

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_EC_PARAMS]

Elliptic Curve parameters are not valid.

[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_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_EXPIRED]

The signer certificate is expired.

[CMSERR_FIPS_KEY_PAIR_CONSISTENCY]

FIPS mode key generation failed pair-wise consistency check.

[CMSERR_ICSF_CLEAR_KEY_SUPPORT_NOT_AVAILABLE]

Clear key support not available due to ICSF key policy.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]

The database type does not support certificates.

gsk_create_database_signed_certificate()

[CMSERR_INCORRECT_KEY_TYPE]

Incorrect key algorithm

[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 5280: *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.

- If the specified certificate key is a Diffie-Hellman key, the *signature_algorithm* must specify either an RSA or a DSA signature.
- If the specified certificate key is an ECC key, the *signature_algorithm* cannot specify a DSA signature.

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_create_database_signed_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}

x509_alg_dsaWithSha224

Digital Signature Standard with SHA-224 digest - {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256

Digital Signature Standard with SHA-256 digest - {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1

Elliptic Curve Digital Signature Algorithm with SHA-1 digest -
{1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224

Elliptic Curve Digital Signature Algorithm with SHA-224 digest -
{1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256

Elliptic Curve Digital Signature Algorithm with SHA-256 digest -
{1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384

Elliptic Curve Digital Signature Algorithm with SHA-384 digest -
{1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512

Elliptic Curve Digital Signature Algorithm with SHA-512 digest -
{1.2.840.10045.4.3.4}

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.
- An ECC key can be used for authentication, digital signature, and 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:

x509_alg_rsaEncryption

RSA encryption - {1.2.840.113549.1.1.1}

gsk_create_database_signed_certificate()

x509_alg_idDsa

Digital Signature Standard (DSS) - {1.2.840.10040.4.1}

x509_alg_dhPublicNumber

Diffie-Hellman (DH) - {1.2.840.10046.2.1}

x509_alg_ecPublicKey

Elliptic Curve Public Key (ECC) - {1.2.840.10045.2.1}

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 sizes of between 512 and 1024 bits when in non-FIPS mode are rounded up to a multiple of 64.
- Key sizes less than 1024 bits can only be generated in non-FIPS mode and are generated according to FIPS 186-2.
- Key sizes of 1024 or 2048 bits are generated according to FIPS 186-3 in both FIPS mode and non-FIPS mode. These are the only valid key sizes in FIPS mode.
- A key size of 1024 or less should specify `x509_alg_dsaWithSha1` as the signature algorithm, while a key size of 2048 bits should specify either `x509_alg_dsaWithSha224` or `x509_alg_dsaWithSha256` as the signature algorithm.
- Key parameters encoded as an ASN.1 sequence consisting of the prime p , the prime divisor q , and the generator g . For 1024-bit and 2048-bit keys, see FIPS 186-3: *Digital Signature Standard (DSS)* for more information about the key parameters, for smaller key sizes see FIPS 186-2: *Digital Signature Standard (DSS)*. Note that key parameters that contain a p of 2048 bits and a q of 160 bits do not conform to FIPS 186-3 and are not supported. The **`gsk_generate_key_parameters()`** routine can be used to generate the key parameters.

DH keys

- Can be used only for end user certificates
- Can only be signed using a certificate containing either an RSA or DSA key
- 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 , and optionally the subprime Q and the subgroup factor J . See RFC 2631: *Diffie-Hellman Key Agreement Method* for more information about the key parameters for non-FIPS mode, and see *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for FIPS mode. The **`gsk_generate_key_parameters()`** routine can be used to generate the key parameters.

ECC keys

- Can be used for both CA certificates and end user certificates.

gsk_create_database_signed_certificate()

- The ECC named curve used to generate the ECC key pair can be specified using either the *key_parameters* buffer or the *key_size* parameter. If the *key_parameters* buffer is supplied the *key_size* parameter will be ignored.
- The *key_parameters* buffer must contain ASN.1 encoded ECC parameters, or be NULL.
- If the *key_parameters* buffer is not supplied, the *key_size* parameter will be rounded up to the nearest supported key size and the default EC named curve for that key size will be used, as specified in Table 3 on page 15.
- In FIPS mode, only NIST recommended curves are supported.

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()

gsk_create_renewal_request()

Creates a PKCS #10 certification renewal request.

This function is deprecated. Use **gsk_create_database_renewal_request()** instead.

Format

```
#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_EC_PARAMS]

Elliptic Curve parameters are not valid.

[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_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.
- [CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.
- [CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.
- [CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.
- [CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.
- [CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.
- [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_PRIVATE_KEY_INFO_NOT_SUPPLIED]
Private key information not supplied.
- [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_renewal_request()** routine creates a certification request as described in PKCS #10, Version 1.7: *Certification Request*. 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_revocation_source()

gsk_create_revocation_source()

Create an OCSP, HTTP CRL, or an extended LDAP CRL revocation source.

Format

```
#include <gskcms.h>

gsk_status gsk_create_revocation_source (
                                gskdb_source *    source,
                                gsk_handle *      revocation_handle)
```

Parameters

source

Specifies the parameters needed for the revocation source handle to be created.

revocation_handle

Returns the revocation data source handle. The application should call the **gsk_free_revocation_source()** routine when it no longer needs access to the revocation source.

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_ATTRIBUTE_INVALID_ENUMERATION]

The enumeration value specified in the source is not valid.

[CMSERR_BAD_HANDLE]

The source handle is NULL or the type field within the source handle is not valid.

[CMSERR_INVALID_NUMERIC_VALUE]

Numeric value is not valid.

[CMSERR_MUTEX_ERROR]

Mutex request failed.

[CMSERR_NO_MEMORY]

Insufficient storage is available.

[CMSERR_REQUIRED_PARAMETER_NOTSET]

Required parameter is not set.

Usage

The **gsk_create_revocation_source()** routine creates a revocation data source to be used in the **gsk_validate_certificate()** or **gsk_validate_certificate_mode()** routines.

The source field has support for three different sources:

LDAP extended CRL support

This allows for a CRL to be obtained from an LDAP server.

HTTP CDP (Certificate Distribution Point) CRL support

This allows for a CRL to be obtained from an HTTP server.

OCSP support

This allows for certificate revocation information to be obtained from an OCSP responder.

The *gskdb_source* structure is defined in the **gskcms.h** include file. This structure contains a source type and a union which contains an embedded source structure. The *source_type* parameter defines the revocation source type and the embedded source structure contains the enablement information. The embedded source structure should be set to binary zeros and then populated with relevant information for the data source. Unused and reserved fields must contain binary zeros.

Extended LDAP directory source

If an extended LDAP directory source is specified in the *gskdb_source* structure, this indicates that an extended LDAP directory source is to be created. The parameters in the *gskdb_extended_directory_source* structure are in Table 11.

Table 11. *gskdb_extended_directory_source* parameters

Parameter name	Description
<i>crlCacheEntryMaxSize</i>	Specifies the maximum size in bytes of a CRL that is allowed to be stored in the LDAP CRL cache. The valid cache entry sizes are 0 through 32000. A size of 0 means there is no limit on the size of the CRL.
<i>crlCacheSize</i>	Specifies the maximum number of CRLs that may reside at one time in the LDAP CRL cache. The valid cache sizes are -1 through 32000. If caching is desired, this parameter should be set to a value greater than 0 or -1. When set to a non-zero value, CRLs are only cached if they contain a nextUpdate value that is later than the current time. The cached CRL will stay in the cache based upon its nextUpdate time value. A value of -1 means there is no limit on the cache size. A value of 0 disables caching.
<i>crlCacheTempCRL</i>	Specifies if a temporary CRL is to be added to the LDAP CRL cache when the CRL does not reside on the LDAP server: TRUE Add entry. FALSE Do not add entry.
<i>crlCacheTempCRLTimeout</i>	Specifies the amount of time in hours that a temporary CRL can reside in the LDAP CRL cache. Valid hours are 1 through 720. This parameter should be set to a non-zero value when <i>crlCacheTempCRL</i> is set to TRUE.
<i>crlSecurityLevel</i>	Specifies the security level when attempting to contact an LDAP server for the certificate revocation list (CRL). Supported security level are: <ul style="list-style-type: none"> • GSKCMS_CRL_SECURITY_LEVEL_LOW • GSKCMS_CRL_SECURITY_LEVEL_MEDIUM • GSKCMS_CRL_SECURITY_LEVEL_HIGH See “gsk_set_directory_enum()” on page 445 for more information about the supported security levels.
<i>ldapPassword</i>	Specifies the password for the LDAP user (if one was specified).
<i>ldapPort</i>	Specifies the LDAP port for the LDAP server. Valid ports are 0 through 65535. A value of 0 means the default LDAP port of 389 is used.

gsk_create_revocation_source()

Table 11. *gskdb_extended_directory_source* parameters (continued)

Parameter name	Description
<i>ldapResponseTimeout</i>	Specifies the time limit in seconds to wait for a response from the LDAP server. The valid time limits are 0 through 43200 seconds (12 hours). A value of 0 means there is no time limit.
<i>ldapServerName</i>	Specifies one or more blank-separated LDAP server host names.
<i>ldapUser</i>	Specifies the LDAP bind distinguished name (DN) used to authenticate with the LDAP server. If this is set to NULL, an anonymous authentication is used to authenticate with the LDAP server.

CDP source

If an CDP source is specified in the *gskdb_source* structure, this indicates that an HTTP CDP data source is to be created. The parameters in the *gskdb_cdp_source* structure are in Table 12.

Table 12. *gskdb_ocsp_source* structure parameters

Parameter name	Description
<i>cdpEnableFlags</i>	Enables the usage of the CDP extension for HTTP URIs. <i>cdpEnableFlags</i> must be set to either GSKCMS_CDP_ENABLE_HTTP or GSKCMS_CDP_ENABLE_ANY.
<i>httpCdpCacheEntryMaxSize</i>	Specifies the maximum size in bytes of a CRL that is allowed to be stored in the HTTP CDP CRL cache. The valid sizes are 0 through 2147483647. A size of 0 means there is no limit on the size of the CRL.
<i>httpCdpCacheSize</i>	Specifies the maximum number of CRLs that are allowed to be stored in the HTTP CDP CRL cache. The valid sizes are 0 through 32000. If caching is desired, this parameter should be set to a value greater than 0. When set to a non-zero value, CRLs are only cached if they contain an <i>nextUpdate</i> value that is later than the current time. The cached CRL will stay in the cache based upon its <i>nextUpdate</i> time value. A value of 0 disables caching.
<i>httpCdpMaxResponseSize</i>	Specifies the maximum response size in bytes that the application will accept from an HTTP server. The valid sizes are 0 through 2147483647. A size of 0 means there is no limit on the size of the response.
<i>httpCdpProxyServerName</i>	Specifies an HTTP proxy server to use when making a connection to obtain an HTTP CRL. A value of NULL indicates no proxy server. When non-NULL, the proxy server hostname or IP address must be specified. When specified, the <i>httpCdpProxyServerPort</i> must be set to a value between 1 and 65535.
<i>httpCdpProxyServerPort</i>	Specifies the HTTP proxy server port to use when making a connection to obtain an HTTP CRL. This port is only used when <i>httpCdpProxyServerName</i> is not NULL. Valid ports are 1 through 65535.
<i>httpCdpResponseTimeout</i>	Specifies the time in seconds to wait for a response from an HTTP server. The valid time limits are 0 through 43200 seconds (12 hours). A value of 0 means there is no time limit.

OCSP source

If an OCSP source is specified in the *gskdb_source* structure, this indicates that an OCSP data source is to be created. The *ocspEnable* parameter must be set to TRUE or *ocspURL* must be specified for an OCSP revocation source to be successfully created. The parameters in the *gskdb_ocsp_source* structure are in Table 13.

Table 13. *gskdb_ocsp_source* structure parameters

Parameter name	Description
<i>ocspCacheEntryMaxSize</i>	Specifies the maximum number of cached certificate statuses to store under an OCSP cache entry block. An OCSP cache entry block is used to store the certificate statuses of certificates with the same issuer's name, issuer's key name hash, and the URL of the OCSP responder contacted. The size must be between 0 and 32000 and must be less than or equal to the size specified for <i>ocspCacheSize</i> . This size is rounded up to the nearest power of 16. The size must be greater than or equal to 0. A size of 0 means there is no limit on the number of entries stored under an OCSP cache entry block.
<i>ocspCacheSize</i>	Specifies the maximum number of OCSP certificate statuses to store in the OCSP cache. This number is rounded up to the nearest power of 16. The size must be between 0 and 32000. If caching is desired, this parameter should be set to a value greater than 0. A value of 0 disables caching.
<i>ocspCheckNonce</i>	Specifies if there should be a nonce value included in the OCSP response: <p>TRUE Nonce in the OCSP response is checked to verify that it is the same as the one that was sent on the OCSP request.</p> <p>FALSE No nonce checking is performed.</p> <p>If set to TRUE, <i>ocspNonceSize</i> must be greater than or equal to 8.</p>
<i>ocspDbHandle</i>	Specifies the database handle returned by the gsk_open_database() routine or the gsk_open_keyring() routine that contains the certificate to be used to sign OCSP requests. This field is required when a non-NULL label is provided in the <i>ocspReqLabel</i> field.
<i>ocspEnable</i>	Specifies if the certificate's AIA extension is queried to obtain the URI values for the OCSP responder to contact: <p>TRUE The URI of the OCSP responder is retrieved from the certificate's AIA extension.</p> <p>FALSE The OCSP responder is not obtained from the AIA extension.</p>
<i>ocspGenerateNonce</i>	Specifies if a nonce should be generated and sent in the OCSP request: <p>TRUE The nonce is sent.</p> <p>FALSE The nonce is not sent.</p> <p>If set to TRUE, <i>ocspNonceSize</i> must be greater than or equal to 8.</p>

gsk_create_revocation_source()

Table 13. *gskdb_ocsp_source* structure parameters (continued)

Parameter name	Description
<i>ocspMaxResponseSize</i>	Specifies the maximum response size in bytes to be accepted from an OCSP responder. The valid response sizes are 0 through 2147483647 and defaults to 20480. A size of 0 means there is no limit on the size of the response.
<i>ocspNonceSize</i>	Specifies the size of the nonce that will be sent to the OCSP responder when nonce support is enabled (<i>ocspGenerateNonce</i> is TRUE). The valid nonce sizes are 8 through 256 and defaults to 8.
<i>ocspProxyServerName</i>	Specifies an OCSP proxy server to use when making a connection to obtain an OCSP response. A value of NULL indicates no proxy server. When non-NULL, the proxy server hostname or IP address must be specified. When specified, the <i>ocspProxyServerPort</i> must be set to a value between 1 and 65535.
<i>ocspProxyServerPort</i>	Specifies the OCSP proxy server port to use when making a connection to obtain an OCSP response. This field is only used when <i>ocspProxyServerName</i> is not NULL. Valid ports are 1 through 65535.
<i>ocspReqLabel</i>	Specifies the label of the certificate to be used to sign OCSP requests. The label is specified in the local code page. When specified, the <i>ocspDbHandle</i> parameter is required.
<i>ocspReqSignatureAlgorithm</i>	<p>Specifies the signature algorithm to be used to sign OCSP requests. The supported signature algorithms are:</p> <ul style="list-style-type: none"> • x509_alg_md5WithRsaEncryption • x509_alg_sha1WithRsaEncryption • x509_alg_dsaWithSha1 • x509_alg_ecdsaWithSha1 • x509_alg_sha224WithRsaEncryption • x509_alg_dsaWithSha224 • x509_alg_ecdsaWithSha224 • x509_alg_sha256WithRsaEncryption • x509_alg_dsaWithSha256 • x509_alg_ecdsaWithSha256 • x509_alg_sha384WithRsaEncryption • x509_alg_ecdsaWithSha384 • x509_alg_sha512WithRsaEncryption • x509_alg_ecdsaWithSha512 <p>To have the default signature algorithm used (RSA with SHA-256), set to 0.</p> <p>Note: When executing in FIPS mode, signature algorithm x509_alg_md5WithRSAEncryption is not supported.</p>
<i>ocspResponseTimeout</i>	Specifies the time in seconds to wait for a response from an OCSP responder. The valid time limits are 0 through 43200 seconds (12 hours) and defaults to 30 seconds. A value of 0 means there is no time limit.

Table 13. gskdb_ocsp_source structure parameters (continued)

Parameter name	Description
<i>ocspURL</i>	<p>Specifies the default URL that is used when contacting an OCSP responder.</p> <p>The URL must conform to the definition of an HTTP url: http_URL = "http:" "//" host [":" port] [abs_path ["?" query]]</p> <p>where host can be an IPv4 address, an IPv6 address, or a domain name.</p>
<i>ocspURLPriority</i>	<p>Specifies the order of precedence for contacting OCSP responder locations if <i>ocspURL</i> is non-NULL and <i>ocspEnable</i> is TRUE.</p> <p>TRUE The URL specified in <i>ocspURL</i> has priority over the certificate's AIA extension.</p> <p>FALSE The URL specified in the AIA extension has priority over the <i>ocspURL</i> value.</p>
<i>ocspUseGetMethod</i>	<p>Specifies the OCSP HTTP method to be used:</p> <p>TRUE HTTP GET method is used.</p> <p>FALSE HTTP POST method is used.</p>

gsk_create_self_signed_certificate()

gsk_create_self_signed_certificate()

Creates a self-signed certificate.

Format

```
#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.

[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_ICSF_CLEAR_KEY_SUPPORT_NOT_AVAILABLE]

Clear key support not available due to ICSF key policy.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_FIPS_KEY_PAIR_CONSISTENCY]

FIPS mode key generation failed pair-wise consistency check.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[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 5280: *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

gsk_create_self_signed_certificate()

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}
- x509_alg_dsaWithSha224**
Digital Signature Standard with SHA-224 digest - {2.16.840.1.101.3.4.3.1}
- x509_alg_dsaWithSha256**
Digital Signature Standard with SHA-256 digest - {2.16.840.1.101.3.4.3.2}
- x509_alg_ecdsaWithSha1**
Elliptic Curve Digital Signature Algorithm with SHA-1 digest - {1.2.840.10045.4.1}
- x509_alg_ecdsaWithSha224**
Elliptic Curve Digital Signature Algorithm with SHA-224 digest - {1.2.840.10045.4.3.1}
- x509_alg_ecdsaWithSha256**
Elliptic Curve Digital Signature Algorithm with SHA-256 digest - {1.2.840.10045.4.3.2}
- x509_alg_ecdsaWithSha384**
Elliptic Curve Digital Signature Algorithm with SHA-384 digest - {1.2.840.10045.4.3.3}
- x509_alg_ecdsaWithSha512**
Elliptic Curve Digital Signature Algorithm with SHA-512 digest - {1.2.840.10045.4.3.4}

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

2048 bits. Key sizes of between 512 and 1024 bits are rounded up to a multiple of 64. A key size of 2048 must be explicitly specified as such. A key size of 1024 or less should specify signature algorithm `x509_alg_dsaWithSha1`, while a key size of 2048 bits should specify either `x509_alg_dsaWithSha224` or `x509_alg_dsaWithSha256` as the signature algorithm.

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 either 1024 bits or 2048 bits. A key size of 1024 bits should specify signature algorithm `x509_alg_dsaWithSha1`, while a key size of 2048 bits should specify either `x509_alg_dsaWithSha224` or `x509_alg_dsaWithSha256` as the signature algorithm.

For an ECC key, the key size will determine the default namedCurve that will be used for the public/private key pair, as specified in Table 3 on page 15. In FIPS mode, only NIST recommended curves are supported. To specify a specific supported elliptic curve, use **gsk_construct_self_signed_certificate()** to create a self-signed certificate.

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 before 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. However, 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()`

`gsk_create_signed_certificate()`

Creates a signed certificate.

This function is deprecated. Use `gsk_create_signed_certificate_record()` instead.

Format

```
#include <gskcms.h>

gsk_status gsk_create_signed_certificate (
    gsk_handle          db_handle,
    const char *       label,
    int                 num_days,
    gsk_boolean        ca_certificate,
    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.

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_EC_PARAMS]**
Elliptic Curve parameters are not valid.
- [CMSERR_BAD_ENCODING]**
The certificate request stream is not valid.
- [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_ECURVE_NOT_FIPS_APPROVED]**
Elliptic Curve not supported in FIPS mode.
- [CMSERR_ECURVE_NOT_SUPPORTED]**
Elliptic Curve is not supported.
- [CMSERR_EXPIRED]**
The signer certificate is expired.
- [CMSERR_ICSF_FIPS_DISABLED]**
ICSF PKCS #11 services are disabled.
- [CMSERR_ICSF_NOT_AVAILABLE]**
ICSF services are not available.
- [CMSERR_ICSF_NOT_FIPS]**
ICSF PKCS #11 not operating in FIPS mode.
- [CMSERR_ICSF_SERVICE_FAILURE]**
ICSF callable service returned an error.
- [CMSERR_INCORRECT_DBTYPE]**
The database type does not support certificates.
- [CMSERR_INCORRECT_KEY_TYPE]**
Incorrect key algorithm
- [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_MISMATCH]**
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.

gsk_create_signed_certificate()

Usage

The **gsk_create_signed_certificate()** routine will generate an X.509 certificate as described in RFC 5280: *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.

If the certificate request contains an ECC key, the signing certificate cannot contain a DSA key.

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 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.
- An ECC key can be used for authentication, digital signature and key agreement.

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.

gsk_create_signed_certificate_record()

Creates a signed certificate.

Format

```
#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.

gsk_create_signed_certificate_record()

- [CMSERR_BACKUP_EXISTS]**
The backup file already exists.
- [CMSERR_BAD_EC_PARAMS]**
Elliptic Curve parameters are not valid.
- [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_ECURVE_NOT_FIPS_APPROVED]**
Elliptic Curve not supported in FIPS mode.
- [CMSERR_ECURVE_NOT_SUPPORTED]**
Elliptic Curve is not supported.
- [CMSERR_ICSF_FIPS_DISABLED]**
ICSF PKCS #11 services are disabled.
- [CMSERR_ICSF_NOT_AVAILABLE]**
ICSF services are not available.
- [CMSERR_ICSF_NOT_FIPS]**
ICSF PKCS #11 not operating in FIPS mode.
- [CMSERR_ICSF_SERVICE_FAILURE]**
ICSF callable service returned an error.
- [CMSERR_INCORRECT_DBTYPE]**
The database type does not support certificates.
- [CMSERR_INCORRECT_KEY_TYPE]**
Incorrect key algorithm.
- [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 signers 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 5280: *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.

If the certificate request contains an ECC key, the signing certificate cannot contain a DSA key.

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_dsaWithSha224

Digital Signature Standard with SHA-224 digest - {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256

Digital Signature Standard with SHA-256 digest - {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1

Elliptic Curve Digital Signature Algorithm with SHA-1 digest - {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224

Elliptic Curve Digital Signature Algorithm with SHA-224 digest - {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256

Elliptic Curve Digital Signature Algorithm with SHA-256 digest - {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384

Elliptic Curve Digital Signature Algorithm with SHA-384 digest - {1.2.840.10045.4.3.3}

`gsk_create_signed_certificate_record()`

`x509_alg_ecdsaWithSha512`

Elliptic Curve Digital Signature Algorithm with SHA-512 digest –
{1.2.840.10045.4.3.4}

When executing in non-FIPS mode, an RSA key size must be between 512 and 4096 bits. A DSA key size must be between 512 and 2048 bits. An ECC key can be a NIST recommended named curve with a key between 192 and 521 or a Brainpool curve with a key between 160 and 512.

When executing in FIPS mode:

- Signature algorithms `x509_alg_md2WithRSAEncryption` and `x509_alg_md5WithRsaEncryption` are not supported.
- An RSA key size must be between 1024 and 4096 bits.
- A DSA key size must be either 1024 bits or 2048 bits.
- An NIST ECC key must be a NIST recommended named curve with a key size between 192 and 521.

For more information about FIPS supported algorithms and key sizes, see “Algorithms and key sizes” on page 19.

A DSA key size of 1024 or less should specify signature algorithm `x509_alg_dsaWithSha1`, while a key size of 2048 bits should specify either `x509_alg_dsaWithSha224` or `x509_alg_dsaWithSha256` as the signature algorithm.

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 as follows:

- An RSA key can be used for authentication, digital signature, and data encryption.
- A DSA key can be used for authentication and digital signature.
- A Diffie-Hellman key can be used for key agreement.
- An ECC key can be used for authentication, digital signature and key agreement.

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

gsk_create_signed_certificate_record()

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.

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.

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 key algorithm or the signature algorithm is not valid.

[CMSERR_BACKUP_EXISTS]

The backup file already exists.

[CMSERR_BAD_EC_PARAMS]

Elliptic Curve parameters are not valid.

[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_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_EXPIRED]

The signer certificate is expired.

[CMSERR_ICSF_CLEAR_KEY_SUPPORT_NOT_AVAILABLE]

Clear key support not available due to ICSF key policy.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]

The database type does not support certificates.

[CMSERR_INCORRECT_KEY_TYPE]

Incorrect key algorithm

gsk_create_signed_certificate_set()

[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_signed_certificate_set()** routine will generate an X.509 certificate as described in RFC 5280: *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.

- If the specified certificate key is a Diffie-Hellman key, the signing certificate must contain either an RSA or a DSA key.
- If the specified certificate key is an ECC key, the signing certificate cannot contain a DSA key.

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
- An ECC key can be used for authentication, digital signature, and 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_dhPublicNumber

Diffie-Hellman (DH) - {1.2.840.10046.2.1}

x509_alg_ecPublicKey

Elliptic Curve Public Key (ECC) - {1.2.840.10045.2.1}

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 sizes of between 512 and 1024 bits when in non-FIPS mode are rounded up to a multiple of 64.
- Key sizes less than 1024 bits can only be generated in non-FIPS mode and are generated according to FIPS 186-2.
- Key sizes of 1024 or 2048 bits are generated according to FIPS 186-3 in both FIPS mode and non-FIPS mode. These are the only valid key sizes in FIPS mode.
- A key size of 1024 bits or less will use SHA1 digest, while a key size of 2048 bits will use SHA256 digest.
- Key parameters encoded as an ASN.1 sequence consisting of the prime p , the prime divisor q , and the generator g . For 1024-bit and 2048-bit keys, see FIPS 186-3: *Digital Signature Standard (DSS)* for more information about the key parameters. For smaller key sizes, see FIPS 186-2: *Digital Signature Standard (DSS)*. Note that key parameters that contain a p of 2048 bits and a q of 160 bits do not conform to FIPS 186-3 and are not supported. The **gsk_generate_key_parameters()** routine can be used to generate the key parameters.

DH keys

- Can be used only for end user certificates
- Can only be signed using a certificate containing either an RSA or a DSA key
- 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 . See RFC 2631: *Diffie-Hellman Key Agreement Method* for more information about the key parameters for non-FIPS mode, and see *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for FIPS mode. The **gsk_generate_key_parameters()** routine can be used to generate the key parameters.

ECC keys

gsk_create_signed_certificate_set()

- Can be used for both CA certificates and end user certificates.
- The ECC named curve used to generate the ECC key pair can be specified using either the *key_parameters* buffer or the *key_size* parameter. If the *key_parameters* buffer is supplied the *key_size* parameter will be ignored.
- The *key_parameters* buffer must contain ASN.1 encoded EC domain parameters, or be NULL.
- If the *key_parameters* buffer is not supplied, the *key_size* parameter will be rounded up to the nearest supported key size and the default EC named curve for that key size will be used, as specified in Table 3 on page 15.
- In FIPS mode only NIST recommended curves are supported.

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 *
    x509_extensions *  revoked_certificates,
    gsk_buffer *       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.

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_EC_PARAMS]

Elliptic Curve parameters are not valid.

gsk_create_signed_crl()

[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_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_EXPIRED]

The signer certificate is expired.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[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 5280: *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.

gsk_create_signed_crl()

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()`

`gsk_create_signed_crl_record()`

Creates a signed certificate revocation list.

Format

```
#include <gskcms.h>

gsk_status gsk_create_signed_crl_record (
    gsk_handle          db_handle,
    const char *       label,
    x509_algorithm_type signature_algorithm,
    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.

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.

[CMSERR_BAD_EC_PARAMS]

Elliptic Curve parameters are not valid.

[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 request signature is not correct.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_EXPIRED]

The signer certificate is expired.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[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_record()** routine will generate an X.509 certificate revocation list (CRL) as described in RFC 5280: *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:

gsk_create_signed_crl_record()

- 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_dsaWithSha224**
Digital Signature Standard with SHA-224 digest - {2.16.840.1.101.3.4.3.1}
- x509_alg_dsaWithSha256**
Digital Signature Standard with SHA-256 digest - {2.16.840.1.101.3.4.3.2}
- x509_alg_ecdsaWithSha1**
Elliptic Curve Digital Signature Algorithm with SHA-1 digest –
{1.2.840.10045.4.1}
- x509_alg_ecdsaWithSha224**
Elliptic Curve Digital Signature Algorithm with SHA-224 digest –
{1.2.840.10045.4.3.1}
- x509_alg_ecdsaWithSha256**
Elliptic Curve Digital Signature Algorithm with SHA-256 digest –
{1.2.840.10045.4.3.2}
- x509_alg_ecdsaWithSha384**
Elliptic Curve Digital Signature Algorithm with SHA-384 digest –
{1.2.840.10045.4.3.3}
- x509_alg_ecdsaWithSha512**
Elliptic Curve Digital Signature Algorithm with SHA-512 digest –
{1.2.840.10045.4.3.4}

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.

gsk_create_signed_crl_record()

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

```
#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 5280: *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 re-create 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()

gsk_decode_certificate_extension()

Decodes an X.509 certificate extension.

Format

```
#include <gskcms.h>
```

```
gsk_status 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
- CertificateIssuer
- CertificatePolicies
- CrlDistributionPoints
- CrlNumber

- CrlReasonCode
- DeltaCrlIndicator
- ExtKeyUsage
- FreshestCRL
- HoldInstructionCode
- HostIDMapping (z/OS specific extension 1.3.18.0.2.18.1)
- InhibitAnyPolicy
- InvalidityDate
- IssuerAltName
- IssuingDistributionPoint
- KeyUsage
- NameConstraints
- OCSPNoCheck
- PolicyConstraints
- PolicyMappings
- PrivateKeyUsagePeriod (not supported in RFC 5280)
- 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

See RFC 5280: *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()`

`gsk_decode_certification_request()`

Decodes a PKCS #10 certification request.

Format

```
#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, Version 1.7: *Certification Request*. 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 re-create 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

```
#include <gskcms.h>

gsk_status gsk_decode_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 5280: *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 time values that return in the *next_update* field of the *x509_tbs_crl* structure within the *x509_crl* structure are *tm_year*, *tm_mon*, *tm_mday*, *tm_hour*, *tm_min*, and *tm_sec*. The **gsk_encode_signature()** routine can be used to re-create 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 DERencoded or PKCS #7encoded data stream.

Format

```
#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 certificates 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

```
#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 certificates chain are discarded). The root certificate for the chain is the final entry in the array.

gsk_decode_import_key()

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_issuer_and_serial_number()

Decodes a PKCS #7 *IssuerAndSerialNumber*.

Format

```
#include <gskcms.h>

gsk_status gsk_decode_issuer_and_serial_number (
    gsk_buffer *          stream,
    pkcs_issuer_serial * issuer_serial)
```

Parameters

stream

Specifies an ASN.1 DER-encoded stream containing a PKCS #7 *IssuerAndSerialNumber*.

issuer_serial

Returns the decoded issuer name and serial number. The application should call the **gsk_free_issuer_and_serial_number()** 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:

[ASN_NO_MORE_DATA]

Input data is incomplete.

[ASN_SELECTION_OUT_OF_RANGE]

Selection is not within the valid range.

[CMSERR_NO_MEMORY]

Insufficient storage is available.

Usage

The **gsk_decode_issuer_and_serial_number()** routine decodes a PKCS #7 (Cryptographic Message Syntax) ASN.1 DER-encoded *IssuerAndSerialNumber* and returns the issuer name and serial number.

gsk_decode_name()

gsk_decode_name()

Decodes an X.509 name.

Format

```
#include <gskcms.h>
```

```
gsk_status gsk_decode_name (
                                gsk_buffer *      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 5280: *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

```
#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, Version 1.2: *Private Key Information Syntax*.

gsk_decode_public_key()

gsk_decode_public_key()

Decodes a public key.

Format

```
#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 5280: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*.

gsk_decode_signer_identifier()

Decodes a PKCS #7 signer identifier.

Format

```
#include <gskcms.h>
```

```
gsk_status gsk_decode_signer_identifier (
    int                signer_info_version,
    gsk_buffer *       encoded_signer_identifier,
    pkcs_signer_id *   decoded_signer_identifier)
```

Parameters

signer_info_version

Specifies the PKCS #7 Cryptographic Message Syntax version that was used to create the signer identifier as described in RFC 3852:

- Specify 1 to use the certificate's *IssuerAndSerialNumber*.
- Specify 3 to use the certificate's *SubjectKeyIdentifier*.

encoded_signer_identifier

Specifies ASN.1 DER-encoded stream containing a PKCS #7 signer identifier.

decoded_signer_identifier

Returns the decoded *pkcs_signer_id* containing either an *IssuerAndSerialNumber* or *SubjectKeyIdentifier*.

The application should call the **gsk_free_signer_identifier()** 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:

[ASN_NO_MORE_DATA]

Input data is incomplete.

[CMSERR_PKCS7_CMSVERSION_NOT_SUPPORTED]

PKCS #7 CMS version is not supported.

[CMSERR_NO_MEMORY]

Insufficient storage is available.

Usage

The **gsk_decode_signer_identifier()** routine reads a PKCS #7 (Cryptographic Message Syntax) ASN.1 DER-encoded signer identifier and returns a *pkcs_signer_id* containing either the *IssuerAndSerialNumber* or the *SubjectKeyIdentifier*.

`gsk_delete_record()`

Deletes a record from a key or request database.

Format

```
#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

```
#include <gskcms.h>

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.

gsk_dn_to_name()

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 run time. An error is returned if the DN contains an unrecognized attribute name.

Table 14. DN attribute names

	Name
C	Country
CN	Common name
DC	Domain component
DNQUALIFIER	Distinguished name qualifier
E	E-mail address
EMAIL	E-mail address (preferred)
EMAILADDRESS	E-mail address
GENERATIONQUALIFIER	Generation qualifier
GIVENNAME	Given name
INITIALS	Initials
L	Locality
MAIL	RFC 822 style address
NAME	Name
O	Organization name
OU	Organizational unit name
PC	Postal code
S	State or province
SERIALNUMBER	Serial number
SN	Surname
SP	State or province
ST	State or province (preferred)
STREET	Street
T	Title

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 run time supports these ASN.1 string types: PRINTABLE, VISIBLE, TELETEX, IA5, UTF8, BMP, and UCS.

```
2.5.4.3=#130E526F6E616C6420486F666666D616E,2.5.4.11=#1308456E6469636F7474,  
2.5.4.10=#130349424D,2.5.4.6=13025553
```

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.

Unicode Letter Description	10646 code	UTF-8	Quoted
=====	=====	=====	=====
LATIN CAPITAL LETTER L	U0000004C	0x4C	L
LATIN SMALL LETTER U	U00000075	0x75	u
LATIN SMALL LETTER C WITH CARON	U0000010D	0xC48D	\C4\8D
LATIN SMALL LETTER I	U00000069	0x69	i
LATIN SMALL LETTER C WITH ACUTE	U00000107	0xC487	\C4\87

SN=Lu\C4\8Di\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:

CN=L. Eagle,OU=Jones\, Dale and Mian,O=IBM,C=US

gsk_encode_base64()

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

```
#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

gsk_encode_certificate_extension()

- CrlNumber
- CrlReasonCode
- DeltaCrlIndicator
- ExtKeyUsage
- FreshestCRL
- HoldInstructionCode
- HostIDMapping (z/OS specific extension 1.3.18.0.2.18.1)
- InhibitAnyPolicy
- InvalidityDate
- IssuerAltName
- IssuingDistributionPoint
- KeyUsage
- NameConstraints
- OCSPNoCheck
- PolicyConstraints
- PolicyMappings
- PrivateKeyUsagePeriod (not supported in RFC 5280)
- SubjectAltName
- SubjectDirectoryAttributes
- SubjectInfoAccess
- SubjectKeyIdentifier

These general name types are supported:

- DirectoryName
- DnsName
- IpAddress
- RegisteredId
- Rfc822Name
- UniformResourceIdentifier

See RFC 5280: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile* for more information about the various certificate extensions.

gsk_encode_ec_parameters()

Encodes the EC domain parameters for an ECC key

Format

```
#include <gskcms.h>

gsk_status gsk_encode_ec_parameters (
    int          arg_count,
    x509_ecurve_type ec_curve,
    gsk_buffer * key_params,
    ...)
```

Parameters

arg_count

Specifies the number of parameters following the *arg_count* parameter. Currently, *arg_count* must be set to 2.

ec_curve

Specifies the EC named curve

key_params

Returns the ASN.1 stream for the EC domain parameters. 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:

[ASN_NO_MEMORY]

Insufficient memory is available.

[CMSERR_BAD_ARG_COUNT]

Variable argument count is not valid.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported

Usage

The **gsk_encode_ec_parameters()** routine will encode the EC domain parameters of an elliptic curve as an ASN.1 stream. The EC domain parameters will be encoded as described in SEC1 (Elliptic Curve Cryptography).

`gsk_encode_export_certificate()`

Encodes an X.509 certificate into a DER or PKCS #7 data stream.

Format

```
#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

gsk_encode_export_certificate()

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. The certificate chain for the subject certificate is supplied from 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_key()

Encodes an X.509 certificate and its private key into a PKCS #12 data stream.

Format

```
#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. The private key must not be stored in ICSF's
| PKDS.

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
40bit RC2 with SHA-1 digest.

x509_alg_pbeWithSha1And128BitRc2Cbc
128-bit RC2 with SHA-1 digest.

x509_alg_pbeWithSha1And40BitRc4
40bit 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.

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 is prompted to enter the password if NULL is specified for this parameter. If the key that is being encoded for export is a secure private key in the TKDS, the maximum password length is 63 bytes.

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_ALG_NOT_SUPPORTED]

The signature algorithm is not valid.

[CMSERR_CRYPTO_HARDWARE_NOT_AVAILABLE]

Cryptographic hardware does not support service or algorithm.

[CMSERR_FMT_NOT_SUPPORTED]

An unsupported export file format is specified.

[CMSERR_ICSF_FIPS_BAD_ALG_OR_KEY_SIZE]

The algorithm or key size is not supported by ICSF in FIPS mode.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_FIPS]

ICSF is not operating in FIPS mode.

[CMSERR_INCORRECT_DBTYPE]

The database type does not support certificates.

[CMSERR_INCORRECT_KEY_ATTRIBUTE]

Parameter contents or key attribute value is incorrect.

[CMSERR_KEY_CANNOT_BE_EXTRACTED]

PKCS #11 key cannot be extracted.

[CMSERR_NO_MEMORY]

Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]

The signer certificate does not have a private key.

[CMSERR_PW_INCORRECT]

The password is not correct.

gsk_encode_export_key()

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 from 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*.

If the certificate's private key is stored as a secure TKDS private key label:

- Only formats **gskdb_export_pkcs12v3_binary** and **gskdb_export_pkcs12v3_base64**, along with algorithm **x509_alg_pbeWithSha1And3DesCbc**, are supported.
- When the private key was created in the TKDS, it was created with the extractable attribute.
- When using this API, you must have the correct access to the CRYPTOZ class. See Chapter 3, "Using cryptographic features with System SSL," on page 11 for more information.

gsk_encode_export_request()

Encodes a certification renewal request as described in PKCS #10, Version 1.7: *Certification Request*.

Format

```
#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 DERencoded.

gskdb_export_der_base64

Base64 ASN.1 DERencoded.

stream

Returns 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. This is a possible error:

[CMSERR_NO_MEMORY]

Insufficient storage is available.

Usage

The **gsk_encode_export_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 header and footer lines.

`gsk_encode_issuer_and_serial_number()`

`gsk_encode_issuer_and_serial_number()`

Encodes a PKCS #7 *IssuerAndSerialNumber*.

Format

```
#include <gskcms.h>

gsk_status gsk_encode_issuer_and_serial_number (
    pkcs_certificate *    certificate,
    gsk_buffer *         stream)
```

Parameters

certificate

Specifies an x.509 certificate containing the issuer name and serial number to be encoded.

stream

Returns the ASN.1 DER-encoded stream containing the *IssuerAndSerialNumber* sequence. 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:

[ASN_SELECTION_OUT_OF_RANGE]

Selection is not within the valid range.

[CMSERR_CERTIFICATE_NOT_PROVIDED]

Input certificate is missing.

[CMSERR_NO_MEMORY]

Insufficient storage is available.

Usage

The `gsk_encode_issuer_and_serial_number()` routine returns a PKCS #7 (Cryptographic Message Syntax) ASN.1 DER-encoded *IssuerAndSerialNumber* from the input certificate. The *IssuerAndSerialNumber* created by calling `gsk_encode_issuer_and_serial_number()` can be decoded using `gsk_decode_issuer_and_serial_number()`.

gsk_encode_name()

Encodes an X.509 name.

Format

```
#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 5280: *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 attempts 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()`

`gsk_encode_private_key()`

Encode a private key.

Format

```
#include <gskcms.h>
```

```
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, Version 1.2: *Private Key Information Syntax*. 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 5280: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*.

gsk_encode_signature()

gsk_encode_signature()

Encodes an ASN.1 stream and the accompanying signature.

Format

```
#include <gskcms.h>

gsk_status gsk_encode_signature (
    gsk_buffer *                unsigned_stream,
    x509_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_encode_signer_identifier()

Encodes a PKCS #7 *SignerIdentifier* from a signer certificate.

Format

```
#include <gskcms.h>

gsk_status gsk_encode_signer_identifier (
    int *                cms_version,
    pkcs_certificate *   signer_certificate,
    gsk_buffer *         stream)
```

Parameters

cms_version

Specifies the PKCS #7 Cryptographic Message Syntax version that was used to create the signer identifier as described in RFC 3852:

- Specify 1 to use the certificate's *IssuerAndSerialNumber*.
- Specify 3 to use the certificate's *SubjectKeyIdentifier*.

signer_certificate

Specifies an x.509 certificate that is a signer certificate.

stream

Returns the ASN.1 DER-encoded stream. 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:

[ASN_SELECTION_OUT_OF_RANGE]

Selection is not within the valid range.

[CMSERR_CERTIFICATE_NOT_PROVIDED]

Input certificate not provided.

[CMSERR_PKCS7_CMSVERSION_NOT_SUPPORTED]

PKCS #7 CMS version is not supported.

[CMSERR_NO_MEMORY]

Insufficient storage is available.

Usage

The **gsk_encode_signer_identifier()** routine creates a PKCS #7 (Cryptographic Message Syntax) signer identifier according to RFC 3852 and returns the ASN.1 DER-encoded signer identifier. The signer identifier created by calling **gsk_encode_signer_identifier()** can be decoded using **gsk_decode_signer_identifier()**.

`gsk_export_certificate()`

Exports a certificate.

Format

```
#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.

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

```
#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 header and footer lines.

gsk_export_key()

Exports a certificate and the associated private key.

Format

```
#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 may be stored in the SAF database or ICSF's TKDS as either a clear or secure key. It cannot be stored in ICSF's PKDS. For a PKCS #11 token, the private key may be stored in ICSF's TKDS as either a clear or secure key. When stored in ICSF's TKDS as a secure key, the key must be extractable.

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_pb1WithSha1And128 BitRc2Cbc

128bit 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:

x509_alg_pbeWithSha1And40BitRc2Cbc

40-bit RC2 with SHA-1 digest

x509_alg_pbeWithSha1And128 BitRc2Cbc

128bit 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 export format must be PKCS #12 Version 3 and the encryption algorithm must be **x509_alg_pbeWithSha1And3DesCbc**.

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 is prompted to enter the password if NULL is specified for this parameter. If the key that is being exported is a secure private key in the TKDS, the maximum password length is 63 bytes.

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_CRYPTO_HARDWARE_NOT_AVAILABLE]

Cryptographic hardware does not support service or algorithm.

[CMSERR_FMT_NOT_SUPPORTED]

An unsupported export file format is specified.

[CMSERR_ICSF_FIPS_BAD_ALG_OR_KEY_SIZE]

The algorithm or key size is not supported by ICSF in FIPS mode.

gsk_export_key()

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_FIPS]

ICSF is not operating in FIPS mode.

[CMSERR_INCORRECT_DBTYPE]

The database type does not support certificates.

[CMSERR_INCORRECT_KEY_ATTRIBUTE]

Parameter contents or key attribute value is incorrect.

[CMSERR_KEY_CANNOT_BE_EXTRACTED]

PKCS #11 key cannot be extracted.

[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.

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.

If the certificate's private key is stored as a secure TKDS private key label:

- Only formats **gskdb_export_pkcs12v3_binary** and **gskdb_export_pkcs12v3_base64**, along with algorithm **x509_alg_pbeWithSha1And3DesCbc**, are supported.
- When the private key was created in the TKDS, it must be created with the extractable attribute.
- When using this API, you must have the correct access to the CRYPTOZ class. See Chapter 3, "Using cryptographic features with System SSL," on page 11 for more information.

gsk_factor_private_key()

Factorizes a private key into its component values.

Format

```
#include <gskcms.h>

gsk_status gsk_factor_private_key(
    pkcs_private_key_info * private_key,
    gsk_private_key *      private_key_factors)
```

Parameters

private_key
Specifies the private key.

private_key_factors
Returns the private key components.

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.

[CMSERR_ALG_NOT_SUPPORTED]

Cryptographic algorithm is not supported.

[CMSERR_PRIVATE_KEY_INFO_NOT_SUPPLIED]

Private key information not supplied.

[CMSERR_PRIVATE_KEY_NOT_SUPPLIED]

Private key structure not supplied.

[CMSERR_STRUCTURE_TOO_SMALL]

Size specified for supplied structure is too small.

Usage

The `gsk_factor_private_key()` function deconstructs the private key into its private key components, formatted for use with ICSF PKCS #11 tokens.

Before calling the function, the application must initialize the size field in *private_key_factors* to the size of the `gsk_private_key` structure. It must also prime *private_key* with the appropriate private key to be factorized before calling the routine.

The routine will return the factorized components of the private key in *private_key_factors*. The `x509_algorithm_identifier` is set with the appropriate value for the private key type when returned.

gsk_factor_private_key_rsa()

Factorizes an RSA private key into its component values.

Note: This function is deprecated. Use `gsk_factor_private_key()` instead.

Format

```
#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 $q^{-1} \bmod p$.

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()

Factorizes a public key into its component values.

Format

```
#include <gskcms.h>

gsk_status gsk_factor_public_key(
    x509_public_key_info *   public_key,
    gsk_public_key *        public_key_factors)
```

Parameters

public_key

Specifies the public key.

public_key_factors

Returns the public key components.

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.

[CMSERR_ALG_NOT_SUPPORTED]

Cryptographic algorithm not supported.

[CMSERR_PUBLIC_KEY_INFO_NOT_SUPPLIED]

Public key information not supplied.

[CMSERR_PUBLIC_KEY_NOT_SUPPLIED]

Public key structure not supplied.

[CMSERR_STRUCTURE_TOO_SMALL]

Size specified for supplied structure is too small.

Usage

The `gsk_factor_public_key()` function deconstructs the public key into its public key components, formatted for use with ICSF PKCS #11 tokens.

Before calling the function, the application must initialize the size field in *public_key_factors* to the size of the `gsk_public_key` structure. It must also prime *public_key* with the appropriate public key to be factorized before calling the routine.

The routine will return the factorized component of the public key in *public_key_factors*. The `x509_algorithm_identifier` is set with the appropriate value for the public key type when returned.

`gsk_factor_public_key_rsa()`

`gsk_factor_public_key_rsa()`

Factorizes an RSA public key into its component values.

Note: This function is deprecated. Use `gsk_factor_public_key()` instead.

Format

```
#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

```
gsk_status gsk_fips_state_query(  
                                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 because of 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_KATPW_ICSF_FAILED]

The power-on known answer tests failed. Either ICSF was not available or FIPS mode was disabled. FIPS mode cannot be set.

Usage

The **gsk_fips_state_set()** routine sets the enumerated value for the System SSL FIPS mode state.

The FIPS mode setting applies to the entire process. Once set, then all threads of the same process execute in FIPS mode. If any thread switches to non-FIPS mode, then all threads in the same process execute in non-FIPS mode.

In order to set FIPS mode, this function must be executed before all other System SSL API functions except for **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 values 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()

gsk_free_attributes_signers()

Releases storage allocated for `gsk_attributes_signers` structure.

Format

```
#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

```
#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()

gsk_free_certificate()

Releases storage allocated for an X.509 certificate.

Format

```
#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

```
#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()`

`gsk_free_certificate_extension()`

Releases storage allocated for an X.509 certificate extension.

Format

```
#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

```
#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()

gsk_free_content_info()

Releases storage allocated for PKCS #7 content information.

Format

```
#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

```
#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()

gsk_free_crls()

Releases storage allocated for an array of X.509 certificate revocation lists.

Format

```
#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

```
#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_issuer_and_serial_number()`

`gsk_free_issuer_and_serial_number()`

Releases storage allocated for PKCS #7 issuer and serial number information.

Format

```
#include <gskcms.h>
```

```
void gsk_free_issuer_and_serial_number (  
    pkcs_issuer_serial *    issuer_serial)
```

Parameters

issuer_serial

Specifies the issuer and serial number structure *pkcs_issuer_serial* to be released.

Usage

The `gsk_free_issuer_and_serial_number()` routine is used to release storage allocated for PKCS #7 issuer and serial information.

gsk_free_name()

Releases storage allocated for an X.509 name.

Format

```
#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_oid()

gsk_free_oid()

Releases storage allocated for OID information.

Format

```
#include <gskcms.h>
```

```
void gsk_free_oid ( gsk_oid * oid)
```

Parameters

oid

Specifies the OID to be released.

Usage

The **gsk_free_oid()** routine is used to release storage allocated for OID.

gsk_free_private_key()

Releases storage allocated for private key information.

Format

```
#include <gskcms.h>
```

```
gsk_status gsk_free_private_key(  
    gsk_private_key *      private_key_factors)
```

Parameters

private_key_factors

Specifies the private key components. The `gsk_private_key` structure will be initialized to zero upon completion.

Usage

The `gsk_free_private_key()` routine is used to release storage allocated for private key component information.

gsk_free_private_key_info()

gsk_free_private_key_info()

Releases storage allocated for private key information.

Format

```
#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()

Releases storage allocated for public key information.

Format

```
#include <gskcms.h>
```

```
gsk_status gsk_free_public_key(  
    gsk_public_key *      public_key_factors)
```

Parameters

public_key_factors

Specifies the public key components. The **gsk_free_public_key** structure will be initialized to zero upon completion.

Usage

The **gsk_free_public_key()** routine is used to release storage allocated for public key component information.

`gsk_free_public_key_info()`

`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

```
#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()

gsk_free_records()

Releases storage allocated for an array of database records.

Format

```
#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_revocation_source()

Frees the revocation source initialized and allocated by `gsk_create_revocation_source()`.

Format

```
#include <gskcms.h>
```

```
void gsk_free_revocation_source (  
                                gsk_handle *   revocation_handle)
```

Parameters

revocation_handle

Specifies a revocation source handle created by the `gsk_create_revocation_source()` routine.

Usage

The `gsk_free_revocation_source()` routine is used to release storage allocated for a revocation source handle.

`gsk_free_signer_identifier()`

`gsk_free_signer_identifier()`

Releases storage allocated for PKCS #7 signer identifier information.

Format

```
#include <gskcms.h>
```

```
void gsk_free_signer_identifier (
                                int *          cms_version,
                                pkcs_signer_id * signer_id)
```

Parameters

cms_version

Specifies the PKCS #7 Cryptographic Message Syntax version that was used to create the signer identifier as described in RFC 3852:

- Specify 1 to use the certificate's *IssuerAndSerialNumber*.
- Specify 3 to use the certificate's *SubjectKeyIdentifier*.

signer_id

Specifies the PKCS #7 signer identifier to be released.

Usage

The `gsk_free_signer_identifier()` routine is used to release storage allocated for PKCS #7 signer identifier.

gsk_free_string()

Releases storage allocated for a string.

Format

```
#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()

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

```
#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 using ICSF when executing in FIPS mode, and as recommended by PKCS #3 (Diffie-Hellman Key Agreement Standard) and RFC 2631: *Diffie-Hellman Key Agreement Method* when in non-FIPS mode. 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 5280: *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.

gsk_generate_key_agreement_pair()

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

```
#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_EC_PARAMS]

Elliptic Curve parameters are not valid.

[CMSERR_BAD_KEY_SIZE]

The key size is not valid.

gsk_generate_key_pair()

[CMSERR_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_FIPS_KEY_PAIR_CONSISTENCY]

FIPS mode key generation failed pair-wise consistency check.

[CMSERR_ICSF_CLEAR_KEY_SUPPORT_NOT_AVAILABLE]

Clear key support not available due to ICSF key policy.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[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 5280: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*.

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 can be between 512 and 1024 bits, which will be rounded up to a multiple of 64 bits, or precisely 2048 bits. Key sizes less than 1024 bits can only be generated in non-FIPS mode and are generated according to FIPS 186-2: *Digital Signature Standard (DSS)*. Keys sizes 1024 and 2048 are generated according to FIPS 186-3: *Digital Signature Standard (DSS)*. The key parameters are the prime p, the prime divisor q, and the generator g. The requested key size must be the same as the size of the prime p. Note that key parameters that contain a p of 2048 bits and a q of 160 bits do not conform to FIPS 186-3 and are not supported. 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.

In non-FIPS mode, 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) including 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.

- **x509_alg_ecPublicKey – ECDSA and ECDH Public Key - {1.2.840.10045.2.1}**

The EC named curve used to generate the ECC key pair can be specified using either the *key_params* buffer or the *key_size* parameter. If the *key_params* buffer is supplied, the *key_size* parameter will be ignored. The *key_params* buffer must contain ASN.1 encoded EC domain parameters, or be NULL. If the *key_params* buffer is NULL, the *key_size* parameter will be rounded up to the nearest supported key size and the default EC named curve for that key size will be used, as specified in Table 3 on page 15. In FIPS mode, only NIST recommended curves are supported.

| For more information about FIPS supported algorithms and key sizes, see
| “Algorithms and key sizes” on page 19 and FIPS 140-2.

`gsk_generate_key_parameters()`

Generates ASN.1 encoded key parameters.

Format

```
#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 can be between 512 and 1024 bits, which will be rounded up to a multiple of 64 bits, or precisely 2048 bits. Key sizes less than 1024 bits can only be generated in non-FIPS mode and are generated according to FIPS 186-2. Keys sizes 1024 and 2048 are generated according to FIPS 186-3. The generated ASN.1 sequence will consist of the prime *P*, the subprime *Q*, and the base *G*. For 2048-bit key size, the size of the subprime *Q* will be 256. See FIPS 186-3: *Digital Signature Standard (DSS)* for more information about the generation of the key parameters for 1024-bit and greater key sizes. See FIPS 186-2: *Digital Signature Standard (DSS)* for smaller key sizes.

- **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. In non-FIPS mode, the generated ASN.1 sequence will consist of the prime P, the base G, the subprime Q, and the subgroup factor J. In FIPS mode, the generated ASN.1 sequence will consist of the prime P and the base G. See RFC 2631: *Diffie-Hellman Key Agreement Method* for more information about the generation of the key parameters, and RFC 5280: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile* for more information about 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).

- **x509_alg_ecPublicKey – ECDSA and ECDH Public Key - {1.2.840.10045.2.1}**

The key size must be between 0 and 521 bits. The key size value will be rounded up to the nearest supported key size, and the default EC named curve for that key size will be used, as specified in Table 3 on page 15. In FIPS mode, only NIST recommended curves are supported.

| For more information about FIPS supported algorithms and key sizes, see
| “Algorithms and key sizes” on page 19 and FIPS 140-2.

`gsk_generate_random_bytes()`

Generates a random byte stream.

Format

```
#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.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

Usage

The `gsk_generate_random_bytes()` routine will return a random byte stream. The application provides the buffer for the byte stream. The length value determines how many bytes will be generated.

System SSL attempts to use the ICSF PKCS #11 pseudo-random callable service (CSFPPRF) to generate a random byte stream. If ICSF is unavailable or returns an error and System SSL is in non-FIPS mode, an internal RNG will be used to generate the random data. If System SSL is in FIPS mode, the API call will fail.

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 afterward.

gsk_generate_secret()

Generates the Diffie-Hellman shared secret.

Format

```
#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()** 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, in non-FIPS mode, the optional key parameters Q and J are supplied as an ASN.1-encoded sequence as defined in either PKCS #3 or RFC 5280: *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

```
#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.

gsk_get_certificate_info()

Returns requested certificate information for an X.509 certificate.

Format

```
#include <gskcms.h>

gsk_status gsk_get_certificate_info(
    gsk_buffer *          cert_stream,
    x509_cert_info_id    cert_info_id,
    gsk_buffer *          cert_info)
```

Parameters

cert_stream

Specifies either a DER-encoded X.509 certificate or a non-decoded TBSCertificate ASN.1 sequence.

cert_info_id

The X.509 certificate information identifier specifying the certificate information to be returned.

cert_info

Returns the requested certificate information. The application should call the `gsk_free_buffer()` routine to release the certificate 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:

[ASN_NO_MEMORY]

Insufficient storage is available.

[ASN_ELEMENTS_MISSING]

Required data element is missing.

[ASN_UNSUPPORTED_VERSION]

Version is not supported.

[CMSERR_BAD_ISSUER_NAME]

Issuer name is not valid.

[CMSERR_BAD_SUBJECT_NAME]

Subject name is not valid.

[CMSERR_ATTRIBUTE_INVALID_ENUMERATION]

The enumeration value is not valid.

Usage

The `gsk_get_certificate_info()` routine returns information about an X.509 certificate. The certificate stream may be either:

gsk_get_certificate_info()

- An X.509 certificate encoded as described in RFC 5280: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*.
- The *derCertificate* field of the *x509_certificate* structure, which contains the non-decoded TBSCertificate ASN.1 sequence.

The application may request certificate information by using one of the following enumeration identifiers.

x509_cert_info_subject_dn_der

The subject distinguished name for the X.509 certificate in binary ASN.1 DER-encoded format.

x509_cert_info_issuer_dn_der

The issuer distinguished name for the X.509 certificate in binary ASN.1 DER-encoded format.

gsk_get_cms_vector()

Obtains the address of the Certificate Management Services function vector.

Format

```
#include <gskcms.h>

void gsk_get_cms_vector (
    gsk_uint32 *           function_mask,
    gsk_cms_vector **     function_vector)
```

Parameters

function_mask

Returns a bit mask indicating the Certificate Management Services level.

function_vector

Returns the address of the Certificate Management Services function vector.

Usage

Certificate Management Services (CMS) 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 CMS sidefile is specified as input to the binder. This causes all CMS functions to be resolved at bind time and will cause the CMS DLL to be implicitly loaded when the application is run.

In order to use runtime binding, the CMS DLL must be explicitly loaded by the application and the CMS functions must be called using indirect addresses. The **gsk_get_cms_vector()** routine allows an application to obtain the address of the CMS function vector containing an entry for each CMS 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 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.

GSKCMS_API_LVL7

CMS functions provided as part of z/OS Version 1 Release 12 are available.

gsk_get_cms_vector()

GSKCMS_API_LVL8

CMS functions provided as part of z/OS Version 1 Release 13 are available.

GSKCMS_API_LVL9

CMS functions provided as part of z/OS Version 2 Release 1 are available.

|
|

GSKCMS_API_LVL10

CMS functions provided as part of z/OS Version 2 Release 2 are available.

gsk_get_content_type_and_cms_version()

Extracts the PKCS #7 *content_info_type*, *content_info_oid*, and *cms_version* from the *pkcs_content_info* structure.

Format

```
#include <gskcms.h>
```

```
gsk_status gsk_get_content_type_and_cms_version (
    gsk_buffer *          content_info,
    pkcs_content_type *  content_info_type,
    gsk_oid *            content_info_oid,
    int *                cms_version)
```

Parameters

content_info

Specifies the encoded *content_info*.

content_info_type

Returns the *pkcs_content_type* enumeration value.

content_info_oid

Returns the *pkcs_content_type* OID. The application should call the **gsk_free_oid()** routine to release the OID elements when they are no longer needed.

cms_version

Returns the PKCS #7 CMSVersion as defined in RFC 3852.

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_CONTENT_DATA]

The content data length is zero.

[CMSERR_NO_MEMORY]

Insufficient storage is available.

Usage

The **gsk_get_content_type_and_cms_version()** routine extracts the type, object id data (OID) and the CMSVersion of the content data from a PKCS #7 (Cryptographic Message Syntax) ASN.1 DER-encoded content information sequence.

Returned field *content_info_oid* contains the OID extracted from the ContentInfo sequence. The returned *cms_version* is from the ContentInfo data type.

ContentInfo data type *pkcs_content_data* does not have a *cms_version* as defined by RFC 3852. For this type, -1 is returned as the *cms_version*.

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

```
#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>

gsk_status 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()` or the `gsk_create_revocation_source()` 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_LDAP_RESPONSE_TIMEOUT]

LDAP response not received within configured time limit.

[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 that are stored in the LDAP directory for the specified subject name. When matching UTF-8 encoded attribute values in the subject name, System SSL uses a case sensitive (exact match) comparison. The directory schema is defined by RFC 2587: *PKIX LDAP Version 2 Schema*. The certificates are stored as attributes of the subject

gsk_get_directory_certificates()

directory entry. Each certificate is encoded as defined by RFC 5280: *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 are released when the **gsk_close_directory()** or the **gsk_free_revocation_source()** 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

```
#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()` or the `gsk_create_revocation_source()` 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_LDAP_RESPONSE_TIMEOUT]

LDAP response not received within configured time limit.

[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. When matching UTF-8

encoded attribute values (`gsk_string_utf8`) in the issuer name, System SSL uses a case sensitive (exact match) comparison. The directory schema is defined by RFC 2587: *PKIX LDAP Version 2 Schema*. The revocation lists are stored as attributes of the issuer directory entry. Each CRL is encoded as defined by RFC 5280: *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.

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.

If the directory handle was created by the `gsk_open_directory()` routine with a non-zero value for the `crl_cache_timeout` parameter, retrieved CRLs are cached so that it is not necessary to contact the LDAP server for subsequent requests for the same issuer. The cached certificate revocation lists will be discarded at the end of the cache timeout specified on the `gsk_open_directory()` routine. The cached revocation lists will also be released when the `gsk_close_directory()` routine is called to close the directory handle. By default, the size of this cache is unlimited, but the size can be changed by invoking the `gsk_set_directory_numeric_value()` after the `gsk_open_directory()` routine. If the cache size is set, the maximum number of CRLs are already stored in the cache and the cache is not due to be flushed based on the cache timeout value, the CRL that was added earliest to the cache is removed. By default, the maximum size in bytes of a CRL that is allowed to be stored in the cache is unlimited or 0, but this size can be changed by calling the `gsk_set_directory_numeric_value()`. If the LDAP server does not contain an CRL for the specified CRL distribution point name and caching is active, by default, a temporary CRL will be cached to alleviate repeated calls to the LDAP server. If this behavior is not desired, it can be changed by invoking the `gsk_set_directory_enum_value()` routine and disabling the addition of these temporary CRLs to the cache. The cached revocation lists will be released when the `gsk_close_directory()` routine is called to close the directory handle.

If the directory handle was created by the `gsk_create_revocation_source()` routine with a non-zero value for the `crlCacheSize` parameter in the `gskdb_extended_directory_source` structure, retrieved CRLs that contain an expiration time that is later than the current time are cached so that it is not necessary to contact the LDAP server for subsequent requests for the same issuer. The cached certificate revocation list resides in the cache as long as the expiration time is not exceeded. Once the expiration time passes, the CRL is removed from the cache when referenced or when the cache size is about to be exceeded when adding new CRLs to the cache. If the `crlCacheSize` parameter is set to a non-zero value, the cache is already full, and the cache does not contain any expired CRLs, the CRL that is closest to expiration is removed from the cache. If the `crlCacheEntryMaxSize` parameter in the `gskdb_extended_directory_source` structure is set to 0, there is no limit on the size of an CRL that can be stored in the cache. However, if `crlCacheEntryMaxSize` is set to a value greater than 0, then that is the largest CRL in bytes that is allowed to be stored in the cache. If the LDAP server does not contain an CRL for the specified CRL distribution point name and caching is active, a temporary CRL will only be cached if the `crlCacheEmptyCRL` parameter in the `gskdb_extended_directory_source` structure is set to TRUE. Caching of a temporary CRL prevents repeated calls to the LDAP server when the CRL does not exist in the directory. The `crlCacheTempCRLTimeout` value in the `gskdb_extended_directory_source` structure indicates the time in hours that the

gsk_get_directory_crls()

| temporary CRL should reside in the cache. When the **gsk_free_revocation_source()**
| routine is called to close the extended directory handle, all cached certificate
| revocation lists and temporary CRLs will be released.

gsk_get_directory_enum()

Gets an enumerated value from an LDAP directory.

Format

```
#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_CACHE_TEMP_CRL

Returns whether an temporary LDAP CRL cache entry is added to the LDAP CRL cache when the CRL does not exist on the LDAP server.

One of two possible settings for `GSKCMS_CRL_CACHE_TEMP_CRL` will be returned:

GSKCMS_CRL_CACHE_TEMP_CRL_ON

A temporary CRL entry is added.

GSKCMS_CRL_CACHE_TEMP_CRL_OFF

A temporary CRL entry is not added.

gsk_get_directory_enum()

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_directory_numeric_value()

Gets an integer value from an LDAP directory.

Format

```
#include <gskcms.h>
```

```
gsk_status gsk_get_directory_numeric_value (
                                gsk_handle          directory_handle,
                                GSKCMS_DIRECTORY_NUM_ID num_id,
                                int *              num_value)
```

Parameters

directory_handle

Specifies an LDAP directory handle returned by `gsk_open_directory()`.

num_id

Specifies the directory numeric identifier.

num_value

Specifies the directory integer 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 numeric identifier is not valid or cannot be used with the specified handle.

[CMSERR_BAD_HANDLE]

The handle is not valid.

[CMSERR_INVALID_NUMERIC_VALUE]

The numeric value is not valid.

Usage

The `gsk_get_directory_numeric_value()` routine returns an integer value for an LDAP directory.

These numeric identifiers are supported:

GSKCMS_CRL_CACHE_ENTRY_MAXSIZE

Returns the maximum size in bytes of a CRL that is allowed to be stored in the LDAP CRL cache.

GSKCMS_CRL_CACHE_SIZE

Returns the maximum number of CRLs that are allowed to be stored in the LDAP CRL cache.

GSKCMS_LDAP_RESPONSE_TIMEOUT

Returns the time in seconds to wait for a response from the LDAP server.

`gsk_get_ec_parameters_info()`

Get the named curve type and key size for EC domain parameters.

Format

```
#include <gskcms.h>

gsk_status gsk_get_ec_parameters_info (
    gsk_buffer *          ec_parameters,
    x509_ec_parameters_info * key_info)
```

Parameters

ec_parameters

Specifies the ASN.1-encoded EC domain parameters to be analyzed.

key_info

Returns the elliptic curve information.

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_EC_PARAMS]

Elliptic Curve parameters are not valid.

[CSMERR_EC_PARAMETERS_NOT_SUPPLIED]

EC parameters not supplied.

[CMSERR_STRUCTURE_TOO_SMALL]

Size specified for supplied structure is too small.

Usage

The `gsk_get_ec_parameters_info()` routine returns the elliptic curve type and key size of the supplied EC domain parameters. Before calling the function, the application must initialize the size field in *key_info* to the size of the `x509_ec_parameters_info` structure.

gsk_get_record_by_id()

Gets a database record using the record identifier.

Format

```
#include <gskcms.h>

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

```
#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()

Gets a database record using the record label.

Format

```
#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()`

`gsk_get_record_by_subject()`

Gets one or more database records using the certificate subject.

Format

```
#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. When matching UTF-8 encoded attribute values (`gsk_string_utf8`) in the subject name, System SSL uses a case sensitive (exact match) comparison.

gsk_get_record_labels()

Gets the record labels for a key or request database.

Format

```
#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

Specifies 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()

gsk_get_update_code()

Gets the database update code.

Format

```
#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

```
#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 `gskcms.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_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

gsk_import_certificate()

[CMSERR_EXPIRED]

The certificate is expired.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[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

```
#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.
- [CMSERR_ECURVE_NOT_FIPS_APPROVED]**
Elliptic Curve not supported in FIPS mode.
- [CMSERR_ECURVE_NOT_SUPPORTED]**
Elliptic Curve is not supported.
- [CMSERR_EXPIRED]**
The certificate is expired.
- [CMSERR_ICSF_FIPS_DISABLED]**
ICSF PKCS #11 services are disabled.
- [CMSERR_ICSF_NOT_AVAILABLE]**
ICSF services are not available.
- [CMSERR_ICSF_NOT_FIPS]**
ICSF PKCS #11 not operating in FIPS mode.
- [CMSERR_ICSF_SERVICE_FAILURE]**
ICSF callable service returned an error.
- [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_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

gsk_import_key()

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 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

```
#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()

gsk_make_data_content()

Creates PKCS #7 Data content information from application data.

Format

```
#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

```
#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()`

`gsk_make_encrypted_data_content()`

Creates PKCS #7 EncryptedData content information.

Format

```
#include <gskcms.h>
```

```
gsk_status 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

[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, Version 2.0: *Password-based Encryption* and PKCS #12, Version 1.0: *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

```
#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

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.

gsk_make_encrypted_data_msg() is not supported when executing in FIPS mode and will return CMSERR_API_NOT_SUPPORTED.

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, Version 2.0: *Password-based Encryption* and PKCS #12, Version 1.0: *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 standard version:

- Specify 0 to create EnvelopedData content as described in PKCS #7 Version 1.5. This version encodes the *IssuerAndSerialNumber* as the *RecipientIdentifier*.
- Specify 1 to create EnvelopedData content as described in PKCS #7 Version 1.6. This version encodes the *IssuerAndSerialNumber* as the *RecipientIdentifier*.
- Specify 2 to create EnvelopedData content as described in PKCS #7 RFC 3852. This version encodes the *SubjectKeyIdentifier* as the *RecipientIdentifier*.

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.

[CMSERR_CONTENT_NOT_SUPPORTED]

The content type is not supported

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_INCORRECT_KEY_USAGE]

A recipient certificate does not allow key encipherment

[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_REQUIRED_EXT_MISSING]

Required certificate extension is missing.

[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 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}

gsk_make_enveloped_data_content()

- **x509_alg_aesCbc128** - 128-bit AES CBC - Key length 16 - {2.16.840.1.101.3.4.1.2}
- **x509_alg_aesCbc256** - 256-bit AES CBC - Key length 32 - {2.16.840.1.101.3.4.1.42}

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

```
#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 encipherment capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate keyEncipherment.

version

Specifies the PKCS #7 EnvelopedData standard version:

- Specify 0 to create EnvelopedData content as described in PKCS #7 Version 1.5. This version encodes the *IssuerAndSerialNumber* as the *RecipientIdentifier*.
- Specify 1 to create EnvelopedData content as described in PKCS #7 Version 1.6. This version encodes the *IssuerAndSerialNumber* as the *RecipientIdentifier*.
- Specify 2 to create EnvelopedData content as described in PKCS #7 RFC 3852. This version encodes the *SubjectKeyIdentifier* as the *RecipientIdentifier*.

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.

gsk_make_enveloped_data_content_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:

[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.

[CMSERR_CONTENT_NOT_SUPPORTED]

The content type is not supported

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_INCORRECT_KEY_USAGE]

A recipient certificate does not allow key encipherment

[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_REQUIRED_EXT_MISSING]

Required certificate extension is missing.

[CMSERR_VERSION_NOT_SUPPORTED]

The version is not valid

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_read_enveloped_data_content()** routine or the **gsk_read_enveloped_data_content_extended()** 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

gsk_make_enveloped_data_content_extended()

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}**
- **x509_alg_aesCbc128 - 128-bit AES CBC - Key length 16 - {2.16.840.1.101.3.4.1.2}**
- **x509_alg_aesCbc256 - 256-bit AES CBC - Key length 32 - {2.16.840.1.101.3.4.1.42}**

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

```
#include <gskcms.h>
```

```
gsk_status gsk_make_enveloped_data_msg (
    int                version,
    pkcs_session_key * session_key,
    pkcs_certificates * recipient_certificates,
    gsk_buffer *       data,
    gsk_buffer *       stream)
```

Parameters

version

Specifies the PKCS #7 EnvelopedData standard version:

- Specify 0 to create an EnvelopedData message as described in PKCS #7 Version 1.5. This version encodes the *IssuerAndSerialNumber* as the *RecipientIdentifier*.
- Specify 1 to create an EnvelopedData message as described in PKCS #7 Version 1.6. This version encodes the *IssuerAndSerialNumber* as the *RecipientIdentifier*.
- Specify 2 to create an EnvelopedData message as described in PKCS #7 RFC 3852. This version encodes the *SubjectKeyIdentifier* as the *RecipientIdentifier*.

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.

[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_REQUIRED_EXT_MISSING]

Required certificate extension is missing.

[CMSERR_VERSION_NOT_SUPPORTED]

The version is not valid.

Usage

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}

gsk_make_enveloped_data_msg()

- **x509_alg_aesCbc128** - 128-bit AES CBC - Key length 16 - {2.16.840.1.101.3.4.1.2}
- **x509_alg_aesCbc256** - 256-bit AES CBC - Key length 32 - {2.16.840.1.101.3.4.1.42}

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 encipherment capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate keyEncipherment.

version

Specifies the PKCS #7 EnvelopedData standard version:

- Specify 0 to create an EnvelopedData message as described in PKCS #7 Version 1.5. This version encodes the *IssuerAndSerialNumber* as the *RecipientIdentifier*.
- Specify 1 to create an EnvelopedData message as described in PKCS #7 Version 1.6. This version encodes the *IssuerAndSerialNumber* as the *RecipientIdentifier*.
- Specify 2 to create an EnvelopedData message as described in PKCS #7 RFC 3852. This version encodes the *SubjectKeyIdentifier* as the *RecipientIdentifier*.

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.

gsk_make_enveloped_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:

[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.

[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_REQUIRED_EXT_MISSING]

Required certificate extension is missing.

[CMSERR_VERSION_NOT_SUPPORTED]

The version is not valid.

Usage

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

gsk_make_enveloped_data_msg_extended()

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}
- `x509_alg_aesCbc128` - 128-bit AES CBC - Key length 16 - {2.16.840.1.101.3.4.1.2}
- `x509_alg_aesCbc256` - 256-bit AES CBC - Key length 32 - {2.16.840.1.101.3.4.1.42}

When executing in FIPS mode, encryption algorithms `x509_alg_rc2CbcPad`, `x509_alg_rc4` and `x509_alg_desCbcPad` are not supported.

`gsk_make_enveloped_private_key_msg()`

Creates a PKCS #7 EnvelopedData message from application data. The application data passed in is the PKCS #11 secure key label name.

Format

```
#include <gskcms.h>

gsk_status gsk_make_enveloped_private_key_msg (
    gsk_uint32          option_flag,
    int                 version,
    x509_algorithm_type encryption_algorithm,
    pkcs_certificates * recipient_certificates,
    gsk_buffer *       secure_key_label,
    gsk_buffer *       stream)
```

Parameters

option_flag

Specifies process options to customize process behavior. Specify execution options using bit setting.

- `GSK_PROCESS_OPTION_ENFORCE_KEYUSAGE` - 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 is supported.
- Any other bit values are ignored.

version

Specify PKCS #7 EnvelopedData version number. Only version 0, PKCS #7 Version 1.5, is supported.

encryption_algorithm

Specifies the algorithm to be used:

- `x509_alg_aesCbc128` for AES with Key length 16.
- `x509_alg_aesCbc256` for AES with Key length 32.
- `x509_alg_desEde3CbcPad` for 3DES with Key length 24.

recipient_certificates

Specifies the certificates for the message recipients. There must be at least one recipient.

secure_key_label

Specifies a PKCS #11 secure private key label object. No other type of object is supported.

stream

Returns the ASN.1 DER-encoded stream. The application calls the `gsk_free_buffer()` routine to release the stream 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 `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 recipient key size is not supported.

[CMSERR_CRYPTO_HARDWARE_NOT_AVAILABLE]

Cryptographic hardware does not support service or algorithm.

[CMSERR_ICSF_FIPS_BAD_ALG_OR_KEY_SIZE]

A recipient algorithm or key size is not FIPS approved for an ICSF operation.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_FIPS]

ICSF is not operating in FIPS mode.

[CMSERR_INCORRECT_KEY_ATTRIBUTE]

Key attributes do not support envelope operation.

[CMSERR_INCORRECT_KEY_USAGE]

A recipient certificate does not allow key encipherment.

[CMSERR_KEY_MISMATCH]

A recipient public key does not support data encryption.

[CMSERR_NO_MEMORY]

Insufficient storage is available.

[CMSERR_NO_PKCS11_OBJECT_NOT_FOUND]

A PKCS #11 key label is either missing or not valid.

[CMSERR_NO_PRIVATE_KEY]

No private key.

[CMSERR_PKCS11_LABEL_INVALID]

PKCS #11 label is not valid.

[CMSERR_RECIPIENT_NOT_FOUND]

No recipient certificates provided.

Usage

The **gsk_make_enveloped_private_key_msg()** routine creates a PKCS #7 (Cryptographic Message Syntax) EnvelopedData message using a TKDS secure private key label and returns the ASN.1 DER-encoded ContentInfo sequence. The **gsk_read_enveloped_data_content()** routine or the **gsk_read_enveloped_data_content_extended()** 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 validated the recipient certificates.

A session key is used to encrypt the message content. A new session key is generated but is not returned to the application. For each recipient, the session key is encrypted with the public key of the recipient and stored in the EnvelopedData message. Each recipient's public key must be type RSA.

In addition, if *option_flag* specifies that key usage is to be enforced, then each recipient certificate key usage must allow key encipherment.

These encryption algorithms are supported. Strong encryption might not be available, depending upon government export regulations.

- **x509_alg_desEde3CbcPad - 168-bit 3DES - Key length 24 - {1.2.840.113549.3.7}**

gsk_make_enveloped_private_key_msg()

- x509_alg_aesCbc128 - 128-bit AES CBC - Key length 16 - {2.16.840.1.101.3.4.1.2}
- x509_alg_aesCbc256 - 256-bit AES CBC - Key length 32 - {2.16.840.1.101.3.4.1.42}

gsk_make_signed_data_content()

Creates PKCS #7 SignedData content information.

Format

```
#include <gskcms.h>
```

```
gsk_status gsk_make_signed_data_content (
    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,
    pkcs_content_info * content_info)
```

Parameters

version

Specifies the PKCS #7 SignedData standard version:

- Specify 0 to create SignedData content as described in PKCS #7 Version 1.4. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 1 to create SignedData content as described in PKCS #7 Version 1.5. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 2 to create SignedData content as described in PKCS #7 Version 1.6. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 3 to create SignedData content as described in PKCS #7 RFC 3852. This version encodes the *SubjectKeyIdentifier* as the *signerIdentifier*.

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.

gsk_make_signed_data_content()

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_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[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_NO_PRIVATE_KEY]

Private key does not exist or is not accessible.

[CMSERR_REQUIRED_EXT_MISSING]

Required certificate extension is missing.

[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_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, DSA, and ECDSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest

SHA-224 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest

SHA-256 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest

SHA-384 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest

SHA-512 digest (RSA and ECDSA 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

```
#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.
- Do not allow zero-length content data

version

Specifies the PKCS #7 SignedData standard version:

- Specify 0 to create SignedData content as described in PKCS #7 Version 1.4. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 1 to create SignedData content as described in PKCS #7 Version 1.5. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 2 to create SignedData content as described in PKCS #7 Version 1.6. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 3 to create SignedData content as described in PKCS #7 RFC 3852. This version encodes the *SubjectKeyIdentifier* as the *signerIdentifier*.

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.

gsk_make_signed_data_content_extended()

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 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. If version 3 is requested, authenticated attributes continue to utilize the signer certificate's *IssuerAndSerialNumber*.

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_CONTENTTYPE_NOT_ALLOWED]

The content-type authenticated attribute is not allowed in *attributes_signers*.

[CMSERR_DIGEST_KEY_MISMATCH]

The digest algorithm is not supported for the private key type.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[CMSERR_INCORRECT_KEY_USAGE]

A signer certificate does not allow digital signature.

[CMSERR_MESSAGEDIGEST_NOT_ALLOWED]

The message-digest authenticated attribute is not allowed in *attributes_signers*.

[CMSERR_NO_CONTENT_DATA]

The content data length is zero.

gsk_make_signed_data_content_extended()

[CMSERR_NO_MEMORY]

Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]

Private key does not exist or is not accessible.

[CMSERR_REQUIRED_EXT_MISSING]

Required certificate extension is missing.

[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.

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, DSA, and ECDSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest

SHA-224 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest

SHA-256 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.1}

gsk_make_signed_data_content_extended()

x509_alg_sha384Digest

SHA-384 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest

SHA-512 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.3}

If authenticated attributes are provided from the *attributes_signers* parameter, then signing certificates for all signers represented within the *gsk_attributes_signers* structure must be provided from the *signer_certificates* parameter.

When executing in FIPS mode, digest algorithms *x509_alg_md2Digest* and *x509_alg_md5Digest* are not supported.

gsk_make_signed_data_msg()

Creates a PKCS #7 SignedData message from application data.

Format

```
#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_cert_keys *  ca_certificates,
    gsk_buffer *       data,
    gsk_buffer *       stream)
```

Parameters

version

Specifies the PKCS #7 SignedData standard version:

- Specify 0 to create SignedData content as described in PKCS #7 Version 1.4. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 1 to create SignedData content as described in PKCS #7 Version 1.5. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 2 to create SignedData content as described in PKCS #7 Version 1.6. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 3 to create SignedData content as described in PKCS #7 RFC 3852. This version encodes the *SubjectKeyIdentifier* as the *signerIdentifier*.

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.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[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_NO_PRIVATE_KEY]

Private key does not exist or is not accessible.

[CMSERR_REQUIRED_EXT_MISSING]

Required certificate extension is missing.

[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.

gsk_make_signed_data_msg()

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, DSA, and ECDSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest

SHA-224 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest

SHA-256 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest

SHA-384 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest

SHA-512 digest (RSA and ECDSA 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

```
#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.
- Do not allow zero-length content data

version

Specifies the PKCS #7 SignedData standard version:

- Specify 0 to create SignedData content as described in PKCS #7 Version 1.4. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 1 to create SignedData content as described in PKCS #7 Version 1.5. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 2 to create SignedData content as described in PKCS #7 Version 1.6. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 3 to create SignedData content as described in PKCS #7 RFC 3852. This version encodes the *SubjectKeyIdentifier* as the *signerIdentifier*.

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.

gsk_make_signed_data_msg_extended()

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. If version 3 is requested, authenticated attributes continue to utilize the signer certificate's *IssuerAndSerialNumber*.

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_CONTENTTYPE_NOT_ALLOWED]

The content-type authenticated attribute is not allowed in *attributes_signers*.

[CMSERR_DIGEST_KEY_MISMATCH]

The digest algorithm is not supported for the private key type.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[CMSERR_INCORRECT_KEY_USAGE]

A signer certificate does not allow digital signature.

[CMSERR_MESSAGEDIGEST_NOT_ALLOWED]

The message-digest authenticated attribute is not allowed in *attributes_signers*.

[CMSERR_NO_CONTENT_DATA]

The content data length is zero.

[CMSERR_NO_MEMORY]

Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]

Private key does not exist or is not accessible.

[CMSERR_REQUIRED_EXT_MISSING]

Required certificate extension is missing.

[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, DSA, and ECDSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest

SHA-224 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.4}

gsk_make_signed_data_msg_extended()

x509_alg_sha256Digest

SHA-256 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest

SHA-384 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest

SHA-512 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.3}

If authenticated attributes are provided from the *attributes_signers* parameter, then signing certificates for all signers represented within the *gsk_attributes_signers* structure must be provided from 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>
```

```
gsk_status gsk_make_wrapped_content (
                                     pkcs_content_info *      content_info,
                                     pkcs_content_info *      wrapped_content)
```

Parameters

content_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()

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_modify_pkcs11_key_label()

Return a `gsk_buffer` that adds or removes the equals sign (=) from the first position of an input TKDS key token label.

Format

```
#include <gskcms.h>

gsk_status gsk_modify_pkcs11_key_label (
    gsk_buffer *      in_buffer,
    gsk_boolean      add_preface,
    gsk_buffer *      out_buffer)
```

Parameters

in_buffer

Specifies the `gsk_buffer` containing the TKDS key token label.

add_preface

Specify TRUE if you want an equal sign (=) prefaced at the beginning of the TKDS key token label. This shifts the original string to the right one position.

Specify FALSE if you want the equal sign (=) removed from the beginning of the TKDS key token label. This shifts the original string to the left one position.

out_buffer

Returns a new `gsk_buffer` with the TKDS key token label in its new form.

Results

The function return value is 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_NO_MEMORY]

Insufficient storage is available.

[CMSERR_PKCS11_LABEL_INVALID]

The input did not have a label.

If *add_preface* is FALSE and the calculated length of the output string minus the equal sign (=) is zero.

Usage

The `gsk_modify_pkcs11_key_label()` routine creates `gsk_buffer` of the TKDS key label either with or without the equal sign (=) in the first position of the string.

- A TKDS key label without the equals sign is always 44 characters long.
- A TKDS key label with an equal sign is always 45 characters long.
- The caller is responsible for freeing the storage that is allocated to create the returned *out_buffer*.
 - If the returned *out_buffer* does not become the *keyToken* field of structure `pkcs_private_key_info`, call `gsk_free_buffer()` to free *out_buffer*.
 - If the returned *out_buffer* is used as the *keyToken* field of structure `pkcs_private_key_info`, the returned *out_buffer* is freed when calling `gsk_free_private_key_info()`.

gsk_modify_pkcs11_key_label()

- If the supplied TKDS key token label already has the equal sign in the first position and *add_preface* is TRUE, a copy of the original string is returned.
- If the supplied TKDS key token label already does not have the equal sign in the first position and *add_preface* is FALSE, a copy of the original string is returned.

gsk_name_compare()

Compares two X.509 names.

Format

```
#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 on their ASN.1 DER encoding. Strings are always stored using UTF-8 encoding. When matching UTF-8 encoded attribute values (x509_string_utf8) in the X.509 names, System SSL uses a case sensitive (exact match) comparison.

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

```
#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 RFC 822 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

gsk_open_database()

gsk_open_database()

Opens a key or request database.

Format

```
#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.

[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_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()** 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 if 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

```
#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.

[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 if 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()`

`gsk_open_directory()`

Opens an LDAP directory.

Format

```
#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_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 user ID 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 key ring.

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_RNG_OUTPUT]

In FIPS mode, random bytes generation produced duplicate output.

[CMSERR_FILE_NOT_FOUND]

The key ring or token does not exist

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[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

gsk_open_keyring()

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:

Certificate private keys are not available when using a SAF key ring owned by another user, except for SITE certificates where CONTROL authority is given to IRR.DIGTCERT.GENCERT in the FACILITY class or for user certificates where READ or UPDATE authority is given to *ringOwner.ringName.LST* resource in the RDATA LIB class.

The application user ID must have READ access to resource USER.tokenname 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 292) before 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.

[CMSERR_KATPW_ICSF_FAILED]

A known answer test failed because ICSF was not available or ICSF encountered an error.

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 CBC 128-bit and AES CBC 256-bit encryption and decryption
- DSA signature generation and verification
- RSA encrypt and decrypt
- RSA signature generation/verification and encryption/decryption
- SHA Digest Algorithms: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, HMAC-SHA-1, HMAC-SHA-256, and HMAC-SHA-384
- TLS V1.0, V1.1 and V1.2 key derivation function
- TripleDES encryption and decryption

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 run time version, release, and available cryptographic levels. The current System SSL run time is Version 4 Release 2. The cryptographic level is a bit mask as follows:

[GSK_CRYPT0_64]

Set if 64-bit encryption keys are supported.

[GSK_CRYPT0_128]

Set if 128-bit encryption keys are supported.

[GSK_CRYPT0_168]

Set if 168-bit encryption keys are supported.

gsk_query_database_label()

Determines if a database label exists

Format

```
#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()

gsk_query_database_record_length()

Queries the database record length.

Format

```
#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_rdttime()

Converts the number of seconds since the POSIX epoch to year/month/day.

Format

```
#include <gskcms.h>
```

```
gsk_timeval * gsk_rdttime (
    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_rdttime()** 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>
```

```
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

```
#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()`

`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:

[CMSERR_CONTENT_NOT_SUPPORTED]

The message content type is not Data.

[CMSERR_NO_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

```
#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**.

gsk_read_encrypted_data_content()

The decryption key is derived from the password as described in PKCS #5, Version 2.0: *Password-based Encryption* and PKCS #12, Version 1.0: *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 might 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

```
#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.

`gsk_read_encrypted_data_msg()` is not supported when executing in FIPS mode and will return `CMSERR_API_NOT_SUPPORTED`.

gsk_read_encrypted_data_msg()

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, Version 2.0: *Password-based Encryption* and PKCS #12, Version 1.0: *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 might 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_CRYPTO_HARDWARE_NOT_AVAILABLE]

Cryptographic hardware does not support service or algorithm.

[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.

gsk_read_enveloped_data_content()

[CMSERR_NO_MEMORY]

Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]

Private key does not exist or is not accessible.

[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 might 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}
- **x509_alg_aesCbc128** - 128-bit AES CBC - {2.16.840.1.101.3.4.1.2}
- **x509_alg_aesCbc256** - 256-bit AES CBC - {2.16.840.1.101.3.4.1.42}

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

```
#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 is 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_CRYPTOHARDWARE_NOT_AVAILABLE]

Cryptographic hardware does not support service or algorithm.

gsk_read_enveloped_data_content_extended()

[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.

[CMSERR_NO_PRIVATE_KEY]

Private key does not exist or is not accessible.

[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 that is created by the **gsk_make_enveloped_data_content()** routine, the **gsk_make_enveloped_data_content_extended()**, or the **gsk_make_enveloped_private_key_msg()** 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 searches 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 might 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}
- **x509_alg_aesCbc128** - 128-bit AES CBC - {2.16.840.1.101.3.4.1.2}
- **x509_alg_aesCbc256** - 256-bit AES CBC - {2.16.840.1.101.3.4.1.42}

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

```
#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_CRYPTO_HARDWARE_NOT_AVAILABLE]

Cryptographic hardware does not support service or algorithm.

[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.

gsk_read_enveloped_data_msg()

[CMSERR_NO_MEMORY]

Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]

Private key does not exist or is not accessible.

[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 encipherment.

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 might 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}
- **x509_alg_aesCbc128** - 128-bit AES CBC - {2.16.840.1.101.3.4.1.2}
- **x509_alg_aesCbc256** - 256-bit AES CBC - {2.16.840.1.101.3.4.1.42}

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_CRYPTOHARDWARE_NOT_AVAILABLE]

Cryptographic hardware does not support service or algorithm.

gsk_read_enveloped_data_msg_extended()

[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.

[CMSERR_NO_PRIVATE_KEY]

Private key does not exist or is not accessible.

[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 might 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}
- **x509_alg_aesCbc128** - 128-bit AES CBC - {2.16.840.1.101.3.4.1.2}
- **x509_alg_aesCbc256** - 256-bit AES CBC - {2.16.840.1.101.3.4.1.42}

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

```
#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.

gsk_read_signed_data_content()

[CMSERR_DIGEST_KEY_MISMATCH]

The digest algorithm is not supported for the private key type.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[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()** 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 attempts 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}

gsk_read_signed_data_content()

x509_alg_sha1Digest

SHA-1 digest (RSA, DSA, and ECDSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest

SHA-224 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest

SHA-256 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest

SHA-384 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest

SHA-512 digest (RSA and ECDSA 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.
- Do not 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

gsk_read_signed_data_content_extended()

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_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[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).

gsk_read_signed_data_content_extended()

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 attempts 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}

x509_alg_sha1Digest

SHA-1 digest (RSA, DSA, and ECDSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest

SHA-224 digest (RSA DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest

SHA-256 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest

SHA-384 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest

SHA-512 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.3}

If authenticated attributes are returned from the *attributes_signers* parameter, then it is recommended that signing certificates for all signers represented within the *gsk_attributes_signers* structure should be requested from 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.

gsk_read_signed_data_msg()

[CMSERR_BAD_SIGNATURE]

Signature is not correct.

[CMSERR_CONTENT_NOT_SUPPORTED]

The message content type is not SignedData or the content of the SignedData message is not Data.

[CMSERR_DIGEST_KEY_MISMATCH]

The digest algorithm is not supported for the private key type.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[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 attempts 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, DSA, and ECDSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest

SHA-224 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest

SHA-256 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest

SHA-384 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest

SHA-512 digest (RSA and ECDSA 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

```
#include <gskcms.h>
```

```
gsk_status gsk_read_signed_data_msg_extended (  
    gsk_process_option          option_flag  
    pkcs_certificates *        local_certificates,  
    gsk_buffer *                stream,  
    gsk_boolean *              used_local,  
    pkcs_certificates *        msg_certificates,  
    pkcs_certificates *        signer_certificates,  
    gsk_attributes_signers *   attributes_signers,  
    gsk_buffer *                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.
- Do not 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 is set to TRUE if the signatures were verified by using just the certificates that are supplied by the `local_certificates` parameter. This parameter is set to FALSE if any of the signatures were verified by using certificates that are contained within the message.

msg_certificates

Returns the X.509 certificates that are 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 that are 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 that is 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 that is 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.

Results

The function return value will be 0 if no error is detected. Otherwise, it is 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_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[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.

gsk_read_signed_data_msg_extended()

Usage

The **gsk_read_signed_data_msg_extended()** routine processes a PKCS #7 (Cryptographic Message Syntax) SignedData message that is 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 that is 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 that are used to verify the message signatures. If a certificate is not found for a message signer, the **gsk_read_signed_data_msg_extended()** routine attempts to locate the signer certificate in the SignedData message. An error is 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 validated the local certificates. The certificates that are contained in the SignedData message are returned in the *msg_certificates* parameter and the *used_local* parameter is 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_mode()** 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, DSA, and ECDSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest

SHA-224 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest

SHA-256 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest

SHA-384 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest

SHA-512 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.3}

gsk_read_signed_data_msg_extended()

If authenticated attributes are returned from the *attributes_signers* parameter, then it is recommended that signing certificates for all signers represented within the *gsk_attributes_signers* structure should be requested from 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()`

`gsk_read_wrapped_content()`

Processes wrapped content information.

Format

```
#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

```
#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_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[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.

[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_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

gsk_replace_record()

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

```
#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_BACKUP_EXISTS]

The backup file already exists.

[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.

gsk_set_default_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_CACHE_TEMP_CRL

Specifies if a temporary LDAP CRL cache entry is added to the LDAP CRL cache when the CRL does not exist on the LDAP server:

GSKCMS_CRL_CACHE_TEMP_CRL_ON

A temporary CRL entry is added. This is the default setting.

GSKCMS_CRL_CACHE_TEMP_CRL_OFF

A temporary CRL entry is not added.

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_set_directory_enum()

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 102 and Appendix A, “Environment variables,” on page 667 for additional information about 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_set_directory_numeric_value()

Sets an integer value for an LDAP directory.

Format

```
#include <gskcms.h>
```

```
gsk_status gsk_set_directory_numeric_value (
                                gsk_handle          directory_handle,
                                GSKCMS_DIRECTORY_NUM_ID num_id,
                                int                 num_value)
```

Parameters

directory_handle

Specifies an LDAP directory handle returned by `gsk_open_directory()`.

num_id

Specifies the directory numeric identifier.

num_value

Specifies the directory integer 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 numeric identifier is not valid or cannot be used with the specified handle.

[CMSERR_BAD_HANDLE]

The handle is not valid.

[CMSERR_INVALID_NUMERIC_VALUE]

The numeric value is not valid.

Usage

The `gsk_set_directory_numeric_value()` routine sets an integer value for an LDAP directory. The LDAP directory must have a valid LDAP handle as initialized using `gsk_open_directory()`.

These numeric identifiers are supported:

GSKCMS_CRL_CACHE_ENTRY_MAXSIZE

Specifies the maximum size in bytes of a CRL that is allowed to be stored in the LDAP CRL cache. Any CRLs larger than this size are not cached. The default is 0, which means there is no limit on the size of the CRL stored in the LDAP CRL cache.

GSKCMS_CRL_CACHE_SIZE

Specifies the maximum number of CRLs that are allowed to be stored in the LDAP CRL cache. The valid cache sizes are -1 through 32000. Specify -1 if the LDAP CRL cache size is to be unlimited. Caching only occurs if the *crl_cache_timeout* specified on the `gsk_open_directory()` call is not 0. The default is unlimited.

gsk_set_directory_numeric_value()

| **GSKCMS_LDAP_RESPONSE_TIMEOUT**
| Specifies the time in seconds to wait for a response from the LDAP server.
| The default is 30 seconds. A value of 0 indicates that there is no time limit.
|

gsk_sign_certificate()

Signs an X.509 certificate.

Format

```
#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_EC_PARAMS]

Elliptic Curve parameters are not valid.

[CMSERR_BAD_KEY_SIZE]

The key size is not valid.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[CMSERR_KEY_MISMATCH]

The supplied key does not match the signature algorithm.

[CMSERR_NO_MEMORY]

Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]

Private key does not exist or is not accessible.

gsk_sign_certificate()

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, DSA key, or an ECDSA key. If executing in FIPS mode, the minimum key size for RSA and DSA keys is 1024 bits, and the minimum key size for ECDSA keys is 192 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}

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_dsaWithSha224

Digital Signature Standard with SHA-224 digest - {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256

Digital Signature Standard with SHA-256 digest - {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1

Elliptic Curve Digital Signature Algorithm with SHA-1 digest -
{1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224

Elliptic Curve Digital Signature Algorithm with SHA-224 digest -
{1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256

Elliptic Curve Digital Signature Algorithm with SHA-256 digest -
{1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384

Elliptic Curve Digital Signature Algorithm with SHA-384 digest -
{1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512

Elliptic Curve Digital Signature Algorithm with SHA-512 digest –
{1.2.840.10045.4.3.4}

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

```
#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_EC_PARAMS]

Elliptic Curve parameters are not valid.

[CMSERR_BAD_KEY_SIZE]

The key size is not valid.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[CMSERR_KEY_MISMATCH]

The supplied key does not match the signature algorithm.

[CMSERR_NO_MEMORY]

Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]

Private key does not exist or is not accessible.

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, a DSA key, or an ECDSA key. If executing in FIPS mode, the minimum key size for RSA and DSA keys is 1024 bits, and the minimum key size for ECDSA keys is 192 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}

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_dsaWithSha224

Digital Signature Standard with SHA-224 digest - {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256

Digital Signature Standard with SHA-256 digest - {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1

Elliptic Curve Digital Signature Algorithm with SHA-1 digest - {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224

Elliptic Curve Digital Signature Algorithm with SHA-224 digest - {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256

Elliptic Curve Digital Signature Algorithm with SHA-256 digest - {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384

Elliptic Curve Digital Signature Algorithm with SHA-384 digest - {1.2.840.10045.4.3.3}

gsk_sign_crl()

x509_alg_ecdsaWithSha512

Elliptic Curve Digital Signature Algorithm with SHA-512 digest –
{1.2.840.10045.4.3.4}

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

```
#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_EC_PARAMS]

Elliptic Curve 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_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

gsk_sign_data()

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[CMSERR_INVALID_KEY_ATTRIBUTE]

Key does not have required PKCS #11 attributes to perform signing.

[CMSERR_KEY_MISMATCH]

The supplied key does not match the signature algorithm.

[CMSERR_NO_MEMORY]

Insufficient storage is available.

[CMSERR_NO_PKCS11_KEY_LABEL]

A TKDS secure key label is either invalid or missing.

[CMSERR_NO_PRIVATE_KEY]

Private key does not exist, is not accessible, or the PKCS #11 TKDS secure key is not a supported algorithm type.

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, a DSA key, or an ECDSA key. If executing in FIPS mode, the minimum key size for RSA and DSA keys is 1024 bits, and the minimum size for ECDSA keys is 192 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 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_dsaWithSha224**
Digital Signature Standard with SHA-224 digest - {2.16.840.1.101.3.4.3.1}
- x509_alg_dsaWithSha256**
Digital Signature Standard with SHA-256 digest - {2.16.840.1.101.3.4.3.2}
- x509_alg_ecdsaWithSha1**
Elliptic Curve Digital Signature Algorithm with SHA-1 digest -
{1.2.840.10045.4.1}
- x509_alg_ecdsaWithSha224**
Elliptic Curve Digital Signature Algorithm with SHA-224 digest -
{1.2.840.10045.4.3.1}
- x509_alg_ecdsaWithSha256**
Elliptic Curve Digital Signature Algorithm with SHA-256 digest -
{1.2.840.10045.4.3.2}
- x509_alg_ecdsaWithSha384**
Elliptic Curve Digital Signature Algorithm with SHA-384 digest -
{1.2.840.10045.4.3.3}
- x509_alg_ecdsaWithSha512**
Elliptic Curve Digital Signature Algorithm with SHA-512 digest -
{1.2.840.10045.4.3.4}
- 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_validate_certificate()`

`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_HTTP_RESPONSE]

HTTP response 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_OCSP_RESPONSE]

OCSP response 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_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_EXPIRED]

The certificate is expired.

[CMSERR_HOSTNAME_NOT_VALID]

HTTP server hostname is not valid.

[CMSERR_HTTP_IO_ERROR]

HTTP server communication error.

[CMSERR_HTTP_RESPONSE_TIMEOUT]

HTTP response not received within the configured time limit.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]

The database type does not support certificates.

[CMSERR_INCORRECT_KEY_USAGE]

The issuer certificate does not allow signing certificates

[CMSERR_INTERNAL_ERROR]

An internal processing error has occurred.

[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_RESPONSE_TIMEOUT]

LDAP response not received within configured time limit.

[CMSERR_MAX_RESPONSE_SIZE_EXCEEDED]

The maximum response size has been exceeded.

[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.

gsk_validate_certificate()

| [CMSERR_NO_MEMORY]
| Insufficient storage is available.

| [CMSERR_NOT_YET_VALID]
| The certificate is not yet valid.

| [CMSERR_OCSP_NONCE_CHECK_FAILED]
| Nonce in OCSP response does not match value in OCSP request.

| [CMSERR_OCSP_RESPONDER_ERROR]
| OCSP request failed with internal responder error.

| [CMSERR_OCSP_RESPONSE_TIMEOUT]
| OCSP response was not received within the configured time limit.

| [CMSERR_OCSP_REQ_ERROR]
| Error creating the OCSP request.

| [CMSERR_OCSP_SIG_REQUIRED]
| OCSP responder requires a signed request.

| [CMSERR_OCSP_EXPIRED]
| The OCSP response has expired.

| [CMSERR_PATH_TOO_LONG]
| The certification chain exceeds the maximum allowed by the CA.

| [CMSERR_REQUIRED_BC_EXT_MISSING]
| The required basic constraints certificate extension is missing.

| [CMSERR_REVINFO_NOT_YET_VALID]
| Revocation information is not yet valid.

| [CMSERR_SELF_SIGNED_NOT_FOUND]
| A self-signed certificate is not found in a trusted data source

| [CMSERR_UNKNOWN_ERROR]
| An unknown error has occurred.

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 19 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 if 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 the root certificate for the chain is found in one of the trusted data sources. If a sole intermediate certificate is found in a SAF key ring and the next issuer is not found in the same SAF key ring, the intermediate certificate will be allowed to act as a trust anchor and the chain will be considered complete. It is strongly recommended that a SAF key ring containing an intermediate certificate also has the rest of the certificate chain connected to the key ring, including the root certificate.

The data sources must contain at least one of the following sources to check for revoked certificates: LDAP directory, LDAP extended directory, HTTP CDP, OCSP, or a CRL source. The LDAP, HTTP CDP, and OCSP data sources are untrusted. A CRL data source can be either trusted or untrusted.

If using an LDAP directory or LDAP extended directory source, 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 is returned if a CRL obtained from an untrusted source cannot be validated.

If using an HTTP CDP data source, the CRL distribution point name indicates a Universal Resource Indicator (URI), which indicates the HTTP server to contact for the CRL.

If using an OCSP data source, the AIA extension in the certificate or the *ocsp_url* parameter specified in the *gskdb_ocsp_source* structure specifies the URI of the OCSP responder to contact.

gsk_validate_certificate()

An LDAP extended directory, HTTP CDP, and OCSP data source is created by calling the **gsk_create_revocation_source()** routine and filling in the *gskdb_source* structure and the embedded specific data source structure. See **gsk_create_revocation_source()** for more information about the creating the data sources.

Security levels for connecting to LDAP directories are based on the GSKCMS_CRL_SECURITY_LEVEL setting. When using the **gsk_open_directory ()** routine, the GSKCMS_CRL_SECURITY_LEVEL setting can be specified using the **gsk_set_directory_enum()** routine. When using the **gsk_create_revocation_source()** routine to create an extended LDAP directory source, the security setting can be set in the *crlSecurityLevel* parameter in the *gskdb_extended_directory_source* structure within the *gskdb_source* structure. Security levels can be set to LOW, MEDIUM or HIGH. See “*gsk_attribute_set_enum()*” on page 92 for more information about CRL security level settings. See “*gsk_create_revocation_source()*” on page 216 for more information about the creation of the extended LDAP directory data source.

By default, the revocation security level for OCSP and HTTP CDP revocation sources is set to GSKCMS_REVOCATION_SECURITY_LEVEL_MEDIUM. See “*gsk_validate_certificate_mode()*” on page 464 for information on the *security_level*.

By default, the maximum number of locations that will be contacted per data source when attempting validation of a certificate containing AIA or CDP extensions is 10. See “*gsk_validate_certificate_mode()*” on page 464 for information on the *max_source_rev_ext_loc_values*.

By default, the maximum number of HTTP URI values that will be contacted when performing validation of a certificate chain containing AIA or CDP extensions is 10. See “*gsk_validate_certificate_mode()*” on page 464 for information on the *max_validation_rev_ext_loc_values*.

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. See the **gsk_get_directory_certificates()** and **gsk_get_directory_crls()** routines for more information concerning the use of LDAP directory entries.
- *gskdb_source_directory_extended* - The source is an LDAP directory. The handle must be the directory handle returned by the **gsk_create_revocation_source()** routine. This is an untrusted data source. Any certificate or revocation list obtained from this source is validated before being accepted. See the **gsk_get_directory_certificates()** and **gsk_get_directory_crls()** routines for more information concerning the use of LDAP directory entries.
- *gskdb_source_ocsp* - The source is an OCSP responder. The handle must be the handle returned by the **gsk_create_revocation_source()** routine. This is an untrusted data source. The revocation information obtained from this source is validated before being accepted.

|
|
|
|

- `gskdb_source_cdp` - The source is an CDP. The handle must be the handle returned by the `gsk_create_revocation_source()` routine. This is an untrusted data source. The CRL obtained from this source is validated before being accepted.
- `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. This is a trusted source. The return value should be 0 if the certificate is not revoked. If the callback routine is unable to check the certificate for revocation and processing should continue to the next data source, the return value should be -1. Otherwise it should be one of the error codes defined in the `gskcms.h` include file.

`gsk_validate_certificate_mode()`

`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  
    [,GSKCMS_CERT_VALIDATE_KEYRING_ROOT validate_root]  
    [,GSKCMS_REVOCAATION_SECURITY_LEVEL security_level]  
    [,gsk_int32                   max_source_rev_ext_loc_values]  
    [,gsk_int32                   max_validation_rev_ext_loc_values]  
    ...)
```

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 to be accepted.

issuer_record_id

Returns the record identifier for the issuer certificate that is used to validate the certificate. The record identifier is 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 that is used for certificate validation.

arg_count

Specifies the number of optional parameters following the *arg_count* parameter. The *arg_count* parameter can be set to either 0, 1, 2, 3, or 4. If set to 1, the *validate_root* parameter must be specified. If set to 2, the *validate_root* and *security_level* parameters must be specified. If set to 3, the *validate_root*, *security_level*, and *max_source_rev_ext_loc_values* parameters must be specified. If set to 4, the *validate_root*, *security_level*, *max_source_rev_ext_loc_values*, and *max_validation_rev_ext_loc_values* parameters must be specified.

validate_root

Specifies how certificates in a SAF key ring are validated. Specify `GSKCMS_CERT_VALIDATE_KEYRING_ROOT_ON` if SAF key ring certificates are validated to the root CA certificate. Specify `GSKCMS_CERT_VALIDATE_KEYRING_ROOT_OFF` if SAF key ring certificates are validated only to the trust anchor certificate when a sole intermediate

certificate exists in the SAF key ring. By default, SAF key ring certificates are only validated to the trust anchor certificate. This setting does not affect the validation of SSL key database file and PKCS #11 token certificates as these certificates are always validated to the root CA certificate.

security_level

Specifies the behavior when an OCSP responder or an HTTP server can not be contacted. Specify `GSKCMS_REVOCATION_SECURITY_LEVEL_LOW` if certificate validation does not fail if the OCSP responder or HTTP server cannot be contacted. Specify `GSKCMS_REVOCATION_SECURITY_LEVEL_MEDIUM` if certificate validation requires the OCSP responder or the HTTP server in a URI value in the CDP extension to be contactable. For an OCSP responder, it must be able to provide a good certificate revocation status. If the certificate status is revoked or unknown, certificate validation fails. For an HTTP server in a CDP extension, it must be contactable and able to provide an CRL that is encoded in a valid manner. If both an OCSP and a CDP data source are specified, both are attempted to be contacted. If they both fail, certificate validation fails. This parameter is only used when the *data_sources* parameter includes at least one CDP or OCSP data source created by `gsk_create_revocation_source()`. Specify `GSKCMS_REVOCATION_SECURITY_LEVEL_HIGH` if certificate validation requires revocation information to be provided by the OCSP responder or HTTP server.

If the *security_level* parameter is not specified, `GSKCMS_REVOCATION_SECURITY_LEVEL_MEDIUM` is used.

Note: The security level for LDAP data sources is set within the data source itself.

max_source_rev_ext_loc_values

Specifies the maximum number of locations that will be contacted per data source when attempting validation of a certificate. The locations for revocation information are specified by the `accessLocation` in the AIA certificate extension for OCSP and the `distributionPoint` in the CDP extension for HTTP CRLs. When an HTTP URI is present in an AIA or CDP extension, an attempt is made to contact the remote HTTP server to obtain revocation information. Both of these extensions can contain multiple location values and therefore have the potential to negatively impact performance when there are a very large number of locations present.

The valid values are 0 through 256. A value of 0 indicates there is no limit.

If the *max_source_rev_ext_loc_values* parameter is not specified, a value of 10 is used.

max_validation_rev_ext_loc_values

Specifies the maximum number of HTTP URI values that will be contacted when performing validation of a certificate chain. The locations for revocation information are specified by the `accessLocation` in the AIA certificate extension for OCSP and the `distributionPoint` in the CDP extension for HTTP CRLs. When an HTTP URI is present in an AIA or CDP extension, an attempt is made to contact the remote HTTP server to obtain revocation information. Both of these extensions can contain multiple location values and therefore have the potential to negatively impact performance when there are a very large number of locations present.

The valid values are 0 through 1024. A value of 0 indicates there is no limit.

gsk_validate_certificate_mode()

| If the *max_validation_rev_ext_loc_values* parameter is not specified, a value of
| 100 is used.

Results

The return status is 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.

[CMSERR_BAD_HTTP_RESPONSE]

HTTP response 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_OCSP_RESPONSE]

OCSP response is not valid.

[CMSERR_BAD_SECURITY_LEVEL_ARG]

Variable argument security level is not valid.

[CMSERR_BAD_SIGNATURE]

The signature is not correct.

[CMSERR_BAD_SUBJECT_NAME]

Subject name is not valid.

[CMSERR_BAD_VALIDATE_ROOT_ARG]

Variable argument validate root 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_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_EXPIRED]

The certificate is expired.

[CMSERR_EXT_NOT_SUPPORTED]

Certificate extension is not supported.

[CMSERR_HOSTNAME_NOT_VALID]

HTTP server hostname is not valid.

[CMSERR_HTTP_IO_ERROR]

HTTP server communication error.

[CMSERR_HTTP_RESPONSE_TIMEOUT]

HTTP response not received within the configured time limit.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]

The database type does not support certificates.

[CMSERR_INCORRECT_KEY_USAGE]

The issuer certificate does not allow signing certificates

[CMSERR_INTERNAL_ERROR]

An internal processing error has occurred.

[CMSERR_INVALID_NUMERIC_VALUE]

The variable argument for *max_source_rev_ext_loc_values* or *max_validation_rev_ext_loc_values* is not valid.

[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_LDAP_RESPONSE_TIMEOUT]

LDAP response not received within configured time limit.

[CMSERR_MAX_RESPONSE_SIZE_EXCEEDED]

The maximum response size has been exceeded.

[CMSERR_MAX_REV_EXT_LOC_VALUES_EXCEEDED]

The maximum number of locations allowed to be contacted has been reached.

gsk_validate_certificate_mode()

[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_NO_ACCEPTABLE_POLICIES]

Acceptable policy intersection cannot be found.

[CMSERR_NO_MEMORY]

Insufficient storage is available.

[CMSERR_NOT_YET_VALID]

The certificate is not yet valid.

[CMSERR_OCSP_NONCE_CHECK_FAILED]

Nonce in OCSP response does not match value in OCSP request.

[CMSERR_OCSP_RESPONDER_ERROR]

OCSP request failed with internal responder error.

[CMSERR_OCSP_RESPONSE_TIMEOUT]

OCSP response was not received within the configured time limit.

[CMSERR_OCSP_REQ_ERROR]

Error creating the OCSP request.

[CMSERR_OCSP_SIG_REQUIRED]

OCSP responder requires a signed request.

[CMSERR_OCSP_EXPIRED]

The OCSP response has expired.

[CMSERR_PATH_TOO_LONG]

The certification chain exceeds the maximum that is allowed by the CA.

[CMSERR_RECORD_NOT_FOUND]

Record not found.

[CMSERR_REQUIRED_BC_EXT_MISSING]

The required basic constraints certificate extension is missing.

[CMSERR_REQUIRED_EXT_MISSING]

Required certificate extension is missing.

[CMSERR_REVINFO_NOT_YET_VALID]

Revocation information is not yet valid.

[CMSERR_UNKNOWN_ERROR]

An unknown error has occurred.

Usage

The **gsk_validate_certificate_mode()** routine validates an X.509 certificate according to the standards defined in RFC 2459: *Internet x.509 Public Key Infrastructure Certificate and CRL Profile*, RFC 3280: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, or RFC 5280: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. Any necessary CA or issuer certificates are obtained from the supplied data sources. The CA certificate is also validated according to the previously mentioned Internet standards.

The *validation_mode* parameter determines the Internet standard that the certificate and certificate chain are validated against. The following validation modes are supported:

- GSKCMS_CERT_VALIDATION_MODE_2459 – validate the certificate against RFC 2459 only.
- GSKCMS_CERT_VALIDATION_MODE_3280 – validate the certificate against RFC 3280 only.
- GSKCMS_CERT_VALIDATION_MODE_5280 – validate the certificate against RFC 5280 only.
- GSKCMS_CERT_VALIDATION_MODE_ANY – attempt to validate the certificate against RFC 2459 initially. If that fails, validate against RFC 3280. If that fails, validate against RFC 5280.

Note: The z/OS specific HostIDMapping certificate extension is supported by System SSL and can be validated as a critical extension in any validation mode.

A root certificate is a self-signed certificate and its signature is verified by using the public key in the certificate. If *accept_root* is FALSE, the root certificate must be found in a trusted data source to be accepted. If *accept_root* is TRUE, the self-signed certificate is accepted if the signature is correct.

An intermediate certificate or an end-entity certificate is a certificate that is signed by another entity. Its signature is verified by 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 the root certificate for the chain is found in one of the trusted data sources. If a sole intermediate certificate is found in a SAF key ring and the next issuer is not found in the same SAF key ring, and *validate_root* is not specified or is set to GSKCMS_CERT_VALIDATE_KEYRING_ROOT_OFF, the intermediate certificate is allowed to act as a trust anchor, and the chain is considered complete. By default, SAF key ring certificates are only validated to the trust anchor certificate. If *validate_root* is set to GSKCMS_CERT_VALIDATE_KEYRING_ROOT_ON, an intermediate certificate in a SAF key ring is not allowed to be established as a trust anchor and full certificate validation to the root CA must occur. Make sure that a SAF key ring containing an intermediate certificate also has the rest of the certificate chain that is connected to the key ring, including the root certificate. The *validate_root* setting does not affect the validation of SSL key database file and PKCS #11 token certificates because these certificates are always validated to the root CA certificate.

| The data sources must contain at least one of the following sources to check for
| revoked certificates: LDAP directory, LDAP extended directory, HTTP CDP, OCSP,
| or a CRL source. The LDAP, HTTP CDP, and OCSP data sources are untrusted. A
| CRL data source can be either trusted or untrusted.

| If using an LDAP directory or LDAP extended directory source, 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 is returned if a CRL obtained from an untrusted source cannot be validated.

gsk_validate_certificate_mode()

If using an HTTP CDP data source, the CRL distribution point name indicates a Universal Resource Indicator (URI), which indicates the HTTP server to contact for the CRL.

If using an OCSP data source, the AIA extension in the certificate or the *ocsp_url* parameter specified in the *gskdb_ocsp_source* structure specifies the URI of the OCSP responder to contact.

An LDAP extended directory, HTTP CDP, and OCSP data source is created by calling the **gsk_create_revocation_source()** routine and filling in the *gskdb_source* structure and the embedded specific data source structure. See **gsk_create_revocation_source()** for more information about the creating the data sources.

Security levels for connecting to LDAP directories are based on the GSKCMS_CRL_SECURITY_LEVEL setting. When using the **gsk_open_directory()** routine, the GSKCMS_CRL_SECURITY_LEVEL setting can be specified using the **gsk_set_directory_enum()** routine. When using the **gsk_create_revocation_source()** routine to create an extended LDAP directory source, the security setting can be set in the *crlSecurityLevel* parameter in the *gskdb_extended_directory_source* structure within the *gskdb_source* structure. Security levels can be set to LOW, MEDIUM or HIGH. See “**gsk_attribute_set_enum()**” on page 92 for more information about CRL security level settings. See “**gsk_create_revocation_source()**” on page 216 for more information about the creation of the extended LDAP directory data source.

These data sources are supported:

- *gskdb_source_key_database* - The source is a key database. The handle must be a database handle that is 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 that is returned by the **gsk_open_directory()** routine. This is an untrusted data source. Any certificate or revocation list that is obtained from this source is validated before it is accepted. See the **gsk_get_directory_certificates()** and **gsk_get_directory_crls()** routines for more information about the use of LDAP directory entries.
- *gskdb_source_directory_extended* - The source is an LDAP directory. The handle must be the directory handle returned by the **gsk_create_revocation_source()** routine. This is an untrusted data source. Any certificate or revocation list obtained from this source is validated before being accepted. See the **gsk_get_directory_certificates()** and **gsk_get_directory_crls()** routines for more information concerning the use of LDAP directory entries.
- *gskdb_source_ocsp* - The source is an OCSP responder. The handle must be the handle returned by the **gsk_create_revocation_source()** routine. This is an untrusted data source. The revocation information obtained from this source is validated before being accepted.
- *gskdb_source_cdp* - The source is an CDP. The handle must be the handle returned by the **gsk_create_revocation_source()** routine. This is an untrusted data source. The CRL obtained from this source is validated before being accepted.
- *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 that is used from this list is validated before it is 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 is validated before it is accepted.
- `gskdb_source_cert_callback` - The source is the address of a callback routine that receives 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 calls 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 codes that are 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 that receives control when a certificate must be checked to see if it has been revoked. This is a trusted source. The return value should be 0 if the certificate is not revoked. If the callback routine is unable to check the certificate for revocation and processing should continue to the next data source, the return value should be -1. Otherwise, it should be one of the error codes that are defined in the **`gskcms.h`** include file.

| The *validate_root* optional parameter must be specified when *arg_count* is set to 1 or
| greater. If *validate_root* is not specified and *arg_count* is set to 1 or greater, an error
| of `CMSERR_BAD_VALIDATE_ROOT_ARG` is returned.

| The *security_level* parameter must be specified when *arg_count* is set to 2 or greater.
| If *security_level* is not specified and *arg_count* is set to 2 or greater, an error of
| `CMSERR_BAD_SECURITY_LEVEL_ARG` is returned.

| The *max_source_rev_ext_loc_values* parameter must be specified when *arg_count* is
| set to 3 or greater. If *max_source_rev_ext_loc_values* is not specified and *arg_count* is
| set to 3 or greater, an error of `CMSERR_INVALID_NUMERIC_VALUE` is returned.

| The *max_validation_rev_ext_loc_values* parameter must be specified when *arg_count*
| is set to 4. If *max_validation_rev_ext_loc_values* is not specified and *arg_count* is set to
| 4, an error of `CMSERR_INVALID_NUMERIC_VALUE` is returned.

If the *arg_count* parameter is 0, any additional parameters that are specified are ignored.

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 19 for more details.

`gsk_validate_extended_key_usage()`

Validate a certificate's extended key usage extension against the supplied extended key usage list.

Format

```
#include <gskcms.h>
```

```
gsk_status gsk_validate_extended_key_usage (
    x509_certificate *      certificate,
    x509_key_usages *      key_usages,
    GSKCMS_VALIDATE_EXTENDED_KEY_USAGE validate_option)
```

Parameters

certificate

Specifies the x.509 certificate to be validated.

key_usages

Specifies a list of x.509 extended key usage purpose types. The count must be at least one. The key purpose type OID field is ignored unless the usage purpose type is set to

GSKCMS_VALIDATE_EXTENDED_KEY_USAGE_CHECK_OID.

validate_option

Specifies the validation option to customize the validation process:

GSKCMS_VALIDATE_EXTENDED_KEY_USAGE_ALL

All the specified extended key purpose types must be present in the certificate's extended key usage extension.

GSKCMS_VALIDATE_EXTENDED_KEY_USAGE_SOME

At least one of the specified extended key purpose types must be present within the certificate's extended key usage extension.

GSKCMS_VALIDATE_EXTENDED_KEY_USAGE_NONE

None of the specified extended key purposes may be present within the certificate's extended key usage extension.

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_VALIDATION_OPTION]

Validation option is not valid.

[CMSERR_CERTIFICATE_NOT_SUPPLIED]

Input certificate is missing.

[CMSERR_CERT_HAS_NO_EXT_KEY_USAGE_EXTENSION]

Certificate does not have an extended key usage extension.

[CMSERR_EXT_KEY_USAGE_COMPARE_FAILED]

Extended key usage comparison failed.

[CMSERR_EXT_KEY_USAGE_COUNT_IS_INVALID]

Extended key usage count is zero.

[CMSERR_EXT_KEY_USAGE_NOT_SUPPLIED]

Extended key usage or key usage purpose types not provided.

| [CMSERR_EXT_KEY_USAGE_TYPE_IS_INVALID]

| Extended key usage type is not supported for this operation.

| **Usage**

| The **gsk_validate_extended_key_usage()** routine compares the provided
| certificate's extended key usage extension values against the provided list of
| extended key purposes. The extended key usage comparison scope is defined by
| the **GSKCMS_VALIDATE_EXTENDED_KEY_USAGE** value.

| If the input certificate does not contain an extended key usage extension, the
| operation returns **CMSERR_CERT_HAS_NO_EXT_KEY_USAGE_EXTENSION**.

| If the *x509_purpose_type* is set to
| **GSKCMS_VALIDATE_EXTENDED_KEY_USAGE_CHECK_OID**, the *gsk_oid* must
| be supplied. In this case, the user-supplied OID will be compared to the certificate
| extended key usage extension value OID. Use this method in instances where a
| certificate's extended key usage extension type and OID are not defined by System
| SSL *x509_purpose_type*.

| The *x509_purpose_type* types are only checked for presence or absence when being
| compared to a certificate's extended key usage.

`gsk_validate_hostname()`

Validates a host certificate against the supplied hostname.

Format

```
#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. A case-sensitive (exact match) comparison is used for comparison with the common name (CN) element of the subject name when the common name attribute value is encoded as UTF-8 data (`x509_string_utf8`).

The *val_option* parameter determines the composition and order of the validation process. A value of:

- `GSKCMS_VALIDATE_HOSTNAME_CN` validates 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` validates the host name against the common name (CN) of the certificate only.
- `GSKCMS_VALIDATE_HOSTNAME_DNS` validates 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` validates the host name against the DNS entry in the subject alternate name extension only.

gsk_validate_hostname()

The host name in the 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-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 is 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 must be validated.

`gsk_validate_server()`

`gsk_validate_server()`

Validate a server certificate.

Format

```
#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 is 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 that is 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. A case-sensitive (exact match) comparison is used for comparison with the common name (CN) element of the subject name when the common name attribute value is encoded as UTF-8 data (`x509_string_utf8`). For other combinations of host name 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 not case-sensitive 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 is ignored (for example, 'dcesec4.endicott.ibm.com.' and 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 must 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_EC_PARAMS]

Elliptic Curve parameters are not valid.

[CMSERR_BAD_KEY_SIZE]

The key size is not valid.

[CMSERR_BAD_SIGNATURE]

The signature is not correct.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[CMSERR_KEY_MISMATCH]

The supplied key does not match the signature algorithm.

[CMSERR_PRIVATE_KEY_INFO_NOT_SUPPLIED]

Private key information not supplied.

gsk_verify_certificate_signature()

[CMSERR_SIGNATURE_NOT_SUPPLIED]

Signature not supplied.

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}

x509_alg_dsaWithSha1

Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_dsaWithSha224

Digital Signature Standard with SHA-224 digest - {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256

Digital Signature Standard with SHA-256 digest - {2.16.840.1.101.3.4.3.2}

x509_alg_md5Sha1WithRsaEncryption

RSA encryption with combined MD5 and SHA-1 digests

x509_alg_ecdsaWithSha1

Elliptic Curve Digital Signature Algorithm with SHA-1 digest -
{1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224

Elliptic Curve Digital Signature Algorithm with SHA-224 digest -
{1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256

Elliptic Curve Digital Signature Algorithm with SHA-256 digest -
{1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384

Elliptic Curve Digital Signature Algorithm with SHA-384 digest -
{1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512

Elliptic Curve Digital Signature Algorithm with SHA-512 digest -
{1.2.840.10045.4.3.4}

gsk_verify_certificate_signature()

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

```
#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_EC_PARAMS]

Elliptic Curve parameters are not valid.

[CMSERR_BAD_KEY_SIZE]

The key size is not valid.

[CMSERR_BAD_SIGNATURE]

The signature is not correct.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[CMSERR_KEY_MISMATCH]

The supplied key does not match the signature algorithm.

[CMSERR_PRIVATE_KEY_INFO_NOT_SUPPLIED]

Private key information not supplied.

[CMSERR_SIGNATURE_NOT_SUPPLIED]

Signature not supplied.

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}

x509_alg_dsaWithSha1

Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_dsaWithSha224

Digital Signature Standard with SHA-224 digest - {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256

Digital Signature Standard with SHA-256 digest - {2.16.840.1.101.3.4.3.2}

x509_alg_md5Sha1WithRsaEncryption

RSA encryption with combined MD5 and SHA-1 digests

x509_alg_ecdsaWithSha1

Elliptic Curve Digital Signature Algorithm with SHA-1 digest -
{1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224

Elliptic Curve Digital Signature Algorithm with SHA-224 digest -
{1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256

Elliptic Curve Digital Signature Algorithm with SHA-256 digest -
{1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384

Elliptic Curve Digital Signature Algorithm with SHA-384 digest -
{1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512

Elliptic Curve Digital Signature Algorithm with SHA-512 digest -
{1.2.840.10045.4.3.4}

gsk_verify_crl_signature()

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      sign_algorithm,
    x509_public_key_info *  key,
    gsk_boolean             is_digest,
    gsk_buffer *            data,
    gsk_buffer *            signature)
```

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_EC_PARAMS]

Elliptic Curve parameters are not valid.

[CMSERR_BAD_KEY_SIZE]

The key size is not valid.

[CMSERR_BAD_SIGNATURE]

The signature is not correct.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]

Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]

Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

gsk_verify_data_signature()

[CMSERR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[CMSERR_KEY_MISMATCH]

The supplied key does not match the signature algorithm.

[CMSERR_PRIVATE_KEY_INFO_NOT_SUPPLIED]

Private key information not supplied.

[CMSERR_SIGNATURE_NOT_SUPPLIED]

Signature not supplied.

Usage

The **gsk_verify_data_signature()** routine validates the signature for a data stream. The public key can be an RSA key, a DSA key, or an ECDSA 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; 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_dsaWithSha224

Digital Signature Standard with SHA-224 digest - {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256

Digital Signature Standard with SHA-256 digest – {2.16.840.1.101.3.4.3.2}

x509_alg_md5Sha1WithRsaEncryption

RSA encryption with combined MD5 and SHA-1 digests

x509_alg_ecdsaWithSha1

Elliptic Curve Digital Signature Algorithm with SHA-1 digest –
{1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224

Elliptic Curve Digital Signature Algorithm with SHA-224 digest –
{1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256

Elliptic Curve Digital Signature Algorithm with SHA-256 digest –
{1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384

Elliptic Curve Digital Signature Algorithm with SHA-384 digest –
{1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512

Elliptic Curve Digital Signature Algorithm with SHA-512 digest –
{1.2.840.10045.4.3.4}

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.

gsk_verify_data_signature()

Chapter 9. Deprecated Secure Socket Layer (SSL) APIs

These application programming interfaces, or APIs, are superseded by the APIs defined in Chapter 7, “API reference,” on page 57.

- `gsk_free_memory()` (see “`gsk_free_memory()`” on page 488)
- `gsk_get_cipher_info()` (see “`gsk_get_cipher_info()`” on page 489)
- `gsk_get_dn_by_label()` (see “`gsk_get_dn_by_label()`” on page 490)
- `gsk_initialize()` (see “`gsk_initialize()`” on page 491)
- `gsk_secure_soc_close()` (see “`gsk_secure_soc_close()`” on page 496)
- `gsk_secure_soc_init()` (see “`gsk_secure_soc_init()`” on page 497)
- `gsk_secure_soc_read()` (see “`gsk_secure_soc_read()`” on page 505)
- `gsk_secure_soc_reset()` (see “`gsk_secure_soc_reset()`” on page 508)
- `gsk_secure_soc_write()` (see “`gsk_secure_soc_write()`” on page 509)
- `gsk_srb_initialize()` (see “`gsk_srb_initialize()`” on page 511)
- `GSKSRBRD()` (see “`GSKSRBRD`” on page 513)
- `GSKSRBWT()` (see “`GSKSRBWT`” on page 514)
- `gsk_uninitialize()` (see “`gsk_uninitialize()`” on page 515)
- `gsk_user_set()` (see “`gsk_user_set()`” on page 516)

Although use of the deprecated set of APIs in this topic is still supported in z/OS Version 2 Release 2, make sure that new applications be developed using the set of APIs defined in Chapter 7, “API reference,” on page 57.

The deprecated APIs are not being explicitly updated to allow utilization of new functionality to be added to System SSL. If an application wants to use new functionality to be added, for example TLS V1.2 protocol, the application must be coded to the SSL APIs in Chapter 7, “API reference,” on page 57.

In addition, make sure that existing applications are modified to use the set of APIs defined in Chapter 7, “API reference,” on page 57. 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 55.

gsk_free_memory()

gsk_free_memory()

Releases storage allocated by the SSL run time.

Format

```
#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 run time.

Related Topics

[“gsk_get_dn_by_label\(\)” on page 490](#)

gsk_get_cipher_info()

Returns the supported cipher specifications.

Format

```
#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 United States only cipher specifications including 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 United States only 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 "050435363738392F303132330A1613100D0915120F0C0306020100" if not in FIPS mode, and "35363738392F303132330A1613100D" 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_initialize\(\)” on page 491](#)

[“gsk_secure_soc_init\(\)” on page 497](#)

`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_free_memory\(\)” on page 488](#)

[“gsk_initialize\(\)” on page 491](#)

[“gsk_secure_soc_init\(\)” on page 497](#)

gsk_initialize()

Initializes the System SSL runtime environment.

Format

```
#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_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[GSK_ERROR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[GSK_ERROR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[GSK_ERROR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

[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, PKCS #12 file, 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_initialize()

[GSK_KEYFILE_BAD_FORMAT]

Key database, PKCS #12 file, or key ring format is not valid.

[GSK_KEYFILE_BAD_PASSWORD]

Key database or PKCS #12 file password is not correct.

[GSK_KEYFILE_IO_ERROR]

Unable to read the key database, PKCS #12 file, key ring or token.

[GSK_KEYFILE_NO_CERTIFICATES]

The key database, PKCS #12 file, key ring or token does not contain any certificates.

[GSK_KEYFILE_OPEN_FAILED]

Unable to open the key database, PKCS #12 file, 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()** causes 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 (SSL V2, SSL V3, and TLS V1.0).

When "SSLV2_OFF" is specified the SSL client/server attempts 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 attempts first to use the TLS V1.0 protocol and falls back to the most secure protocol that the server supports, 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 accepts 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, PKCS #12 file, 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 user ID is used if the user ID 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, except for SITE certificates where CONTROL authority is given to IRR.DIGTCERT.GENCERT in the FACILITY class or for user certificates where READ or UPDATE authority is given to *ringOwner.ringName.LST* resource in the RDATALIB class.

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 user ID 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 or PKCS #12 file as a null-terminated character string. Specify NULL to indicate that no password is provided.

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

gsk_initialize()

server and is a null-terminated character string. An anonymous bind is 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 are 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`.

gsk_initialize() supported 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, PKCS #12 file, key ring, or token is read.

The **gsk_initialize()** routine supports the following environment variables (See Appendix A, "Environment variables," on page 667 for information about using the environment variables):

GSK_CERT_VALIDATE_KEYRING_ROOT

Specifies the setting of how certificates in a SAF key ring are validated.

GSK_CERT_VALIDATION_MODE

Specifies which internet standard is used for certificate validation.

GSK_CLIENT_AUTH_NOCERT_ALERT

Specifies whether the SSL server application accepts a connection from a client where client authentication is requested and the client fails to supply an X.509 certificate.

GSK_CRL_CACHE_TIMEOUT

Specifies the number of hours that a cached LDAP CRL remains valid.

GSK_CRL_SECURITY_LEVEL

Specifies the level of security to be used when contacting LDAP servers to check CRLs for revoked certificates.

GSK_EXTENDED_RENEGOTIATION_INDICATOR

Specifies the level of enforcement of renegotiation indication as specified by RFC 5746 during the initial handshake.

GSK_KEY_LABEL

Specifies the label of the key that is used to authenticate the application.

GSK_RENEGOTIATION

Specifies the type of session renegotiation that is allowed for an SSL environment.

GSK_RENEGOTIATION_PEER_CERT_CHECK

Specifies if the peer certificate is allowed to change during renegotiation.

GSK_SYSPLEX_SIDCACHE

Specifies whether sysplex session caching is supported.

GSK_TLS_CBC_PROTECTION_METHOD

Specifies an optional SSL V3.0 or TLS V1.0 CBC IV protection method when writing application data.

GSKV2CACHESIZE

Specifies the number of entries in the SSL V2 session cache. The value that is specified by the `GSK_V2_SIDCACHE_SIZE` environment variable is used if the `GSKV2CACHESIZE` variable is not defined.

GSK_V2_CIPHER_SPECS

Specifies the SSL V2 cipher specifications in order of preference.

GSKV3CACHESIZE

Specifies the number of entries in the SSL V3 session cache. The value that is specified by the `GSK_V3_SIDCACHE_SIZE` environment variable is used if the `GSKV3CACHESIZE` variable is not defined.

GSK_V3_CIPHER_SPECS

Specifies the SSL V3 cipher specifications in order of preference (2-character values).

Related Topics

["gsk_secure_soc_close\(\)" on page 496](#)

["gsk_secure_soc_init\(\)" on page 497](#)

["gsk_secure_soc_read\(\)" on page 505](#)

["gsk_secure_soc_write\(\)" on page 509](#)

["gsk_uninitialize\(\)" on page 515](#)

gsk_secure_soc_close()

Closes a secure socket connection.

Format

```
#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()" on page 491

"gsk_secure_soc_init()" on page 497

"gsk_secure_soc_read()" on page 505

"gsk_secure_soc_write()" on page 509

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 is 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_ICSF_CLEAR_KEY_SUPPORT_NOT_AVAILABLE]

ICSF clear key support not available.

[GSK_ERROR_ICSF_FIPS_DISABLED]

ICSF PKCS #11 services are disabled.

[GSK_ERROR_ICSF_NOT_AVAILABLE]

ICSF services are not available.

[GSK_ERROR_ICSF_NOT_FIPS]

ICSF PKCS #11 not operating in FIPS mode.

[GSK_ERROR_ICSF_SERVICE_FAILURE]

ICSF callable service returned an error.

gsk_secure_soc_init()

[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.

[GSK_ERROR_NO_PRIVATE_KEY]

Certificate does not contain a private key or the private key is unusable.

[GSK_ERROR_REQ_CERT_BC_EXT_MISSING]

The required basic constraints certificate extension is missing.

[GSK_ERROR_RNG]

Error encountered when generating random bytes.

[GSK_ERROR_SELF_SIGNED]

A self-signed certificate cannot be validated.

[GSK_ERROR_SOCKET_CLOSED]

Socket connection closed by peer application.

[GSK_ERROR_UNKNOWN_CA]

A certification authority certificate is missing.

[GSK_ERROR_UNSUPPORTED_CERTIFICATE_TYPE]

The certificate type is not supported by System SSL.

[GSK_ERROR_VALIDATION]

Certificate validation error.

[GSK_KEYFILE_BAD_DNAME]

The specified key is not found in the key database or the key is not trusted.

[GSK_KEYFILE_BAD_LABEL]

The DName field of the gsk_soc_init_data structure is an empty string. If the default key is to be used, the DName field must be NULL.

[GSK_KEYFILE_DUPLICATE_NAME]

The key database contains multiple certificates with the same subject name as the distinguished name specified in the connection initialization data.

[GSK_SOC_BAD_V2_CIPHER]

SSL V2 cipher is not valid.

[GSK_SOC_BAD_V3_CIPHER]

SSL/TLS V3 cipher is not valid.

[GSK_SOC_NO_READ_FUNCTION]

No read function is specified in the connection initialization data.

[GSK_SOC_NO_WRITE_FUNCTION]

No write function is specified in the connection initialization data.

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, DSA, or Diffie-Hellman as the public/private key algorithm. In FIPS mode, the RSA or DSA key size must be at least 1024 bits and the Diffie-Hellman key size must be 2048 bits. An RSA certificate can be used with an RSA or ephemeral Diffie-Hellman key exchange. A DSA certificate can be used with an ephemeral Diffie-Hellman key exchange. A Diffie-Hellman certificate can be used in a fixed Diffie-Hellman key exchange. 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.
- Diffie-Hellman certificates using fixed Diffie-Hellman key exchange must allow key agreement.
- RSA or DSA certificates using ephemeral Diffie-Hellman key exchange must allow digital signature.
- Other RSA certificates must allow key encipherment.

System SSL does not accept Verisign Global Server ID certificates. When specified, System SSL uses 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 that is stored in the connection is used to retrieve the certificate from the key database, PKCS #12 file, 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.

gsk_secure_soc_init()

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:

- "0A" = 168-bit Triple DES encryption with SHA-1 message authentication and RSA key exchange
- "0D" = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate
- "10" = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "13" = 168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA 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 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

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 servers cipher list for a match in the clients list. The first matching cipher is used.

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, PKCS #12 file, key ring or token is read.

The environment variables that are 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 intended handshake type as follows:

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

gsk_secure_soc_init()

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 ("7364" if United States only encryption is enabled (System SSL Security Level 3 FMID is installed) and "64" otherwise). Valid cipher specifications that are not supported because of 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 ("35363738392F303132330A1613100D0915120F0C" if United States only encryption is enabled (System SSL Security Level 3 FMID is installed) and in non-FIPS mode, "35363738392F303132330A1613100D" if United States only encryption is enabled (System SSL Security Level 3 FMID is installed) and in FIPS mode, and "0915120F0C" 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 because of the installed cryptographic level are 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 87 for additional information about 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 87 for additional information about 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
- "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

gsk_secure_soc_init()

- "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.

Related Topics

[“gsk_get_cipher_info\(\)” on page 489](#)

[“gsk_get_dn_by_label\(\)” on page 490](#)

[“gsk_initialize\(\)” on page 491](#)

[“gsk_secure_soc_close\(\)” on page 496](#)

[“gsk_secure_soc_read\(\)” on page 505](#)

[“gsk_secure_soc_reset\(\)” on page 508](#)

[“gsk_secure_soc_write\(\)” on page 509](#)

gsk_secure_soc_read()

Reads data using a secure socket connection.

Format

```
#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_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_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_NO_NEGOTIATION]

An attempt was made to renegotiate a session when renegotiation is disabled or the peer rejected an attempted session renegotiation.

gsk_secure_soc_read()

[GSK_ERROR_RENEGOTIATION_INDICATION]

Peer did not signal support for TLS Renegotiation Indication.

[GSK_ERROR_SOCKET_CLOSED]

Socket connection closed by peer application.

[GSK_ERROR_WOULD_BLOCK]

A complete SSL record is not available.

[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()** if 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()" on page 491

"gsk_secure_soc_close()" on page 496

"gsk_secure_soc_init()" on page 497

"gsk_secure_soc_write()" on page 509

`gsk_secure_soc_reset()`

Resets the session keys for a secure connection.

Format

```
#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_ERR_NO_NEGOTIATION]

An attempt was made to renegotiate a session when renegotiation is disabled.

[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()`” on page 497

gsk_secure_soc_write()

Writes data using a secure socket connection.

Format

```
#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 because of 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

gsk_secure_soc_write()

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()** if 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()" on page 491

"gsk_secure_soc_close()" on page 496

"gsk_secure_soc_init()" on page 497

"gsk_secure_soc_read()" on page 505

gsk_srb_initialize()

Initializes SRB support.

Format

```
#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 **GSKSRBRD** and **GSKSRBWT** 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.

See *z/OS MVS Programming: Authorized Assembler Services Guide* for more information about service request blocks.

gsk_srb_initialize()

Related Topics

"GSKSRBRD" on page 513

"GSKSRBWT" on page 514

GSKSRBRD

Reads from 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 `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 *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. See 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” on page 514](#)

[“gsk_initialize\(\)” on page 491](#)

[“gsk_secure_soc_close\(\)” on page 496](#)

[“gsk_secure_soc_init\(\)” on page 497](#)

[“gsk_secure_soc_write\(\)” on page 509](#)

[“gsk_srb_initialize\(\)” on page 511](#)

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 `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 *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 `gskssl.h` include file. See the description of the `gsk_secure_soc_write()` routine for more information.

Usage

The GSKSRBWT routine is called to write to 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 GSKSRBWT routine will pass the write request to one of the SRB service tasks. The service task will then call the `gsk_secure_soc_write()` routine. The GSKSRBWT routine will not return until the `gsk_secure_soc_write()` routine has completed.

Related Topics

["GSKSRBRD" on page 513](#)

["gsk_initialize\(\)" on page 491](#)

["gsk_secure_soc_close\(\)" on page 496](#)

["gsk_secure_soc_init\(\)" on page 497](#)

["gsk_secure_soc_write\(\)" on page 509](#)

["gsk_srb_initialize\(\)" on page 511](#)

gsk_uninitialize()

Terminates the SSL environment.

Format

```
#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 is not closed until the last connection is closed.

Related Topics

[“gsk_initialize\(\)” on page 491](#)

[“gsk_secure_soc_init\(\)” on page 497](#)

gsk_user_set()

Sets an application callback.

Format

```
#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 is used only for SSL servers (the internal cache is always used for SSL clients). This sets the session identifier cache for existing connections including 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.

```
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_data_buffer * Put (
    gsk_data_buffer *   ssl_session_data,
    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).

```
void Delete (
    const unsigned char * session_id,
    unsigned int         session_id_length,
    gsk_sslversion       ssl_version)
```

The routine specified by the *FreeDataBuffer* entry is called to release the data buffer returned by the *Get* routine.

```
void FreeDataBuffer (
    gsk_data_buffer *   ssl_session_data)
```

[GSK_RESET_SIDCACHE_CALLBACK]

This function resets the session identifier cache callback. The internal session identifier cache is used instead of an application session identifier cache. This resets the session identifier cache for existing connections including 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.

```
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.

gsk_user_set()

Related Topics

["gsk_initialize\(\)" on page 491](#)

["gsk_secure_soc_init\(\)" on page 497](#)

Chapter 10. Certificate/Key management

This topic discusses the use of the z/OS shell-based **gskkyman** utility to manage private keys, certificates, and tokens. In addition, see “gskkyman command line mode examples” on page 580 for more detailed examples using the **gskkyman** utility.

This topic also provides an overview of the certificate revocation support provided in System SSL.

Introduction

SSL connections use 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) might need to be created, or certificates must be requested, received, and managed.

System SSL supports three methods for managing PKI private keys and certificates:

- A z/OS shell-based program called **gskkyman**. **gskkyman** creates, completes, 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. See *z/OS Security Server RACF Command Language Reference* for details about 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.
- PKCS #12 standard files created according to PKCS #12 V3.0. These files must be created as binary format files whose fully qualified file name does not exceed 251 characters in length and does not end with **.kdb**, **.rdb**, or **.sth**.

System SSL supports PKCS #12 certificate and private key objects types. Any other object types within the file are ignored. All certificates within the files are treated as trusted certificates and no certificate can be identified as a default certificate.

The PKCS #12 file is protected by a password and the integrity of the file is ensured by a SHA-1 message authentication value.

Because PKCS #12 files do not have labels as certificates in key database files (SAF key rings or PKCS #11 tokens can have labels as certificates in key database files), when the certificates are read into storage, they are assigned a label using either the PKCS #12 friendly name, if one exists, or the certificate's subject distinguished name. When the friendly name or the subject distinguished name value is greater than 127 characters, only the first 127 characters are used. If multiple certificates have the same friendly name value, the first encountered certificate is read into storage. Any other certificate with that friendly name is ignored. If a certificate is encountered that does not contain a friendly name and the subject distinguished name is empty, the processing of the PKCS #12 fails. As with key database files, SAF key rings and PKCS #11 tokens, the label is case sensitive.

- 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 or a PKCS #12 file, the 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.

x.509 certificate revocation

When x.509 certificates are issued, they are assigned a validity period that defines a start and end (expiration) date and time for the certificate. Certificates are considered valid if used during the validity period. If the certificate is deemed to be no longer trustable prior to its expiration date, it can be revoked by the issuing Certificate Authority (CA). The process of revoking the certificate is known as certificate revocation. There are a number of reasons why certificates are revoked. Some common reasons for revocation are:

- Encryption keys of the certificate have been compromised.
- Errors within an issued certificate.
- Change in usage of the certificate.
- Certificate owner is no longer deemed trusted.

Two methods of revocation are supported:

Certificate Revocation List (CRL)

A CRL is a list of revoked certificates (by serial number) that have been issued and then subsequently revoked by a given CA. CRLs are generally published on a periodic interval or can be published only when a certificate is revoked by the CA.

The CRL, like a certificate, is signed by the owning CA to ensure the authenticity of the CRL contents and has a start and end (expiration) date and time. The start date and time is known as `thisUpdate` and the end date and time is known as `nextUpdate`.

Supported CRLs can be obtained from a dedicated LDAP server or through a certificate's CRL Distribution Point (CDP) extension. HTTP Uniform Resource Identifier (URIs) values within the CDP may be used.

For information about configuring your SSL application to perform CRL revocation checking for SSL secure connections, see "SSL/TLS partner certificate revocation checking" on page 46.

For information about utilizing CRL revocation information from a CMS application, see "`gsk_validate_certificate_mode()`" on page 464.

Online Certificate Status Protocol (OCSP) responses

OCSP is an internet protocol used for obtaining the revocation status of an x.509 certificate. The protocol defines the type of data that is exchanged between the requester of the revocation status (OCSP client) and the server (OCSP responder) providing the revocation status information. Certificate revocation information is provided by the OCSP responder through an OCSP response.

The OCSP response, like a CRL, is signed by the owning CA (or designated CA) to ensure the authenticity of the OCSP response contents and has a start and end (expiration) date and time. The start date and time is known as `thisUpdate` and the end date and time is known as the `nextUpdate`.

Supported OCSP responses can be obtained from a dedicated OCSP responder or through OCSP responders identified through a certificate's Authority Information Access (AIA) extension. AIA extensions may be used when the extension contains an entry with an OCSP access method and a URI access location. The AIA extension can contain multiple entries.

For information about configuring your SSL application to perform OCSP revocation checking for SSL secure connection, see “SSL/TLS partner certificate revocation checking” on page 46.

For information about utilizing OCSP revocation information from a CMS application, see “gsk_validate_certificate_mode()” on page 464.

gskkyman Overview

gskkyman is a z/OS shell-based program that creates, completes, 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 **gskkyman** utility only supports clear key operations.

The **gskkyman** utility only supports certificates that conform to RFC 2459: *Internet x.509 Public Key Infrastructure Certificate and CRL Profile* or RFC 3280: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*.

RFC 5280: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile* certificates can be used with **gskkyman** provided they conform to RFC 3280 rules for the certificate issuer name and subject name comparisons. Specifically, RFC 3280 indicates that UTF-8 values in the distinguished names must pass a case-sensitive (exact match) comparison to be considered equal. The **gskkyman** utility uses the issuer name and subject name values in the certificate to determine if a certificate is self-signed, and to perform certificate chaining. Therefore, **gskkyman** expects distinguished name attribute values to match according to a case-sensitive comparison when they are encoded as UTF-8 strings. Certificates that contain distinguished names with UTF-8 encoded attribute values for either the issuer name, the subject name, or both, that match through a case-insensitive comparison, can be created according to RFC 5280. Such certificates cause the **gskkyman** utility to fail checking for self-signed certificates and fail to correctly build certificate chains. Therefore, these certificates cannot be used with **gskkyman**.

The interface to **gskkyman**, while command-line based, is an interactive dialog between you (the user) and the utility. At each step, the interactive **gskkyman** utility prompts you with one or more lines of output and expects a numeric choice to be supplied as input at the prompt. When a choice is made, the **gskkyman** utility prompts you for the individual pieces of information that is 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 you want other than the default, enter the value at the prompt and press Enter. If a value is entered that is outside of the acceptable range of inputs, you are prompted again for the information.

Note: For a description of command-line mode functions and options, see “gskkyman command line mode syntax” on page 577.

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, **gskkyman** is located. If your **PATH** environment variable does not contain this directory, add **/usr/lpp/gskssl/bin** to your **PATH** using:

```
PATH=$PATH:/usr/lpp/gskssl/bin
```

/usr/lib/nls/msg/En_US.IBM-1047 (and **/usr/lib/nls/msg/Ja_JP.IBM-939** for JCPT42J 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:

```
export NLSPATH=$NLSPATH:/usr/lpp/gskssl/lib/nls/msg/%L/%N
```

This setting assumes that your environment has the **LANG** environment variable set to **En_US.IBM-1047** (or **Ja_JP.IBM-939** for JCPT42J installations that expect Japanese messages and prompts). If **LANG** is not set properly, set the **NLSPATH** environment variable using this command:

```
export NLSPATH=/usr/lpp/gskssl/lib/nls/msg/En_US.IBM-1047/%N:$NLSPATH
```

or for JCPT42J installations that expect Japanese messages and prompts:

```
export NLSPATH=/usr/lpp/gskssl/lib/nls/msg/Ja_JP.IBM-939/%N:$NLSPATH
```

The DLLs for System SSL are installed into a partitioned data set (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** data set cannot be put into the LPA.

If the System SSL DLLs are not in either the dynamic LPA or system link list, you must set the **STEPLIB** environment variable to find the DLLs. For example:

```
export STEPLIB=$STEPLIB:<HLQ>.SIEALNKE
```

During installation, the sticky bit is set on for the **gskkyman** utility. If the sticky is turned off, attempts to invoke the **gskkyman** utility results in message **GSK00009E** indicating that a problem exists with the installation of the SSL utility, **gskkyman**.

To check the sticky bit setting, issue:

```
ls -l /usr/lpp/gskssl/bin/gskkyman
```

The first part of the output should be:

```
-rwxr-xr-t
```

The **t** indicates that the sticky bit is on.

To set the sticky bit on, from an authorized id, issue:

```
chmod +t /usr/lpp/gskssl/bin/gskkyman
```

If access to the ICSF callable services are protected with **CSFSERV** class profiles on your system, the user ID issuing the **gskkyman** utility might need to be given **READ** authority to call ICSF callable services **CSFIQA**, **CSFPPRE**, **CSFPGKP**, **CSFPGSK**, **CSFPGAV**, **CSFPTRD**, **CSFPTRC**, **CSFPPKS**, and **CSFPPKV**. If these callable services are protected with a generic **CSF*** profile in the **CSFSERV** class, access can be granted by entering:

```
PERMIT CSF* CLASS(CSFSERV) ID(user-ID) ACCESS(READ)
SETROPTS RACLIST(CSFSERV) REFRESH
```

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 suggested 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 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, you should set up a group to control access to the key database file. In this case, it is suggested 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 is 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. The resources for this class are:

- USER.tokenname
- SO.tokenname

The **gskkyman** utility provides limited functionality for PKCS #11 token certificates that have secure private keys. If a PKCS #11 certificate has a secure private key, the following functions are allowed:

- Showing certificate and key information.
- Setting the key as default.
- Exporting a certificate to a file.
- Deleting a certificate and key.
- Changing the label.

If a PKCS #11 token certificate has a secure private key, the following functions are not allowed:

- Copying certificate and key to another token.
- Exporting certificate and key to a file.

- Creating a signed certificate and key.
- Creating a certificate renewal request.

A PKCS #11 token certificate with a clear private key is allowed full **gskkyman** functionality.

When displaying token key information for a PKCS #11 certificate's private key, the private key type indicates the private key is either clear or secure.

Table 15 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 including 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 15. SAF access levels

Resource	Function	SAF access level
USER.token-name CRYPTOZ	Create/delete/modify CA certificate and private key	Control
	Create/delete/modify user certificate and private key	Update
	Read certificate and private key	Read
	Set default key	Update
SO.token-name CRYPTOZ	Create or delete token	Update
	Read/create/delete/modify certificate (but not the private key)	Read
	Read/create/delete/modify private key	Control
	Set default key	Read

gskkyman interactive mode descriptions

Interactive mode is entered when the **gskkyman** utility is entered without any parameters. A series of menus are presented to allow you to select the database functions to be performed. Leading and trailing blanks are removed from data entries but embedded blanks are retained. Blanks are not removed from passwords.

Database menu

This is the main menu that is displayed when the **gskkyman** utility starts:

```
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:
```

Figure 2. Database menu

Create new database

This option creates a new key database and the associated request database. You are 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 523 for a discussion of FIPS mode databases).

The fully-qualified key database name must be between 2 and 251 characters. The file can contain an extension consisting of 1 to 3 characters. The suggested extension is ".kdb". The maximum database name is 247 characters if the name does not end with an extension 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 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 that 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 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 that 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 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.

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. See FIPS 186-3: *Digital Signature Standard (DSS)* and RFC 2631: *Diffie-Hellman Key Agreement Method* for more information about 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 file name. The fully-qualified certificate file name 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 before 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 shown throughout this topic by the key database menu and token menu 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

Database: Database_name
Expiration Date: Expiration Date
Type: non-FIPS or FIPS

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal
  certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Store database password
11 - Show database record length

0 - Exit program

Enter option number (press ENTER to return to
previous menu):
====>

```

Figure 3. Key Management Menu

```

Token Management Menu

Token: Token_name

Manufacturer: z/OS PKCS11 API
Model: HCR77B0
Flags: 0x00000509 (INITIALIZED,PROT AUTH
  PATH,USER PIN INIT,RNG)

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal
  certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Delete token

0 - Exit program

Enter option number (press ENTER to return to
previous menu):
====>

```

Figure 4. 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:

```

Key and Certificate Menu

Label: Certificate_label_name

1 - Show certificate information
2 - Show key information
3 - Set key as default
4 - Set certificate trust status
5 - Copy certificate and key to another
  database/token
6 - Export certificate to a file
7 - Export certificate and key to a file
8 - Delete certificate and key
9 - Change label
10 - Create a signed certificate and key
11 - Create a certificate renewal request

0 - Exit program

Enter option number (press ENTER to return to
previous menu):
====>

```

Figure 5. Key and Certificate Menus

```

Token Key and Certificate Menu

Label: Certificate_label_name

1 - Show certificate information
2 - Show key information
3 - Set key as default
4 - Set certificate trust status
5 - Copy certificate and key to another
  database/token
6 - Export certificate to a file
7 - Export certificate and key to a file
8 - Delete certificate and key
9 - Change label
10 - Create a signed certificate and key
11 - Create a certificate renewal request

0 - Exit program

Enter option number (press ENTER to return to
previous menu):
====>

```

Figure 6. Token Key and Certificate 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 into a FIPS mode database from 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 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. See FIPS 186-2: *DIGITAL SIGNATURE STANDARD (DSS)* and RFC 2631: *Diffie-Hellman Key Agreement Method* for more information about 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 email 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 email 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). 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 and the following digest type:

- For RSA signatures, the digest type matches that used in the signature algorithm of the signing certificate. If the digest type is not a SHA-based digest, then SHA-1 is used.
- For DSA signatures using a 1024-bit DSA key, the digest type is SHA-1. When using a 2048-bit DSA key, the user is offered a choice of SHA-2 digest algorithms.
- For ECC Signatures, the digest type is the suggested digest for the key size of the ECC private key, as specified in Table 2 on page 15.

Possible signature algorithms are:

- x509_alg_sha1WithRsaEncryption
- x509_alg_sha224WithRsaEncryption
- x509_alg_sha256WithRsaEncryption
- x509_alg_sha384WithRsaEncryption
- x509_alg_sha512WithRsaEncryption
- x509_alg_dsaWithSha1
- x509_alg_dsaWithSha224
- x509_alg_dsaWithSha256
- x509_alg_ecdsaWithSha256
- x509_alg_ecdsaWithSha384
- x509_alg_ecdsaWithSha512

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

Label: Certificate_label_name

1 - Show certificate information
2 - Set certificate trust status
3 - Copy certificate to another database/token
4 - Export certificate to a file
5 - Delete certificate
6 - Change Label

0 - Exit program

Enter option number (press ENTER to return to
previous menu):

```

Figure 7. Certificate Menu

```

Token Certificate Menu

Label: Certificate_label_name

1 - Show certificate information
2 - Set certificate trust status
3 - Copy certificate to another database/token
4 - Export certificate to a file
5 - Delete certificate
6 - Change Label

0 - Exit program

Enter option number (press ENTER to return to the
previous menu):
====>

```

Figure 8. Token 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 may only be copied into a FIPSmode database from another FIPSmode database. A certificate may not be copied from a non-FIPSmode database or a PKCS #11 token to a FIPSmode 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 displays the next set of labels. Selecting one of the label numbers will display this menu:

```

Request Menu

Label: label_name

1 - Show key information
2 - Export certificate request to a file
3 - Delete certificate request and key
4 - Change label

0 - Exit program

Enter option number (press ENTER to return to
previous menu):

```

Figure 9. Request Menu

```

Token Certificate Request Menu

Label: label_name

1 - Show key information
2 - Export certificate request to a file
3 - Delete certificate request and key
4 - Change label

0 - Exit program

Enter option number (press ENTER to return to
previous menu):
===>

```

Figure 10. Token Certificate Request 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 creates a certificate request using either RSA or DSA encryption for the public and private keys. The certificate request is 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 is also used when the certificate is received, so it must be unique in both the request and key databases. It must consist of characters that 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 is also used when the certificate is received, so it must be unique in the token. It must consist of characters that 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 email 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 that are separated by periods. An email address consists of a user name and a domain name that is 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>

).

Receive requested certificate or a renewal certificate: This option receives 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: *Internet x.509 Public Key Infrastructure Certificate and CRL Profile*, RFC 3280: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, RFC 5280: *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 creates a self-signed certificate using either RSA, DSA, or ECC encryption for the public and private keys, and a certificate signature that is based on a SHA digest algorithm. The SHA digest algorithm that is used depends on the key algorithm that is chosen for the certificate:

- If an RSA certificate is requested, the user is prompted to choose the SHA digest algorithm required.
- An ECC certificate uses the suggested digest for the key size of the ECC key, as specified in Table 2 on page 15.

- A 1024-bit DSA certificate uses SHA-1. For a 2048-bit DSA certificate, the user is prompted to choose the SHA digest algorithm required.

Possible signature algorithms are:

- x509_alg_sha1WithRsaEncryption
- x509_alg_sha224WithRsaEncryption
- x509_alg_sha256WithRsaEncryption
- x509_alg_sha384WithRsaEncryption
- x509_alg_sha512WithRsaEncryption
- x509_alg_dsaWithSha1
- x509_alg_dsaWithSha224
- x509_alg_dsaWithSha256
- x509_alg_ecdsaWithSha256
- x509_alg_ecdsaWithSha384
- x509_alg_ecdsaWithSha512

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.

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 is also used when the certificate is received, so it must be unique in both the request and key databases. It must consist of characters that 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 is also used when the certificate is received, so it must be unique in the token. It must consist of characters that 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 email 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 email address consists of a user name and a domain name that is 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 suggested for use in production environments and should only be used to facilitate

test environments before 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. However, certificates that are signed by a certificate authority indicate that, at least at the time of signature, the certificate authority approved the information that is 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, 2106.

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 before 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: *Internet x.509 Public Key Infrastructure Certificate and CRL Profile*, RFC 3280: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, RFC 5280: *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 is imported if 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 is imported if 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 or z/OS PKCS #11 token already contains the certificate.

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 a 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
- 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 11 on page 538).

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 11 on page 538 shows the **gskkyman** start menu.

```

# gskkyman <enter>

      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:
===>

```

Figure 11. Starting Menu for gskkyman

From the **Database Menu** for **gskkyman**, you can create a new key database, open an existing key database, display the contents of a certificate file, change a database password, change a database record length, delete a database, create, delete, and manage a z/OS PKCS #11 token, or exit **gskkyman**.

Creating, opening, and deleting a key database file

To create a new key database, enter **1** at the command prompt on the **Database Menu**:

```

      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: 1 <enter>
Enter key database name (press ENTER to return to menu): mykey.kdb <enter>
Enter database password (press ENTER to return to menu): <enter password>
Re-enter database password: <enter password>
Enter password expiration in days (press ENTER for no expiration): 35 <enter>
Enter database record length (press ENTER to use 5000): <enter>

Enter 1 for FIPS mode database or 0 to continue: 0 <enter>

Key database /home/sufw11/ssl_cmd/mykey.kdb created.

Press ENTER to continue.
===>

```

Figure 12. Creating a New Key Database

Figure 12 on page 538 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:

1. When dealing with certificates which may be large or have large key sizes, for example 2048 or 4096, an initial key record length of 5000 may be required.
2. The maximum length of the password specified for a key database file is 128 characters.
3. When creating a new key database file, you will be prompted whether you want a FIPS or non-FIPS database file created. For more information about FIPS mode databases, see “Key database files” on page 523.

After entering the database record length, a message displays confirming that your database was created (see Figure 12 on page 538). You are prompted to press Enter to continue. Doing so displays the **Key Management Menu** for the database you have created:

```
Key Management Menu

Database: /home/sufw11/ssl_cmd/mykey.kdb
Expiration Date: 2025/12/02 10:11:12
Type: non-FIPS

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Store database password
11 - Show database record length

0 - Exit program

Enter option number (press ENTER to return to previous menu):
===>
```

Figure 13. Key Management Menu for **gskkyman**

Figure 13 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 14 on page 540). 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.

```
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: 2 <enter>
Enter key database name (press ENTER to return to menu): mykey.kdb <enter>
Enter database password (press ENTER to return to menu): <enter password>

===>
```

Figure 14. 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 15).

```
Key Management Menu

Database: /home/sufw11/ssl_cmd/mykey.kdb
Expiration Date: 2025/12/02 10:11:12
Type: non-FIPS

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Store database password
11 - Show database record length

0 - Exit program

Enter option number (press ENTER to return to previous menu):
===>
```

Figure 15. Key Management Menu

To delete an existing database, from the **Database Menu**, select option 5 (see Figure 16 on page 541):

```
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 16. Deleting an Existing Key Database

You are prompted to enter the key database name that you want 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) is also 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**:

```
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: 3 <enter>
Enter key database name (press ENTER to return to menu): mykey.kdb <enter>
Enter database password (press ENTER to return to menu): <enter current password>
Enter new database password (press ENTER to return to menu): <enter new password>
Re-enter database password: <enter new password>
Enter password expiration in days (press ENTER for no expiration): <enter>

Database password changed.

Press ENTER to continue.
===>
```

Figure 17. Changing a Key Database Password

Figure 17 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 and 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 opened the key database and are displaying the **Key Management Menu** panel.

```
Key Management Menu

Database: /home/sufw1/ssl_cmd/mykey.kdb
Expiration Date: 2025/12/02 10:11:12
Type: non-FIPS

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Store database password
11 - Show database record length

0 - Exit program

Enter option number (press ENTER to return to previous menu): 10 <enter>

Database password stored in /home/sufw1/ssl_cmd/mykey.sth.

Press ENTER to continue.
===>
```

Figure 18. Key Management Menu

Figure 18 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**:

```
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: 11 <enter>
Enter token name (press ENTER to return to menu): TOKEN1 <enter>

Token successfully created

Press ENTER to continue.
```

Figure 19. Creating a New z/OS PKCS #11 Token

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.tokenname" have the authority to create the z/OS PKCS #11 token with the name "tokenname".

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 19 on page 543). You are prompted to press Enter to continue. Doing so redisplay 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:

```
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: 13 <enter>
Enter token name (press ENTER to return to menu): TOKEN1 <enter>
```

Figure 20. Opening a z/OS PKCS #11 Token from token name

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 21. Opening a z/OS PKCS #11 Token from 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 Figure 28 on page 550).

```
Token Management Menu

Token: TOKEN1

Manufacturer: z/OS PKCS11 API
Model: HCR77B0
Flags: 0x00000509 (INITIALIZED,PROT AUTH PATH,USER PIN INIT,RNG)

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Delete token

0 - Exit program

Enter option number (press ENTER to return to previous menu):
===>
```

Figure 22. Token Management Menu

Note: Only users with SAF access level of READ, UPDATE, or CONTROL to the CRYPTOZ resource "so.tokenname" or "user.token.name" have the authority to open the z/OS PKCS #11 token with the name "tokenname".

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:

```
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.

===>
```

Figure 23. Deleting an existing z/OS PKCS #11 Token

If option **10** on the **Token Management Menu** is used:

```
Token Management Menu

Token: TOKEN1

Manufacturer: z/OS PKCS11 API
Model: HCR77B0
Flags: 0x00000509 (INITIALIZED, PROT AUTH PATH, USER PIN INIT, RNG)

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Delete token

0 - Exit program

Enter option number (press ENTER to return to previous menu): 10 <enter>
To confirm token delete, enter token name again (press ENTER to cancel delete): TOKEN1 <enter>

Token successfully deleted

Press ENTER to continue.

===>
```

Figure 24. Deleting an existing z/OS PKCS #11 Token

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.tokenname" have the authority to delete the z/OS PKCS #11 token with the name "tokenname".

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 558.

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 suggested for production environments and should only be used to facilitate test environments before 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. However, 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 are prompted for a number of items to define the certificate, including the intended use of the certificate, the key algorithm and key size, and possibly the digest algorithm for the certificate signature.

```

Key Management Menu

Database: /home/sufw11/ssl_cmd/mykey.kdb
Expiration Date: 2025/12/02 10:11:12
Type: non-FIPS

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a
  renewal certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Store database password
11 - Show database record length

0 - Exit program

Enter option number (press ENTER to return to
previous menu): 6 <enter>
====>

```

Figure 25. Creating a Self-Signed Certificate-Key Management Menu

```

Token Management Menu

Token: TOKEN1

Manufacturer: z/OS PKCS11 API
Model: HCR77B0
Flags: 0x00000509 (INITIALIZED,PROT AUTH
  PATH,USER PIN INIT,RNG)

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal
  certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Delete token

0 - Exit program

Enter option number (press ENTER to return to
previous menu): 6 <enter>

```

Figure 26. Creating a Self-Signed Certificate-Token Management Menu

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. ECC key certificates represent certificates that use Elliptic Curve Cryptography. The larger the key size, the more secure the generated key will be. Note that CPU usage increases as the key size increases.

If an RSA-based certificate is selected, you will be prompted to select the key size and the digest type for the signature algorithm. See Figure 27 on page 549 for an example of selecting the key size and digest type.

If a 1024-bit DSA certificate is selected, SHA-1 will be used for the signature algorithm. If a 2048-bit DSA certificate is selected, you will be prompted to select the digest type for the signature algorithm from a list of SHA-based digest types.

If an ECC certificate is selected, you will be prompted to select the ECC key type and curve type. The suggested digest for the key size of the ECC key will be used for the signature algorithm, as specified in Table 2 on page 15. See “Creating a signed ECC certificate and key” on page 566 for more information.

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 27 on page 549 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.

```

Certificate Usage

1 - CA certificate
2 - User or server certificate

Select certificate usage (press ENTER to return to menu): 2 <enter>

Certificate Key Algorithm

1 - Certificate with an RSA key
2 - Certificate with a DSA key
3 - Certificate with an ECC key

Select certificate key algorithm (press ENTER to return to menu): 1 <enter>

RSA Key Size

1 - 1024-bit key
2 - 2048-bit key
3 - 4096-bit key

Select RSA key size (press ENTER to return to menu): 1 <enter>

Signature Digest Type

1 - SHA-1
2 - SHA-224
3 - SHA-256
4 - SHA-384
5 - SHA-512

Select digest type (press ENTER to return to menu): 1 <enter>
Enter label (press ENTER to return to menu): Server Cert <enter>
Enter subject name for certificate
Common name (required): My Server Certificate <enter>
Organizational unit (optional): ID <enter>
Organization (required): IBM <enter>
City/Locality (optional): Endicott <enter>
State/Province (optional): NY <enter>
Country/Region (2 characters - required): US <enter>
Enter number of days certificate will be valid (default 365): 244 <enter>

Enter 1 to specify subject alternate names or 0 to continue: 0 <enter>

Please wait .....

Certificate created.

Press ENTER to continue.
===>

```

Figure 27. Creating a Self-Signed Certificate

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 tokens 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 about setting the default certificate, see "Marking a certificate (and private key) as the default certificate" on page 558.

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 that the self-signed certificate must be imported into the partner's database or z/OS PKCS #11 token. For more information about importing certificates, see "Importing a certificate from a file as a trusted CA certificate" on page 571.

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**.

```
Key Management Menu

Database: /home/sufw11/ssl_cmd/mykey.kdb
Expiration Date: 2025/12/02 10:11:12
Type: non-FIPS

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal
  certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Store database password
11 - Show database record length

0 - Exit program

Enter option number (press ENTER to return to
previous menu): 4 <enter>
===>
```

Figure 28. Creating a certificate request-Key Management Menu

```
Token Management Menu

Token: TOKENABC

Manufacturer: z/OS PKCS11 API
Model: HCR77B0
Flags: 0x00000509 (INITIALIZED,PROT AUTH
PATH,USER PIN INIT,RNG)

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal
  certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Delete token

0 - Exit program

Enter option number (press ENTER to return to
previous menu): 4 <enter>
===>
```

Figure 29. Creating a certificate request-Token Management Menu

When creating a certificate request, you are prompted to select the key algorithm and the key size for the certificate to be requested. RSA key certificates are the most common. DSA key certificates represent certificates that follow the FIPS-186 government standard. ECC key certificates represent certificates that use Elliptic Curve Cryptography. The larger the key size, the more secure the encryption/decryption generated key is.

If an RSA-based certificate is selected, you are prompted to select the digest type for the signature algorithm from a list of SHA-based digest types.

If a 1024-bit DSA certificate is selected, SHA-1 is used for the signature algorithm. If a 2048-bit DSA certificate is selected, you are prompted to select the digest type for the signature algorithm from a list of SHA-based digest types.

If an ECC certificate is selected, you will be prompted to select the ECC key type and curve type. The suggested digest for the key size of the ECC key is used for the signature algorithm, as specified in Table 2 on page 15.

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 Key Algorithm** menu appears:

```
Certificate Key Algorithm
1 - Certificate with an RSA key
2 - Certificate with a DSA key
3 - Certificate with an ECC key

Select certificate key algorithm (press ENTER to return to menu): 1 <enter>

RSA Key Size
1 - 1024-bit key
2 - 2048-bit key
3 - 4096-bit key

Select RSA key size (press ENTER to return to menu): 1 <enter>

Signature Digest Type
1 - SHA-1
2 - SHA-224
3 - SHA-256
4 - SHA-384
5 - SHA-512

Select digest type (press ENTER to return to menu): 3 <enter>
Enter request file name (press ENTER to return to menu): certreq.arm <enter>
Enter label (press ENTER to return to menu): Test Server Cert <enter>
Enter subject name for certificate
Common name (required): Test Server <enter>
Organizational unit (optional): ID <enter>
Organization (required): IBM <enter>
City/Locality (optional): Endicott <enter>
State/Province (optional): NY <enter>
Country/Region (2 characters - required): US <enter>

Enter 1 to specify subject alternate names or 0 to continue: 0 <enter>

Please wait .....

Certificate request created.

Press ENTER to continue.
===>
```

Figure 30. Creating a Certificate Request

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:

```
Subject Alternate Name Type
1 - Directory name (DN)
2 - Domain name (DNS)
3 - E-mail address (SMTP)
4 - Network address (IP)
5 - Uniform resource identifier (URI)

Select subject alternate name type (press ENTER if name is complete): 1 <enter>
Enter subject name for certificate
Common name (required): Test server <enter>
Organizational unit (optional): IBM <enter>
Organization (required): IBM <enter>
City/Locality (optional): Endicott <enter>
State/Province (optional): NY <enter>
Country/Region (2 characters - required): US <enter>

Subject Alternate Name Type
1 - Directory name (DN)
2 - Domain name (DNS)
3 - E-mail address (SMTP)
4 - Network address (IP)
5 - Uniform resource identifier (URI)

Select subject alternate name type (press ENTER if name is complete): <enter>

Please wait .....
```

Figure 31. Specifying subject alternate names

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 file name. If you choose to exit **gskkyman**, the program ends. Otherwise, the **Key Management Menu** or the **Token Management Menu** (see Figure 15 on page 540) 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 30 on page 551:

```

$ cat certreq.arm<enter>
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBnjCCAQcCAQAwXjELMAKGA1UEBhMCMVVMxCzAJBgNVBAGTAk5ZMREwDwYDVQQH
EwhFbmRyY290dDEMMAoGA1UEChMDSUJNMQswCQYDVQQLLEwJJRDEUMBGA1UEAxML
VGZvdCBTZXJ2ZXIwZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMFb+u7rmqdN
vgk0p43Wn4/KNZayCOva0jBjxZXT79IddhzTxg16YL3Ac1xEanNe+sA4jB9FBjgP
Uh3oAn6tS7FiB47Nv0w+ALK+3iv8UVjhwxxBe5Ebh4j6ZNeX2kJRvGwITfUSyF8t
nXNmavsBk8dAcaozdho7D/GwiFxRLzr7AgMBAAGgADANBgkqhkiG9w0BAQsFAA0B
gQB0kJIC5u8FSLaVJv++n96bP1NyMEDzH/3s0Z9CzIM/Z2YSSuhoB8SL+y0BSvtj
sbnEcoS3+nXj1yTobgAdkXQixDRjLi almauqAdgjEeazSV9/FMM1IPxTGZjG7bou
pfY4UtKXFAEg5TcZdSaCUv95qkopg8mDNoHg8NX9cpvoRg==
-----END NEW CERTIFICATE REQUEST-----
$

```

Figure 32. Contents of certreq.arm after Certificate Request Generation

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 is named the same as the key database file, except it has an extension of .rdb. For example, a key database file of key.kdb causes 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.

The certificate request must not be deleted from the database while the request is being processed by the signing certificate authority. The database certificate request is required for applicable processing when the signed certificate from the certificate authority is received. The removal of the certificate request from the database causes the private key associated with the certificate request to be lost.

Receiving the signed certificate or renewal certificate

When 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** utility. This file should be in the current working directory when **gskkyman** is started. If this file is on another working directory, you must specify the fully qualified name.

Note: 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, see “Importing a certificate from a file as a trusted CA certificate” on page 571.

To receive a certificate that is issued on your behalf, from the **Key Management Menu** or **Token Management Menu**, see Figure 15 on page 540 and enter option 5.

```

Key Management Menu

Database: /home/sufw11/ss1_cmd/mykey.kdb
Expiration Date: 2025/12/02 10:11:12
Type: non-FIPS

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal
  certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Store database password
11 - Show database record length

0 - Exit program

Enter option number (press ENTER to return to
previous menu): 5 <enter>

Enter certificate file name (press ENTER to return
to menu): signed.arm <enter>

Certificate received.

Press ENTER to continue.
====>

```

Figure 33. Receiving a Certificate Issued for your Request-Key Management Menu

```

Token Management Menu

Token: TOKENABC

Manufacturer: z/OS PKCS11 API
Model: HCR77B0
Flags: 0x00000509 (INITIALIZED,PROT AUTH
      PATH,USER PIN INIT,RNG)

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal
  certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Delete token

0 - Exit program

Enter option number (press ENTER to return to
previous menu): 5 <enter>

Enter certificate file name (press ENTER to return
to menu): signed.arm <enter>

Certificate received.

Press ENTER to continue.
====>

```

Figure 34. Receiving a Certificate Issued for your Request-Token Management Menu

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 (See “Creating a certificate request” on page 550). After you receive the certificate, press Enter to continue working with the **Key Management Menu** or **Token Management Menu**. Upon completion of this step and before the System SSL APIs using the certificate during the SSL handshake processing, you must 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 about setting the default certificate, see “Marking a certificate (and private key) as the default certificate” on page 558.

When received into a key database file, the certificate's expiration date should be monitored. When the expiration date is nearing (do not wait until it is expired), a new certificate should be obtained to replace the existing certificate. The new certificate can be a brand new certificate with new public/private keys or a renewal certificate where existing keys and certificate information is used. See Figure 29 on page 550 for more information about a new or renewal certificate.

Managing keys and certificates

When 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

- Create a signed ECC certificate and key
- Create a certificate to be used with a fixed Diffie-Hellman key exchange
- Create a certificate renewal request

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 15 on page 540) select **1**, (Manage keys and certificates). This displays the **Key and Certificate List**.

```

Key and Certificate List

Database: /home/sufw11/ssl_cmd/mykey.kdb

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 35. Key and Certificate List

```

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 36. Token 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 37).

```

Key and Certificate Menu

Label: Server Cert

1 - Show certificate information
2 - Show key information
3 - Set key as default
4 - Set certificate trust status
5 - Copy certificate and key to another
  database/token
6 - Export certificate to a file
7 - Export certificate and key to a file
8 - Delete certificate and key
9 - Change label
10 - Create a signed certificate and key
11 - Create a certificate renewal request

0 - Exit program

Enter option number (press ENTER to return to
previous menu): 1 <enter>
====>

```

Figure 37. Key and Certificate Menu

```

Token Key and Certificate Menu

Label: Server Cert

1 - Show certificate information
2 - Show key information
3 - Set key as default
4 - Set certificate trust status
5 - Copy certificate and key to another
  database/token
6 - Export certificate to a file
7 - Export certificate and key to a file
8 - Delete certificate and key
9 - Change label
10 - Create a signed certificate and key
11 - Create a certificate renewal request

0 - Exit program

Enter option number (press ENTER to return to
previous menu): 1 <enter>
====>

```

Figure 38. Token Key and Certificate Menu

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 39 on page 556):

```

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: 2010/02/20
Expiration date: 2015/10/22
Signature algorithm: sha1WithRsaEncryption
Issure 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 2D 99
48 71 F6 67 DE B9 8D EB B7 9E 86 80 0A 91 0E FA
38 25 AF 46 88 82 E5 73 A8 A0 9B 24 5D 0D 1F CC
65 6E 0C B0 D0 56 84 18 87 9A 06 9B 10 A1 73 DF
B4 58 39 6B 6E C1 F6 15 D5 A8 A8 3F AA 12 06 8D
31 AC 7F B0 34 D7 8F 34 67 88 09 CD 14 11 E2 4E
45 56 69 1F 78 02 80 DA Dc 47 91 29 BB 36 C9 63
5C C5 E0 D7 2D 87 7B A1 B7 32 B0 7B 30 BA 2A 2F
31 AA EE A3 67 DA DB 02 03 01 00 01

Number of extensions: 4

Enter 1 to display extensions, 0 to return to menu: 1 <enter>
===>

```

Figure 39. Certificate Information

Note: For a z/OS PKCS #11 certificate, the Record ID and Issuer Record ID is N/A.

From the **Certificate Information** screen, you can also enter **1** to display certificate extensions:

```

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 40. Certificate extensions list

Enter 3 on the Certificate Extensions List to show key usage information:

```
Certificate signature
CRL signature

Press ENTER to continue.
===>
```

Figure 41. 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 42) or the **Token key information** menu (see Figure 43 or Figure 44 on page 558 :

```
Key Information

Label: Server Cert
Record ID: 13
Issuer Record ID: 13
Default key: Yes
Private key algorithm: rsaEncryption
Private key size: 1024
Subject name: My Server Certificate
              ID
              IBM
              Endicott
              NY
              US

Press ENTER to continue.
===>
```

Figure 42. Key information menu

```
Token key information

Label: Sample RSA Certificate 1
Record ID: N/A
Issuer Record ID: N/A
Default key: Yes
Private key algorithm: rsaEncryption
Private key size: 1024
Private key type: Secure
Subject name: Certificate with secure private key
              ID
              IBM
              Endicott
              NY
              US

Press ENTER to continue.
===>
```

Figure 43. Token key information menu of a certificate with a secure private key

```

Token key information

Label: Sample RSA Certificate 2
Record ID: N/A
Issuer Record ID: N/A
Default key: Yes
Private key algorithm: rsaEncryption
Private key size: 1024
Private key type: Clear
Subject name: Certificate with clear private key
              ID
              IBM
              Endicott
              NY
              US

Press ENTER to continue.
====>

```

Figure 44. Token key information menu of a certificate with a clear private key

Note: For a z/OS PKCS #11 certificate, the Record ID and Issuer Record ID is 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 that are 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 15 on page 540), choose **1**, (Manage keys and certificates), and on the **Key and Certificate List** (see Figure 35 on page 555, choose the label number you want to work with. The **Key and Certificate Menu** or **Token Key and Certificate Menu** displays:

```

Key and Certificate Menu

Label: My Server Certificate

1 - Show certificate information
2 - Show key information
3 - Set key as default
4 - Set certificate trust status
5 - Copy certificate and key to another database
6 - Export certificate to a file
7 - Export certificate and key to a file
8 - Delete certificate and key
9 - Change label
10 - Create a signed certificate and key
11 - Create a certificate renewal request

0 - Exit program

Enter option number (press ENTER to return to
previous menu): 3 <enter>

Default key set.

Press ENTER to continue.
====>

```

Figure 45. Marking a certificate (and private key) as the default certificate-Key and Certificate Menu

```

Token Key and Certificate Menu

Label: My Server Certificate

1 - Show certificate information
2 - Show key information
3 - Set key as default
4 - Set certificate trust status
5 - Copy certificate and key to another
  database/token
6 - Export certificate to a file
7 - Export certificate and key to a file
8 - Delete certificate and key
9 - Change label
10 - Create a signed certificate and key
11 - Create a certificate renewal request

0 - Exit program

Enter option number (press ENTER to return to the
previous menu): 3 <enter>

Default key set.

Press ENTER to continue.
====>

```

Figure 46. Marking a certificate (and private key) as the default certificate-Token Key and Certificate Menu

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 are created, it might 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 might 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 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:

```
Export File Format

1 - Binary ASN.1 DER
2 - Base64 ASN.1 DER
3 - Binary PKCS #7
4 - Base64 PKCS #7

Select export format (press ENTER to return to menu): 1 <enter>
Enter export file name (press ENTER to return to menu): expfile.der <enter>

Certificate exported.

Press ENTER to continue.
===>
```

Figure 47. Copying a Certificate Without its Private Key

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 about 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 571). 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 servers certificate, while a server with the imported certificate can validate the clients certificate when client authentication is requested.

You must also determine if 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 about setting the default certificate, see “Marking a certificate (and private key) as the default certificate” on page 558.

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:

```
Export File Format
1 - Binary PKCS #12 Version 1
2 - Base64 PKCS #12 Version 1
3 - Binary PKCS #12 Version 3
4 - Base64 PKCS #12 Version 3
Select export format (press ENTER to return to menu): 3 <enter>
Enter export file name (press ENTER to return to menu): expfile.p12 <enter>
Enter export file password (press ENTER to return to menu): <enter password>
Re-enter export file password: <enter password>
Enter 1 for strong encryption, 0 for export encryption: 1 <enter>
Certificate and key exported.
Press ENTER to continue.
====>
```

Figure 48. Copying a Certificate and Private key to a Different Key Database-Export File Format

```
Export File Format
1 - Binary PKCS #12 Version 3
2 - Base64 PKCS #12 Version 3
Select export format (press ENTER to return to menu): 1 <enter>
Enter export file name (press ENTER to return to menu): expfile.p12 <enter>
Enter export file password (press ENTER to return to menu): <enter password>
Re-enter export file password: <enter password>
Enter 1 for strong encryption, 0 for export encryption: 1 <enter>
Certificate and key exported.
Press ENTER to continue.
====>
```

Figure 49. Copying a Certificate and Private key to a Different Key Database-Export File Format

The second display applies to z/OS PKCS #11 tokens.

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. 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 are asked for a file name and password. You then 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 about receiving the certificate into the key database file, see “Importing a certificate from a file with its private key” on page 573). 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 from a key database on the same

system: To copy a certificate and its private key from one key database to another key database or z/OS PKCS #11 token on the same system, you need to know the target key database file name and password, or the z/OS PKCS #11 token name. If the target database is a FIPS database, then the source database must also be a FIPS database. Copying into a FIPS database from a non-FIPS database or z/OS PKCS #11 token is not supported. If the target database is a non-FIPS database or z/OS PKCS #11 token, then the source may be a non-FIPS database, a FIPS database, or a z/OS PKCS #11 token. 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 or z/OS PKCS #11 token.

```
Key and Certificate Menu

Label: newimp

1 - Show certificate information
2 - Show key information
3 - Set key as default
4 - Set certificate trust status
5 - Copy certificate and key to another database/token
6 - Export certificate to a file
7 - Export certificate and key to a file
8 - Delete certificate and key
9 - Change label
10 - Create a signed certificate and key
11 - Create a certificate renewal request

0 - Exit program

Enter option number (press Enter to return to previous menu): 5 <enter>
Enter 1 to specify token name or
  2 to specify database name
  (press ENTER to return to menu): 2 <enter>
Enter key database name (press Enter to return to previous menu): target.kdb <enter>
Enter database password (press Enter to return to previous menu): <enter password>

Record copied.

Press ENTER to continue.
===>
```

Figure 50. Copying a Certificate with its Private Key to a Key Database on the Same System

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.

Note: When a certificate with a key marked as default is copied from a key database into another token or database, it is not marked as the default key in that token or database.

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 need to know the targets file name and password.

```
Token Key and Certificate Menu

Label: newimp

1 - Show certificate information
2 - Show key information
3 - Set key as default
4 - Set certificate trust status
5 - Copy certificate and key to another database/token
6 - Export certificate to a file
7 - Export certificate and key to a file
8 - Delete certificate and key
9 - Change label
10 - Create a signed certificate and key
11 - Create a certificate renewal request

0 - Exit program

Enter option number (press ENTER to return to previous menu): 5 <enter>

Enter 1 to specify token name or
2 to specify database name
  (press ENTER to return to menu): 1 <enter>
Enter token name (press ENTER to return to menu): TOKENDEF <enter>

Record copied.

Press ENTER to continue.
===>
```

Figure 51. Copying a Certificate with its Private Key to a z/OS PKCS #11 Token on the Same System

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 51 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.

Note: When a certificate with a key marked as default is copied from a key database into another token or database, it is not marked as the default key in that token or database.

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 that 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 52), choose **8** to delete the certificate and key.

```

Key and Certificate Menu

Label: newimp

1 - Show certificate information
2 - Show key information
3 - Set key as default
4 - Set certificate trust status
5 - Copy certificate and key to another database
6 - Export certificate to a file
7 - Export certificate and key to a file
8 - Delete certificate and key
9 - Change label
10 - Create a signed certificate and key
11 - Create a certificate renewal request

0 - Exit program

Enter option number (press ENTER to return to previous menu): 8 <enter>
Enter 1 to confirm delete, 0 to cancel delete: 1 <enter>

Record deleted.

Press ENTER to continue.
====>

```

Figure 52. Delete Certificate and Key-Key and Certificate Menu

```

Token Key and Certificate Menu

Label: newimp

1 - Show certificate information
2 - Show key information
3 - Set key as default
4 - Set certificate trust status
5 - Copy certificate and key to another database/token
6 - Export certificate to a file
7 - Export certificate and key to a file
8 - Delete certificate and key
9 - Change label
10 - Create a signed certificate and key
11 - Create a certificate renewal request

0 - Exit program

Enter option number (press ENTER to return to previous menu): 8 <enter>
Enter 1 to confirm delete, 0 to cancel delete: 1 <enter>

Record deleted.

Press ENTER to continue.
====>

```

Figure 53. Delete Certificate and Key-Token Key and Certificate Menu

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 54), choose **9** to change the label:

```

Key and Certificate Menu

Label: cacert

1 - Show certificate information
2 - Show key information
3 - Set key as default
4 - Set certificate trust status
5 - Copy certificate and key to another database
6 - Export certificate to a file
7 - Export certificate and key to a file
8 - Delete certificate and key
9 - Change label
10 - Create a signed certificate key
11 - Create a certificate renewal request

0 - Exit program

Enter option number (press ENTER to return to previous menu): 9 <enter>
Enter label (press ENTER to return to menu): cacert2 <enter>

Label changed.

Press ENTER to continue.
====>

```

Figure 54. Changing a Certificate Label-Key and Certificate Menu

```

Token and Certificate Menu

Label: cacert

1 - Show certificate information
2 - Show key information
3 - Set key as default
4 - Set certificate trust status
5 - Copy certificate and key to another database/token
6 - Export certificate to a file
7 - Export certificate and key to a file
8 - Delete certificate and key
9 - Change label
10 - Create a signed certificate key
11 - Create a certificate renewal request

0 - Exit program

Enter option number (press ENTER to return to previous menu): 9 <enter>
Enter label (press ENTER to return to menu): cacert2 <enter>

Label changed.

Press ENTER to continue.
====>

```

Figure 55. Changing a Certificate Label-Token and Certificate Menu

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 fast path 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**. 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 56), choose option **10** to create a signed certificate and key.

Note: This requires the displayed certificate to have signing capability.

```
Key and Certificate Menu
Label: cacert
1 - Show certificate information
2 - Show key information
3 - Set key as default
4 - Set certificate trust status
5 - Copy certificate and key to another database
6 - Export certificate to a file
7 - Export certificate and key to a file
8 - Delete certificate and key
9 - Change label
10 - Create a signed certificate and key
11 - Create a certificate renewal request
0 - Exit program
Enter option number (press ENTER to return to
previous menu): 10 <enter>
====>
```

Figure 56. Select 10 to Create a Signed Certificate and Key-Key and Certificate Menu

```
Token Key and Certificate Menu
Label: cacert
1 - Show certificate information
2 - Show key information
3 - Set key as default
4 - Set certificate trust status
5 - Copy certificate and key to another database/token
6 - Export certificate to a file
7 - Export certificate and key to a file
8 - Delete certificate and key
9 - Change label
10 - Create a signed certificate and key
11 - Create a certificate renewal request
0 - Exit program
Enter option number (press ENTER to return to
previous menu): 10 <enter>
====>
```

Figure 57. Select 10 to Create a Signed Certificate and Key-Token Key and Certificate Menu

The **Certificate Usage** menu appears, followed by menus to select the certificate key algorithm and key size (or ECC key type and EC named curve if ECC is selected as the certificate key algorithm. See “Creating a signed ECC certificate and key” on page 566.) Once these details are 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.

```
Certificate Usage
1 - CA certificate
2 - User or server certificate

Select certificate usage (press ENTER to return to menu): 2 <enter>

Certificate Key Algorithm
1 - Certificate with an RSA key
2 - Certificate with a DSA key
3 - Certificate with an ECC key
4 - Certificate with a Diffie-Hellman key

Select certificate key algorithm (press ENTER to return to menu): 1 <enter>

RSA Key Size
1 - 1024-bit key
2 - 2048-bit key
3 - 4096-bit key

Select RSA key size (press ENTER to return to menu): 1 <enter>
Enter label (press ENTER to return to menu): signedcert <enter>
Enter subject name for certificate
Common name (required): My signed Certificate <enter>
Organizational unit (optional): ID <enter>
Organization (required): IBM <enter>
City/Locality (optional): Endicott <enter>
State/Province (optional): NY <enter>
Country/Region (2 characters - required): US <enter>
Enter number of days certificate will be valid (default 365): 300 <enter>

Enter 1 to specify subject alternate names or 0 to continue: 1

Please wait .....
```

Figure 58. Enter Certificate Details

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.

```
Subject Alternate Name Type
  1 - Directory name (DN)
  2 - Domain name (DNS)
  3 - E-mail address (SMTP)
  4 - Network address (IP)
  5 - Uniform resource identifier (URI)

Select subject alternate name type (press ENTER if name is complete): 1 <enter>
Enter subject name for certificate
Common name (required): Test server <enter>
Organizational unit (optional): ID <enter>
Organization (required): IBM <enter>
City/Locality (optional): Endicott <enter>
State/Province (optional): NY <enter>
Country/Region (2 characters - required): US <enter>

      Subject Alternate Name Type
        1 - Directory name (DN)
        2 - Domain name (DNS)
        3 - E-mail address (SMTP)
        4 - Network address (IP)
        5 - Uniform resource identifier (URI)

Select subject alternate name type (press ENTER if name is complete): <enter>
Please wait .....
```

Figure 59. Subject Alternate Name Type

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 signed ECC certificate and key

If ECC is selected as the certificate key algorithm in the **Certificate Key Algorithm menu**, you are prompted to choose the ECC key type (for user or server certificates only) to be set in the new certificate and the EC named curve to be used when generating the ECC key. Supported EC named curves are outlined in “Elliptic Curve Cryptography support” on page 14.

The following example creates an end-entity certificate with an ECDSA key using a 256-bit NIST suggested named curve.

```

Certificate Usage
1 - CA certificate
2 - User or server certificate

Select certificate usage (press ENTER to return to menu): 2 <enter>

Certificate Key Algorithm
1 - Certificate with an RSA key
2 - Certificate with a DSA key
3 - Certificate with an ECC key
4 - Certificate with a Diffie-Hellman key

Select certificate key algorithm (press ENTER to return to menu): 3 <enter>

ECC Key Type
1 - General ECC key
2 - ECDSA Key
3 - ECDH key

Select ECC key type (press ENTER to return to menu): 2 <enter>

```

Figure 60. Selecting the ECC Key Type

The selected key type determines the setting of the keyUsage extension in the new certificate. A general ECC key allows Digital Signature, Non-repudiation and Key Agreement. An ECDSA key allows Digital Signature and Non-repudiation. An ECDH key allows Key Agreement only.

If option 1 is selected in the **Certificate Usage menu**, requesting a CA certificate, the **ECC Key Type menu** does not appear. The keyUsage extension of the new certificate is set to allow the certificate to be used to sign certificates and certificate revocation lists.

|
|

Once the key type has been selected, you are prompted to select the ECC curve type. For a FIPS database, Brainpool standard curves are not supported.

```
ECC Curve Type
1 - NIST recommended curve
2 - Brainpool standard curve
Select ECC curve type (press ENTER to return to menu): 1 <enter>

NIST Recommended Curve Type
1 - secp192r1
2 - secp224r1
3 - secp256r1
4 - secp384r1
5 - secp521r1
Select NIST recommended curve (press ENTER to return to menu): 3 <enter>

Enter label (press ENTER to return to menu): signedECCcert <enter>
Enter subject name for certificate
Common name (required): My signed ECC Certificate <enter>
Organizational unit (optional): ID <enter>
Organization (required): IBM <enter>
City/Locality (optional): Endicott <enter>
State/Province (optional): NY <enter>
Country/Region (2 characters - required): US <enter>
Enter number of days certificate will be valid (default 365): 300 <enter>

Enter 1 to specify subject alternate names or 0 to continue: 0 <enter>

Please wait .....

Certificate created.

Press ENTER to continue.
```

Figure 61. Selecting the ECC Curve Type

For a FIPS database, some curves may not be recommended for use and may not appear in the **ECC Curve Type menu**. After selecting the curve type you are prompted to enter the certificate label, subject name, expiration and (optionally) subject alternate names. See “Creating a signed certificate and key” on page 564 for more information.

Creating a certificate to be used with a fixed Diffie-Hellman key exchange

Create a server certificate to be used during an SSL handshake using a fixed Diffie-Hellman key exchange. Fixed Diffie-Hellman requires the certificates being used by both sides of the exchange to be based off 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**. You are asked to select the key type and key size. Only 1024-bit DSA keys, 2048-bit DSA keys, or 2048-bit fixed Diffie-Hellman keys are valid for use in a FIPS database. When the key type is determined, you are 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.

```

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: 6 <enter>

```

```

Key Type
1 - DSA key
2 - Diffie-Hellman key
Select key type (press ENTER to return to menu): 2 <enter>

Diffie-Hellman Key Size
1 - 1024-bit key
2 - 2048-bit key
Select Diffie-Hellman key size (press ENTER to return to menu): 1 <enter>
Enter key parameter file name (press ENTER to return to menu): dh_key_1024.keyfile <enter>

Please wait .....
Key parameter file created.
Press ENTER to continue

```

Figure 62. Creating a key parameter file to be used with Diffie-Hellman

When the key parameter file is created, the next step is to create the signed certificate by using an existing certificate in the key database file or z/OS PKCS #11 token to sign the server certificate. From the **Key Management Menu** or **Token Management Menu**, select **1 - Manage keys and certificates** to display the **Key and Certificate List**. From the **Key and Certificate List**, select a CA certificate by entering the appropriate selection number, and then choose option **10** to create a signed certificate and key. This requires the displayed certificate to contain an RSA or a DSA key and have signing capability.

Select "User or server certificate" by choosing option **2** in the **Certificate Usage menu**, followed by option **4 - Certificate with a Diffie-Hellman key** in the **Certificate Key Algorithm menu**, and then select the Diffie-Hellman key size. The key size must match the key size of the key parameters created previously.

When the certificate type is determined, you are 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).

```

Certificate Usage

 1 - CA certificate
 2 - User or server certificate

Select certificate usage (press ENTER to return to menu): 2 <enter>

Certificate Key Algorithm

 1 - Certificate with an RSA key
 2 - Certificate with a DSA key
 3 - Certificate with an ECC key
 4 - Certificate with a Diffie-Hellman key

Select certificate key algorithm (press ENTER to return to menu): 4 <enter>

Diffie-Hellman Key Size

 1 - 1024-bit key
 2 - 2048-bit key

Select key size (press ENTER to return to menu): 1 <enter>
Enter key parameter file name (press ENTER to return to menu): dh_key_1024.keyfile <enter>
Enter label (press ENTER to return to menu): DSA_cert_with_DH_1024_key <enter>
Enter subject name for certificate:
  Common name (required): DSA cert with DH 1024 key <enter>
  Organizational unit (optional): Test <enter>
  Organization (required): Test <enter>
  City/Locality (optional): Poughkeepsie <enter>
  State/Province (optional): NY <enter>
  Country/Region (2 characters - required): US <enter>
Enter number of days certificate will be valid (default 365): 5000 <enter>

Enter 1 to specify subject alternate names or 0 to continue: 0 <enter>

Please wait .....

Certificate created.

Press ENTER to continue.

```

Figure 63. Creating a certificate to be used with Diffie_Hellman

When the certificate is created, the next step is to determine if the certificate must be transferred to another database. If the certificate does not need to reside elsewhere, you must 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 about setting the default certificate, see “Marking a certificate (and private key) as the default certificate” on page 558. If the certificate must be transferred, see “Copying a certificate (and private key) to a different key database or z/OS PKCS #11 token” on page 559 for more information.

Creating a certificate renewal request

Certificate renewal requests allow for existing signed certificates that have expired or 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 and 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**. 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 64) choose option **11** to create a certificate renewal request.

```

Key and Certificate Menu

Label: cacert

1 - Show certificate information
2 - Show key information
3 - Set key as default
4 - Set certificate trust status
5 - Copy certificate and key to another database
6 - Export certificate to a file
7 - Export certificate and key to a file
8 - Delete certificate and key
9 - Change label
10 - Create a signed certificate key
11 - Create a certificate renewal request

0 - Exit program

Enter option number (press ENTER to return to previous menu): 11 <enter>
Enter request filename (press ENTER to return to menu): renewal.arm <enter>

Certificate request created.

Press ENTER to continue.
====>

```

Figure 64. Select 11 to Create a Certificate Renewal Request-Key and Certificate Menu

```

Token Key and Certificate Menu

Label: cacert

1 - Show certificate information
2 - Show key information
3 - Set key as default
4 - Set certificate trust status
5 - Copy certificate and key to another database/token
6 - Export certificate to a file
7 - Export certificate and key to a file
8 - Delete certificate and key
9 - Change label
10 - Create a signed certificate key
11 - Create a certificate renewal request

0 - Exit program

Enter option number (press ENTER to return to previous menu): 11 <enter>
Enter request filename (press ENTER to return to menu): renewal.arm <enter>

Certificate request created.

Press ENTER to continue.
====>

```

Figure 65. Select 11 to Create a Certificate Renewal Request-Token Key and Certificate Menu

Enter the request file name (press ENTER to return to menu). The certificate request is created. Press enter to continue. After creating the certificate renewal request, perform the following steps:

1. If you want a certificate authority (CA) to sign the certificate, send the certificate request to the CA. See “Sending the certificate request” on page 553. If you are acting as your own CA, use the **gskkyman** command line interface to sign the certificate. See “Using gskkyman to be your own certificate authority (CA)” on page 574.
2. Receive the renewed certificate into your key database. See “Receiving the signed certificate or renewal certificate” on page 553.

Importing a certificate from a file as a trusted CA certificate

If you are using a certificate authority for generating your certificates that are 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 that is imported regardless of whether 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 that is 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 before the import. If the CA certificates chain consists of more than one certificate and the certificates exist in individual files, you must import the certificates starting with the root CA certificate.

If you are 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**. The following figures contain lists of CAs for which certificates are stored on key database creation:

```
Certificate List

Database: /home/sufwl1/ssl_cmd/mykey.kdb

1 - VeriSign Class 1 Public Primary CA
2 - VeriSign Class 2 Public Primary CA
3 - VeriSign Class 3 Public Primary CA
4 - Thawte Server CA
5 - Thawte Premium Server CA
6 - Thawte Personal Basic CA
7 - Thawte Personal Freemail CA
8 - Thawte Personal Premium CA
9 - Equifax Secure Certificate Authority

0 - Return to selection menu

Enter label number (ENTER for more labels, p for previous list):
===>
```

Figure 66. Certificate List (part 1)

```
Certificate List

Database: /home/sufwl1/ssl_cmd/mykey.kdb

1 - Equifax Secure eBusiness CA-1
2 - Equifax Secure eBusiness CA-2
3 - Equifax Secure Global eBusiness CA-1
4 - VeriSign Class 1 Public Primary CA - G2
5 - VeriSign Class 2 Public Primary CA - G2
6 - VeriSign Class 3 Public Primary CA - G2
7 - VeriSign Class 4 Public Primary CA - G2
8 - VeriSign Class 1 Public Primary CA - G3
9 - VeriSign Class 2 Public Primary CA - G3

0 - Return to selection menu

Enter label number (ENTER to return to selection menu, p for previous list):
===>
```

Figure 67. Certificate List (part 2)

```

Certificate List

Database: /home/sufw11/ssl_cmd/mykey.kdb

1 - VeriSign Class 3 Public Primary CA - G3
2 - VeriSign Class 4 Public Primary CA - G3
3 - VeriSign Class 3 Public Primary CA - G5

0 - Return to selection menu

Enter label number (ENTER to return to selection menu, p for previous list):
====>

```

Figure 68. Certificate List (part 3)

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:

```

Key Management Menu

Database: /home/sufw11/ssl_cmd/mykey.kdb
Expiration Date: 2025/12/02 10:11:12
Type: non-FIPS

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Store database password
11 - Show database record length

0 - Exit program

Enter option number (press ENTER to return to previous menu): 7 <enter>
Enter import file name (press ENTER to return to menu): cert.arm <enter>
Enter label (press ENTER to return to menu): cacert2 <enter>

Certificate imported.

Press ENTER to continue.
====>

```

Figure 69. Importing a Certificate from a File-Key Management Menu

```

Token Management Menu

Token: TOKENABC

Manufacturer: z/OS PKCS11 API
Model: HCR77B0
Flags: x00000509 (INITIALIZED, PROT AUTH PATH, USER PIN INIT, RNG)

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Delete Token

0 - Exit program

Enter option number (press ENTER to return to previous menu): 7 <enter>
Enter import file name (press ENTER to return to menu): cert.arm <enter>
Enter label (press ENTER to return to menu): cacert2 <enter>

Certificate imported.

Press ENTER to continue.
====>

```

Figure 70. Importing a Certificate from a File-Token Management Menu

You are prompted to enter the certificate file name and your choice of a unique label that are assigned to the certificate.

When the certificate is imported, you receive a message that indicates the import was successful. The certificate is treated as "trusted" so that it can be used in verifying incoming certificates. For a program that is acting as an SSL server, this certificate is used during the verification of a client's certificate. For a program that is acting as an SSL client, this certificate is used to verify the server's certificate that 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 560

for more information). PKCS #12 files are password-protected to allow encryption of the private key information. If the CA certificate that is 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 before the import. From the **Key Management Menu** or **Token Management Menu**, enter 8 to import a certificate and a private key:

```

Key Management Menu

Database: /home/sufw11/ssl_cmd/anne.kdb
Expiration Date: 2025/12/02 10:11:12
Type: non-FIPS

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Store database password
11 - Show database record length

0 - Exit program

Enter option number (press ENTER to return to previous menu): 8 <enter>
Enter import file name (press ENTER to return to menu): cert.p12 <enter>
Enter import file password (press ENTER to return to menu): <enter password>
Enter label (press ENTER to return to menu): newcert <enter>

Certificate and key imported.

Press ENTER to continue.
====>

```

Figure 71. Importing a Certificate and Private Key from a File-Key Management Menu

```

Token Management Menu

Token: TOKENABC

Manufacturer: z/OS PKCS11 API
Model: HCR77B0
Flags: x00000509 (INITIALIZED, PROT AUTH PATH, USER PIN INIT, RNG)

1 - Manage keys and certificates
2 - Manage certificates
3 - Manage certificate requests
4 - Create new certificate request
5 - Receive requested certificate or a renewal certificate
6 - Create a self-signed certificate
7 - Import a certificate
8 - Import a certificate and a private key
9 - Show the default key
10 - Delete Token

0 - Exit program

Enter option number (press ENTER to return to previous menu): 8 <enter>
Enter import file name (press ENTER to return to menu): cert.p12 <enter>
Enter import file password (press ENTER to return to menu): <enter password>
Enter label (press ENTER to return to menu): newcert <enter>

Certificate and key imported.

Press ENTER to continue.
====>

```

Figure 72. Importing a Certificate and Private Key from a File-Token Management Menu

You are 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 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 tokens 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 about setting the default certificate, see “Marking a certificate (and private key) as the default certificate” on page 558).

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 terminates if a non-FIPS certificate is encountered. Certificates that are processed before the failing certificate is 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 to maintain adherence to the FIPS criteria.

Using gskkyman to be your own certificate authority (CA)

The **gskkyman** utility provides the capability for you to act as your own Certificate Authority (CA). If your own CA, you are authorized to sign certificate requests for yourself or others. This is convenient if you 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 wants 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 that are needed to become your own CA to allow secure communication between a client and a server. This example reflects the steps that are followed when the CA is on a different system or is a different user than the issuer of the certificate request.

Certificate Authority (System A)	Server or Client (System B)
Step 1 - Create a key database	
Create a key database using the gskkyman utility: <ul style="list-style-type: none"> • From the Database Menu, select option 1 - Create new database See "Creating, opening, and deleting a key database file" on page 538 for details.	Create a key database using the gskkyman utility: <ul style="list-style-type: none"> • From the Database Menu, select option 1 - Create new database See "Creating, opening, and deleting a key database file" on page 538 for details.
Step 2 - Create a Root Certificate Authority certificate	
Create a Certificate Authority certificate: <ul style="list-style-type: none"> • From the Key Management Menu, select option 6 - Create a self-signed certificate • From the Certificate Usage menu, select option 1 - CA certificate See "Creating a self-signed server or client certificate" on page 547 for details.	No action required.
Step 3 - Create a certificate request	
No action required.	Create a certificate request: <ul style="list-style-type: none"> • From the Key Management Menu, select option 4 - Create new certificate request See "Creating a certificate request" on page 550 for details.
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 553.
Step 5 - Sign the certificate request	

Certificate Authority (System A)	Server or Client (System B)
<p>Before signing a certificate for a client or server, you must make sure that the requester has a legitimate claim to request the certificate. After verifying the claim, you can create a signed certificate.</p> <p>To sign the certificate request, the gskkyman utility must be issued using command-line options (see “gskkyman command line mode syntax” on page 577 for a description of the options). The gskkyman utility must be issued with these parameters:</p> <pre>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</pre> <p>Example: This command allows you to sign a request certificate and allow the certificate to be valid for 360 days.</p> <pre>gskkyman -g -x 360 -cr server_request.arm -ct server_signed_cert.arm -k CA.kdb -l labelname</pre> <p>After you entered the command, you are prompted to enter the database password.</p> <p>Note:</p> <ol style="list-style-type: none"> 1. The signed certificate is an end user certificate unless the -ca option is specified. 2. The file name that is 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 is set to the end date for the CA certificate if the requested certificate lifetime exceeds the CA certificate lifetime. 	<p>No action required.</p>
Step 6 - Send the signed CA certificate and the newly signed certificate to the requester	
<p>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 559. Send (for example, without its private key FTP) the Base64 file and the newly signed certificate (created in Step 4) to the requester.</p>	<p>No action required.</p>
Step 7 - Import the CA certificate	
<p>No action required.</p>	<p>Import the CA certificate. See “Importing a certificate from a file as a trusted CA certificate” on page 571.</p>
Step 8 - Receive the signed certificate	

Certificate Authority (System A)	Server or Client (System B)
No action required.	Receive the signed certificate. See “Receiving the signed certificate or renewal certificate” on page 553. Note: Depending upon the SSL application, you might have to either send the CA certificate to the client, or the server application might 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**. See “Copying a certificate with its private key” on page 560 for details about 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**. See “Importing a certificate from a file with its private key” on page 573 for more information.

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**. See “Copying a certificate with its private key” on page 560 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 data set.
3. Use the RACDCERT command with the ADD operand and the data set 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** utility is used for key database management, z/OS PKCS #11 token management, and to display the certificates within a PKCS #12 file.

Format

gskkyman

```
gskkyman -dc|-dcv [-k filename|-t tokename|-p12 filename] [-l label]
gskkyman -dk [-k filename]
gskkyman -e|-i [-k filename|-t tokename] [-l label] [-p filename]
```

```

gskkyman -g [-x days] [-cr filename] [-ct filename] [-k filename] [-t tokename]
          [-l label] [-kt {ecgen|ecdsa|ecdh}] [-ca] [-ic]
gskkyman -h|-?
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, record length and type.
- 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.
- ?** Display the command syntax.

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 is generated if **-ca** is specified. An end user certificate is generated if **-ca** is not specified.
- cr** Specifies the name of the certificate request file. You are prompted for the file name if this option is not specified.
- ct** Specifies the name of the output generated signed certificate file. You are 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 is overwritten.
- ic** The certification chain certificates are included in the certificate file if **-ic** is specified. Otherwise, just the signed certificate is included in the certificate file.
- k** Specifies the name of the key database. This option is mutually exclusive with the **-t** option and the **-p12** option. You are prompted for the key database file name if either this option or the **-t** option or the **-p12** option is 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**.
- kt** Specifies the key type of the certificate to be created. This option is valid

when signing an end user certificate or certificate request containing an ECC public key and affects the settings of the keyUsage extension of the certificate created. Valid key type options are ecgen, ecdsa and ecdh. ecgen creates a certificate with digitalSignature, nonRepudiation and keyAgreement set, ecdsa creates a certificate with digitalSignature and nonRepudiation set, and ecdh creates a certificate with keyAgreement set. If the **-kt** option is not specified for an end user ECC certificate or certificate request, the default option is ecgen. For other certificate types the **-kt** option is ignored.

- l** Specifies the certificate label. The label must be enclosed in double quotation marks if it contains one or more spaces. If the certificate is being used to sign a certificate request (sign function), the certificate must be a CA. The label for the default key is used if this option is not specified (export or sign function) or you are 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 to be used to import or export certificates. You are prompted for the file name if this option is not specified.
- p12**
Specifies the name of the PKCS #12 file containing the certificates to be displayed. This option is mutually exclusive with the **-k** option and the **-t** option. 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. Lastly, the PKCS #12 file cannot end with .kdb, .rdb or .sth.
- t** Specifies the name of the token to be managed. This option is mutually exclusive with the **-k** option and the **-p12** 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 allowed, 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 expires 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** utility is used to manage a token, a key database and its associated request database, or to list the contents of a PKCS #12 file. Interactive menus are displayed if no command options are specified. Otherwise, the requested token/database/PKCS #12 file function is performed and the **gskkyman** utility exits.

Note: The ability to display the contents of a PKCS #12 file is not supported through the interactive menu-driven interface.

If the command specifies the **-t** (token name) option, then the requested function is performed for the identified token. If the specified PKCS #11 token certificate contains a secure private key, then only display functions **-dc** and **-dcv** are supported. If the **gskkyman** utility supplies both the **-t** and **-l** (label name) options, then only the PKCS #11 certificate with the matching label is checked for a secure private key. If the certificate does not have a secure private key, then both the **-e** (export) or **-g** (sign) functions can be processed.

If the command specifies the **-p12** (PKCS #12 file) option on the display functions **-dc** or **-dcv**, if **-l** option is used, the certificate with matching label is displayed. If **-l** option is not used, all certificates within the file are displayed.

If the command does not specify the **-t** or the **-p12** option, then it is assumed that the function is to be performed for a key database. If the **-k** option, the **-t** option, and the **-p12** option are not supplied, the user is prompted for a key database file name.

If any combination of **-k**, **-t**, and **-p12** is 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 file name extension of **.kdb**. The request database contains requests for new certificates and always has a file name extension of **.rdb**. The database stash file contains the masked database password and always has a file name 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 is 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 is not deleted if an error occurs while rewriting the database file. If this happens, you can replace the database file with the temporary database file to recover from the error. If an error does occur and you do not rename or delete the temporary file, you receive an error on the next database update operation indicating the backup file 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 that is returned from the token so that certificates with private keys might be interspersed with certificates without private keys.

gskkyman command line mode examples

Command mode is entered when the **gskkyman** utility is entered with parameters. The requested token/database function is performed and then the utility exits.

- 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 are 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 that is identified by the record label are exported to a file in PKCS #12 Version 3 format using strong encryption. The default key is exported if the '-l' option is not specified. You are prompted for the key database file name if the '-k' and the '-t' option is not specified. You are 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 are prompted for the label if the '-l' option is not specified. You are prompted for the key database file name if the '-k' and the '-t' option is not specified. You are 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 -kt keytype -ca -ic
```

The certificate request that is 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 are prompted for the key database file name if the '-k' option is not specified. You are prompted for the certificate request file name if the '-cr' option is not specified. You are prompted for the signed certificate file name if the '-ct' option is not specified.

The signed certificate is an end user certificate unless the -ca option is specified. A certification authority certificate has basic constraints and key usage extensions that allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate has basic constraints and key usage extensions that allow the certificate to be used 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.
- An ECC key depends on the keytype option supplied. A general ECC key (-kt ecgen) can be used for authentication, digital signature, and key agreement. An ECDSA key (-kt ecdsa) can be used for authentication and digital signature. An ECDH key (-kt ecdh) can be used for key agreement. The default option is ecgen.

Any certificate can be used to sign the new certificate if 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 then used, it might fail the validation checking if the signing certificate is not a valid certification authority certificate.

The signature algorithm that are used to sign the new certificate is based on the key algorithm of the signing certificate. An RSA signature uses the most secure and compatible SHA-based hash in use in the signature algorithm of either the signing certificate or the certificate request. A DSA signature with a 1024-bit DSA

key uses SHA-1. A DSA signature with a 2048-bit DSA key uses SHA-256. An ECC signature uses the suggested digest for the key size of the ECC private key, as specified in Table 2 on page 15.

Possible signature algorithms are:

- x509_alg_sha1WithRsaEncryption
- x509_alg_sha224WithRsaEncryption
- x509_alg_sha256WithRsaEncryption
- x509_alg_sha384WithRsaEncryption
- x509_alg_sha512WithRsaEncryption
- x509_alg_dsaWithSha1
- x509_alg_dsaWithSha256
- x509_alg_ecdsaWithSha256
- x509_alg_ecdsaWithSha384
- x509_alg_ecdsaWithSha512

The certificate file contains the generated X.509 certificate in DER-encoded Base64 format if the `-ic` option is not specified. The certificate file contains 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 you are prompted for the key database password, the certificates will be displayed. You are prompted for the key database file name if the `-k` option is not specified. Because of the number of certificates that can exist in a key database file, it is suggested that you redirect the output to a file. This allows for easy review of the certificates and any post-processing of the certificate output.

- Display key database expiration date:

```
gskkyman -dk -k filename
```

After you are prompted for the key database password, the full key database path and file name, expiration date, record length, and database type are displayed. You are prompted for the key database file name if the `-k` option is not specified.

gskkyman command line mode displays

Command mode is entered when **gskkyman** is entered with parameters. The requested token/database function is performed and then the utility exits.

- **gskkyman** command-mode key database file display

When the key database password is correctly entered:

Command:

```
gskkyman -dk -k example.kdb
```

Output:

```
Database: /home/sufw11/ssl_cmd/example.kdb
Expiration Date: 2025/12/02 10:11:12
Record length: 5000
Type: non-FIPS
```

- **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:
    45ac4d23000a6023
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:
    2015/01/16 21:02:02
Expiration date:
    2020/01/16 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
    28 93 7C AE 12 E8 CD 55 16 E1 05 53 63 C1 84 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
Certificate extensions:
    4

```

- **gskkyman** command-mode PKCS #11 token certificate display

Command:

```
gskkyman -dc -t my.token -l rsa1024CASecure
```

Output:

```

Label:
    <rsa1024CASecure>
Trusted:
    Yes
Version:
    3
Serial number:
    00
Issuer's Name:
    <CN=rsa CA linecmd 1,OU=ibm,O=stg,C=US>
Subject's Name:
    <CN=rsa CA linecmd 1,OU=ibm,O=stg,C=US>
Effective Date:
    2012/06/06 04:00:00
Expiration Date:
    2025/11/01 03:59:59
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 A8 CF 98 A5 EE A9 F3 FD 59
A6 6F F8 F1 CF 85 00 26 DA D3 04 52 EA E0 94 62
B4 DB 32 FC A7 AE E8 BF 1C 0B 6B A8 78 25 BF D4
9C BE 1E 15 8C 37 36 F2 94 E9 5F 58 8B CB CB BB
FA AF 47 BD 5D BA 77 C2 B6 8B 15 91 C7 5A B1 28
62 BB 23 80 80 50 DB 2F 49 38 9C B6 4D 0E 2F EC
87 63 E5 AE 99 EC 9D 87 A7 94 D4 BF EA A1 0E F0
00 56 C7 A6 9E 25 18 BF F6 2F 7B D4 E1 C4 91 E4
9F F0 50 DE 3D 94 3D 02 03 01 00 01
Private key:
Yes
Private key type:
Secure
Default key:
Yes
Certificate extensions:
3

```

- **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:
45ac4d23000a6023
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:
2015/01/16 21:02:02
Expiration date:
2020/01/16 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
28 93 7C AE 12 E8 CD 55 16 E1 05 53 63 C1 84 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:
      <test@ibm.com>
Non-critical Extension: 2
  subjectKeyIdentifier:
    91 DA 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 reads 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. Blank 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 is 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 is 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 is 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 is 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 that is 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 19 for setup requirements necessary to execute in FIPS mode.

To have GSKSRVR execute in non-FIPS mode and provide sysplex session ID caching for non-FIPS application servers, remove or comment out this environment variable. GSKSRVR starts 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 (gsksrvr.envar) from /usr/lpp/gskssl/examples/ to /etc/gskssl/server/ with a new file name of "envar". By default, the full path is /etc/gskssl/server/envar (change the directory name to match the home directory created). 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)

Note: The sample started task procedure routes informational messages, such as GSK01001I, to standard out, while error messages, such as GSK01015E are routed to standard error. If you want to route informational and error messages to the same place in the job log, change:

```
// / 1>DD:STDOUT 2>DD:STDERR')
```

to

```
// / >DD:STDOUT 2>&1')
```

4. Create the GSKSRVR user and associate it with the GSKSRVR started procedure. Replace 'nnnnn' in the ADDUSER command with a non-zero value which is not assigned to another user.

```
ADDUSER GSKSRVR DFLTGRP(SYS1) NOPASSWORD OMVS(UID(nnnnn) PROGRAM(/bin/sh) HOME(/etc/gskssl/server))
```

```
RDEFINE STARTED GSKSRVR.** STDATA(USER(GSKSRVR) GROUP(SYS1) TRUSTED(YES))
```

```
SETROPTS RACLIST(STARTED) REFRESH
```

5. Ensure that the pdsename.SIEALNKE and CEE.SCEERUN data sets are APF-authorized and are either in the link list concatenation or are specified as 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. To activate the exit, add this to the appropriate MPFLSTxx member in SYS1.PARMLIB.

```
BPXI004I,SUP(NO),USEREXIT(STARTSSL)
```

This starts GSKSRVR when OMVS initialization is complete, assuming the GSKMSGXT program was linked as STARTSSL and placed in a LNKLIST data set.

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 SYSSL 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 registers 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 you want.
8. If access to the ICSF callable services are protected with CSFSERV class profiles on your system, the GSKSRVR user ID might need to be given READ authority to call the ICSF CSFIQA and CSFPPRF callable services. These services are protected by the CSFIQA and CSFRNG profiles. If these callable services are protected with a generic CSF* profile in the CSFSERV class, access can be granted by entering:

```
PERMIT CSF* CLASS(CSFSERV) ID(GSKSRVR) ACCESS(READ)
SETROPTS RACLIST(CSFSERV) REFRESH
```

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. " " is 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 includes 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. See 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 if 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 higher TLS protocol 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 established a TLS V1.0 or higher TLS protocol 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 does not support the maximum fragment length TLS extension, a full rehandshake occurs.

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 user ID on one system in the sysplex must represent the same user on all other systems in the sysplex (that is, user ID ZED on System A has the same access rights as user ID 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, TLS V1.1, or TLS V1.2 session created by the SSL server is then 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)
SETOPTS RACLIST(FACILITY) REFRESH
```

Component trace support

For information about component trace support, see “Component trace support” on page 592.

Hardware cryptography failure notification

For information about cryptographic hardware failure notification, see Chapter 3, “Using cryptographic features with System SSL,” on page 11.

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 might 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 from the UNIX System Services **export** shell command. For usage information about this command, see the *z/OS UNIX System Services Command Reference*. For information about setting environment variables outside of the shell, see *z/OS XL C/C++ Programming Guide* and the *z/OS Language Environment Programming Guide*.

The following facilities 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 before 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 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

- **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.

Obtaining diagnostic information

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 is /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 might occur. To allow trace information to be obtained, the trace file name specified should contain a '%' character in the file name. This allows the process identifier to be placed within the file name and each process to write to its own trace file.

It is suggested that if the default trace file value is not being used, the trace file name always contain a '%' character. This eliminates the need to know whether 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 data set or an SVC dump of the GSKSRVR address space. Data set *hlq.SIEAMIGE* containing the SSL trace record format routine to be used by IPCS must be accessible through either a steplib or in the lnklist.

Note: The System SSL started task provides component trace support only for SSL applications, therefore, component trace of the System SSL started task itself is not supported. Therefore, the jobname for the System SSL started task should not be used as one of the jobnames in the MVS TRACE command. Tracing for the System SSL started task can be accomplished by setting the GSK_TRACE environment variable.

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 or less are considered to be a wildcard entry and tracing is started for jobs whose names match those entries for the length of the entry.

See *z/OS MVS Diagnosis: Tools and Service Aids* for more information about setting up and using component trace. See *z/OS MVS System Commands* for more information about the TRACE command. See *z/OS MVS IPCS User's Guide* for more information about 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 is 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 is traced. Subsequent jobs with the same name are not 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 data set. A sample started procedure is shipped as GSK.SGSKSAMP(GSKWTR). Copy this procedure to SYS1.PROCLIB(GSKWTR) and modify as necessary to meet your installation requirements. This MVS operator command starts 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 defaults 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 is 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 from 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 stop the SSL component trace and close the trace writer data set:

```
TRACE CT,OFF,COMP=GSKSRVR  
TRACE CT,WTRSTOP=GSKWTR
```

Obtaining diagnostic information

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 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 stops 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.

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, and so on. All trace entries are 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 are 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 GSKSRVR 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
SYSNAME(C01)
COMP(GSKSRVR)
**** 11/14/2005
```

Obtaining diagnostic information

```

SYSNAME  MNEMONIC  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 located 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:

MNEMONIC	ENTRY ID	DESCRIPTION	EXPLANATION
MESSAGE	00000001	SSL_ENTRY	Entry into the function named in the following System SSL header line (i.e. line 2) occurred
MESSAGE	00000002	SSL_EXIT	Exit from the function named in the following System SSL header line occurred
MESSAGE	00000004	SSL_ERROR	Error was detected by the function named in following line 2 with error description in line 3
MESSAGE	00000008	SSL_INFO	Information generated by the function named in following line 2 - for example, supplied parameters
DUMP	00000010	SSL_EBCDIC_DUMP	Dump of buffer contents formatted in EBCDIC, by the function named in following line 2
DUMP	00000020	SSL_ASCII_DUMP	Dump of buffer contents formatted in ASCII, by the function name in following line 2

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 C01      MESSAGE   00000001  20:43:46.694762  SSL_ENTRY
2 Job TCP341 Process 00020032 Thread 00000004 gsk_secure_socket_read
3 Handle 7E828198, Size 1
4 C01      MESSAGE   00000008  20:43:46.695013  SSL_INFO
5 Job TCP341 Process 00020032 Thread 00000004 gsk_read_v3_record
6 Calling read routine for 5 bytes
7 C01      MESSAGE   00000004  20:43:46.695317  SSL_ERROR
8 Job TCP341 Process 00020032 Thread 00000004 gsk_read_v3_record
9 Socket closed by 192.168.50.80.1472.
10 C01     MESSAGE   00000004  20:43:46:695478  SSL_ERROR
11 Job TCP341 Process 00020032 Thread 00000004 gsk_secure_socket_read
12 SSL V3 data read failed with 192.168.50.80.1472.
13 C01     MESSAGE   00000002  20:43:46.695599  SSL_EXIT
14 Job TCP341 Process 00020032 Thread 00000004 gsk_secure_socket_read
15 Exit status 000001A4 (420)
16 Length 0

```

Obtaining diagnostic information

1. The start of a trace record reporting that a function is 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 is not 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:

```
1 C01          DUMP          00000020  20:43:45.724056  SSL_ASCII_DUMP

2 Job TCP341    Process 00020032  Thread 00000004  send_v3_server_messages
3 SERVER-HELLO message
4 00000000: 02000046 03014373 B1011649 3E508E04  *...F..Cs...I>P..*
  00000010: A620B42C 8422C287 8015BC54 7850C435  *. ,,".....TxP.5*
  00000020: 540B6864 A4702000 020032C0 A8325005  *T.hd.p ...2..2P.*
  00000030: E9000000 00000000 00000000 00000043  *.....C*
  00000040: 73B10100 00051A00 0500          *s.....*
```

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

If 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:

```
DUMP COMM=(title of dump)
```

Obtaining diagnostic information

- Reply with:
R x,JOBNAME=(GSKSRVR),SDATA=(RGN,LSQA,ALLNUC,PSA,TRT,CSA,SQA),END
- Issue
TRACE CT,OFF,COMP=GSKSRVR

to turn off the trace.

Note: You need to take the dump before you turn off the CTRACE.

Obtaining diagnostic information

Chapter 13. Messages and codes

This topic lists the messages and codes issued by System SSL:

- SSL function return codes (“SSL function return codes”)
- Deprecated SSL function return codes (“Deprecated SSL function return codes” on page 618)
- ASN.1 status codes (014CExxx) (“ASN.1 status codes (014CExxx)” on page 629)
- CMS status codes (03353xxx) (“CMS status codes (03353xxx)” on page 633)
- SSL started task messages (GSK01nnn) (“SSL started task messages (GSK01nnn)” on page 656)
- Utility messages (GSK00nnn) (“Utility messages (GSK00nnn)” on page 665)

SSL function return codes

System SSL functions return the value 0 (GSK_OK) if no error is detected. Otherwise, one of the return codes listed in the `gskssl.h` include file is returned.

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 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, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. When using a PKCS #12 file, this error can also occur when the file is being processed during the establishment of the SSL/TLS environment when a certificate is encountered where there is no friendly name PKCS #12 attribute and the certificate's subject distinguished name is empty.

User response: Specify a label that exists in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. If encountered when establishing a SSL/TLS environment using a PKCS #12 file, verify any certificate that has no subject distinguished name is assigned a PKCS #12 friendly name attribute.

7 No certificates available.

Explanation: The key database, PKCS #12 file, 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 a PKCS #12 file, ensure that the file contains the necessary certificates. 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 that 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, PKCS #12 file, SAF key ring, 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, PKCS #12 file, 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. 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 that the latest changes are available. Collect a System SSL trace that contains the error and then contact your service representative if the problem persists.

For more information, see Chapter 4, "System SSL and FIPS 140-2," on page 19.

9 Cryptographic processing error.

Explanation: An error is detected by a cryptographic function. This error might 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 19.

10 ASN processing error.

Explanation: An error is detected while processing a certificate field. This error can also occur when a TLS client or server received a message containing a TLS extension that was not correctly formed. The TLS extension data may contain a length field that has an incorrect value.

User response: If using TLS extensions, ensure that the TLS extension data is correct for both the TLS server and client. If the error persists, 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 run time.

User response: Collect a System SSL trace containing the error and then contact your service representative.

13 Size specified for supplied structure is too small

Explanation: The value of the size field in the structure indicates that the size of the structure is insufficient.

User response: Ensure that the size field in the structure that is being used is initialized to the size of structure.

14 Required gsk_all_cipher_suites structure not supplied

Explanation: A gsk_all_cipher_suites structure required by the API was not supplied on the function call.

User response: Ensure that all parameters required by the API are specified on the function call

102 Error detected while reading certificate database

| **Explanation:** An error is detected while reading the key database, the PKCS #12 file, or retrieving entries from the SAF key ring or z/OS PKCS #11 token.

| **User response:** If the problem persists, 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 run time 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. Re-create 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 run time needs at least one CA or self-signed certificate 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 RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

201 No key database password supplied.

Explanation: A password stash file is specified but the SSL run time 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. Re-create 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, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. This error can occur if no name is supplied or the database, PKCS #12 file, key ring, or token does not exist. When using a PKCS #12 file, the file name cannot end with .kdb, .rdb or .sth.

| **User response:** Verify that the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token exists and is accessible by the application. This value is case-sensitive. Ensure that the case is preserved with your request. Collect a System SSL trace 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` utility to assign a new password for the key database.

302 Connection is active.

Explanation: An SSL secure connection operation cannot be completed because of an active request for the connection.

User response: Retry the failing request when the currently active request 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: This error can occur if:

- The client and server cipher specifications do not contain at least one value in common. Client and server cipher specifications might be limited depending on which System SSL FMIDs are installed. See Appendix C, “Cipher suite definitions,” on page 687 for more information. Server cipher specifications are dependent on the type of algorithms that are used by the server certificate (RSA, DSA, ECDSA and/or Diffie-Hellman), which might limit the options available during cipher negotiation.
- 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.
- A client supporting only SSL V2 has specified session ID cache reuse by specifying `GSK_ENABLE_CLIENT_SET_PEER_ID` set to ON or `GSK_REQ_CACHED_SESSION` set to ON. Session ID cache reuse is not supported with protocol SSL V2.
- A server supporting only SSL V2 has specified session ID cache reuse through the `GSK_SID_VALUE` attribute. Session ID cache reuse is not supported with protocol SSL V2.
- Attempting to use a certificate with its ECC private key in the ICSF PKDS and only fixed ECDH ciphers are specified.
- Using the TLS V1.1 or higher protocol and only the 40-bit export ciphers are specified.
- Using the TLS V1.2 or higher protocol and only 56-bit DES ciphers are specified.
- Using the TLS V1.2 or higher protocol and none of the server cipher specifications use key algorithms that are listed in the signature algorithms pairs sent by the client.
- An attempt was made to use a certificate with its DH secure private key in the ICSF PKDS. Only clear private keys are supported.
- An attempt was made to use a certificate with its ECC secure private key in the ICSF PKDS. Only clear private keys are supported.
- Using Suite B mode and no required Suite B ciphers were specified.

User response: Ensure that the client and the server have at least one cipher specification and protocol in common. Verify that specified session identifier is correct, not expired, that the cache is large enough to hold the cached session entry, and that maximum connection has not been reached.

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 run time.

User response: Collect a System SSL trace that contains a dump with 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 run time 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 not supported while executing in FIPS mode. If using a PKCS #12 file as the certificate database, the label is either the certificate's friendly name or the subject's distinguished name.

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. If using a PKCS #12 file, use the **gskkyman** command line option **-dc** or **-dcv** to display the contents of the PKCS #12 file. The friendly name or subject distinguished name values is displayed in the label field.

For more information about FIPS, see Chapter 4, "System SSL and FIPS 140-2," on page 19.

408 Key database password is not correct.

Explanation: The System SSL run time 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. Re-create 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 that 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 because of 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 SSL protocol you want 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 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 run time is unable to access a file or system facility.

User response: Ensure that 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, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token.

| **User response:** Add the self-signed certificate to the key database, PKCS #12 file, 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 that the latest changes are available.

420 Socket closed by remote partner.

Explanation: The remote partner closed the socket. This error is also reported if the remote partner 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. See Table 18 on page 687 for more information about the supported SSL V2 cipher suite definitions.

422 SSL V3 cipher is not valid.

Explanation: The SSL V3 cipher is not valid.

| **User response:** Specify a valid cipher. See Table 19 on page 687 for more information about the supported SSL V3 cipher suite definitions.

427 LDAP is not available.

Explanation: The System SSL run time 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 using a SAF key ring that is owned by another user, 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 user ID of the application or be SITE certificates. This error can occur when using z/OS PKCS #11 tokens if the user ID of the application does not have appropriate access to the CRYPTOZ class. This error can occur when using private keys associated with user certificates in a SAF key ring that is owned by another user if the user ID of the application does not have appropriate access to the *ringOwner.ringName.LST* resource in the RDATALIB class.

User response: Ensure that the ICSF started task is started before the application if the private key is stored in ICSF. When using z/OS PKCS #11 tokens, ensure that the user ID has appropriate access to the CRYPTOZ class.

If executing in FIPS mode, ensure that the certificate that is being used does not have its private key stored in ICSF.

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 is 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. This code occurs if renegotiation is disabled, or if the client or server rejects the renegotiation. If using the TLS protocol, and a no renegotiation alert is sent to the peer or received from the peer, then SSL processing continues using the current session parameters. If using the TLS or the SSL V3 protocol, and a handshake failure alert is sent to the peer or received from the peer, then the SSL connection is closed.

User response: If the session parameters are expected to be successfully reset, then the connection must be closed.

433 Key exceeds allowable export size.

Explanation: The key size that is 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 run time uses 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 RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

436 Certificate revocation list cannot be found.

Explanation: The required certificate revocation list (CRL) cannot be found in the specified LDAP server when the `gsk_crl_security_level` is set to MEDIUM or HIGH or the CRL cannot be found in the HTTP server indicated in the CRL distribution points extension and the `GSK_REVOCATION_SECURITY_LEVEL` is set to MEDIUM or HIGH.

User response: If contacting an LDAP server to retrieve the CRL, verify that the CRL is present in the LDAP entry being searched and is valid. Verify that the certificate's issuer is the same as the CRL issuer. Contact the certification authority and obtain the required CRL.

If contacting an HTTP server to retrieve the CRL, verify that the CRL is present on the HTTP server. Contact the HTTP server administrator to verify that the CRL is present on the server. If there are `crlIssuers` present in the CRL distribution point extension, verify that there is at least one match between those and the CRL issuer. If a match cannot be found in the `crlIssuers` in the CRL distribution point extension or there are no `crlIssuers` present, verify that the certificate's issuer is the same as the CRL issuer. The HTTP server administrator may need to contact the certification authority to obtain the required CRL.

Collect a System SSL trace containing the error and then contact your service representative if the problem persists.

437 Connection closed.

Explanation: For `gsk_secure_socket_read()`, a close notification is received from the peer application. For `gsk_secure_socket_write()`, a close notification is 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 is sent to the peer application.

User response: None

438 Internal error reported by remote partner.

Explanation: The peer application detected an internal error while performing an SSL operation and 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 sent an alert message that is not recognized by the System SSL run time.

User response: Collect a System SSL trace and then contact your service representative.

440 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.
- Diffie-Hellman 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 fixed Diffie-Hellman key exchange must allow key agreement.
- ECC client and server certificates using fixed EC Diffie-Hellman (ECDH) 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 (user) or server end-entity certificate, ensure that the appropriate option was selected from the Certificate Usage menu to create a client (user) or server certificate. The Certificate Usage menu consists of options for creating certificate authority and client (user) / server end-entity certificates.

442 Multiple certificates exist for label.

Explanation: Access of certificate/key from 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.

| If using a PKCS #12 file, use the **gskkyman** command line option **-dc** or **-dcv** to display the contents of the PKCS #12
| file. The friendly name or subject distinguished name value is displayed as the label. Ensure the specified label is
| unique in the PKCS #12 file.

443 Multiple keys are marked as the default.

Explanation: Access of key from 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.

444 Error encountered generating random bytes.

Explanation: The SSL/TLS handshake encountered an error while generating random bytes.

User response: Retry the secure connection. Contact your service representative if the error persists.

445 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.

446 TLS extension mismatch has been encountered.

Explanation: The TLS client 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.

User response: Ensure that the TLS server is operating correctly. If the problem persists, collect a System SSL trace and contact your service representative.

447 Required TLS extension has been rejected.

Explanation: The TLS server or client encountered a communicating partner that does not support a TLS extension that is defined as required.

User response: 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.

448 Requested server name is not recognized.

Explanation: The TLS server is unable to match the server names that are supplied in a "Server Name Indication" type TLS extension, and either the TLS server or TLS client determined this scenario to be fatal.

User response: Ensure that the TLS extension data is correct for both the TLS server and client.

449 Unsupported fragment length was received.

Explanation: The TLS server 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.

User response: 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.

450 TLS extension length field is not valid.

Explanation: The TLS client or server 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.

451 Elliptic Curve is not supported.

Explanation: The EC domain parameters that are defined for the elliptic curve public or private key are not supported.

User response: Ensure the elliptic curve public/private key pair uses a supported elliptic curve. See Chapter 3, "Using cryptographic features with System SSL," on page 11 for the list of elliptic curves that are supported by System SSL.

452 EC Parameters not supplied

Explanation: A gsk_buffer structure containing the EC domain parameters was not supplied on the call.

User response: Supply a gsk_buffer structure containing the EC domain parameters on the function call.

453 Signature not supplied

Explanation: A gsk_buffer structure containing the signature was not supplied on the call.

User response: Supply a gsk_buffer structure containing the signature on the function call.

454 Elliptic Curve parameters are not valid

Explanation: The EC domain parameters that are defined for the elliptic curve public or private key are not valid. Either no parameters could be found or the parameters could not be successfully decoded.

User response: Ensure the elliptic curve public/private key pair uses a valid elliptic curve.

455 ICSF services are not available

Explanation: A cryptographic process cannot be completed because of ICSF callable services being unavailable. This error might also occur when attempting to use a cipher suite that uses ICSF to perform a United States only encryption algorithm (such as AES-GCM) when ICSF is only able to use US export restricted encryption algorithms.

User response: Ensure that ICSF is running and operating correctly. If ICSF is running correctly, ensure that ICSF is able to use United States only encryption algorithms.

456 ICSF callable service returned an error

Explanation: An ICSF callable service that is employed to facilitate a cryptographic process returned an error condition. This error can occur if the user ID of the application does not have appropriate access to the RACF CSFSERV class resource profiles.

User response: Ensure that ICSF is operating correctly and that the user ID of the application has appropriate access to the RACF CSFSERV class resource profiles. See Table 4 on page 17 or Table 5 on page 17 for information about resource profiles. Collect a System SSL trace and verify the ICSF return code and reason code relating to the error. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about ICSF return and

reason codes. If the problem persists contact your service representative.

457 ICSF PKCS #11 not operating in FIPS mode

Explanation: While running in FIPS mode, an attempt was made to use ICSF PKCS #11 services, which were not operating in FIPS mode.

User response: Ensure that ICSF is configured to run in FIPS mode.

458 The SSL V3 expanded cipher is not valid

Explanation: The SSL V3 4-character cipher is not valid.

User response: Specify a valid 4-character cipher. See Table 20 on page 691 for more information about supported 4-character ciphers.

459 Elliptic Curve is not supported in FIPS mode.

Explanation: The EC domain parameters that are defined for the elliptic curve public or private key are not approved in FIPS mode.

User response: Ensure the elliptic curve for the public or private key is valid in FIPS mode. See Chapter 4, “System SSL and FIPS 140-2,” on page 19 for a list of elliptic curves that are supported by System SSL when running in FIPS mode.

460 Required TLS Renegotiation Indication not received

Explanation: TLS Renegotiation Indication was not received on the initial handshake with peer as required by the GSK_EXTENDED_RENEGOTIATION_INDICATOR environment variable or the gsk_attribute_set_enum enumeration ID GSK_EXTENDED_RENEGOTIATION_INDICATOR. If a server receives this code, then the GSK_EXTENDED_RENEGOTIATION_INDICATOR is set to either SERVER or BOTH and the client did not signal TLS Renegotiation Indication on the initial client hello. If a client receives this code, then the GSK_EXTENDED_RENEGOTIATION_INDICATOR is set to either CLIENT or BOTH and the server did not signal TLS Renegotiation Indication on the initial server hello.

User response: Ensure that the peer is configured to signal TLS Renegotiation Indication. If the peer does not support TLS Renegotiation Indication, and connection is required, then adjust the local setting of the environment variable GSK_EXTENDED_RENEGOTIATION_INDICATOR to “OPTIONAL” or the gsk_attribute_set_enum enumeration ID GSK_EXTENDED_RENEGOTIATION_INDICATOR to GSK_EXTENDED_RENEGOTIATION_INDICATOR_OPTIONAL.

461 EC domain parameter format is not supported.

Explanation: The server key exchange message contains an elliptic curve parameter format or named curve specification that is not supported

User response: For ephemeral ECDH cipher suites, ensure that only the named curve EC domain parameter format is used in the server key exchange message, with a named curve that is supported by System SSL.

462 Elliptic Curve point format is not supported.

Explanation: The elliptic curve public value is specified using an EC point format that is not supported.

User response: Ensure the elliptic curve public value is specified using a supported EC point format. System SSL supports only the uncompressed EC points format.

463 Cryptographic hardware does not support service or algorithm

Explanation: A call requiring cryptographic hardware was made to ICSF. The current installation hardware does not support the service or algorithm that is being used.

User response: Ensure that the correct protocol is in use for your installation, and that the cryptographic hardware required for this service or algorithm is available to ICSF.

464 Elliptic curve list is not valid.

Explanation: The supported elliptic curve list is not formatted correctly.

User response: Ensure the value that is supplied for `GSK_CLIENT_ECURVE_LIST` contains only entries for elliptic curves that are supported by System SSL. See Table 22 on page 695 for a list of supported elliptic curve definitions. Ensure that each entry uses 4 decimal digits.

When operating in Suite B mode, ensure that the value supplied contains the elliptical curves that are required by the Suite B profile in use. See "Suite B cryptography support" on page 45 for a list of required elliptical curves for each Suite B profile.

465 ICSF PKCS #11 services are disabled

Explanation: An attempt was made to use ICSF PKCS #11 services, which are disabled because of an ICSF FIPS self-test failure.

User response: Stop and restart ICSF. System SSL might need restarting to regain the full hardware benefit from ICSF. Contact your service representative if the error persists.

466 Signature algorithm pairs list is not valid.

Explanation: The supported signature algorithm pairs list is not correctly formatted.

User response: Ensure the value that is supplied for `GSK_TLS_SIG_ALG_PAIRS` contains only valid entries for hash and signature algorithm pairs that are supported by System SSL, and that each entry is defined using 4 digits. See Table 23 on page 695 for a list of valid 4-character signature algorithm pair definitions.

467 Signature algorithm not in signature algorithm pairs list.

Explanation: A signature algorithm that is used to sign a local or peer certificate is not included in the signature algorithm pairs list. The server certificate chain must use signature algorithms included in the signature algorithm pairs that are presented by the client during the TLS handshake. The client certificate chain must use signature algorithms included in the signature algorithm pairs that are presented by the server during the TLS handshake.

User response: Ensure that the signatures of the local and peer certificates in the certificate chain use signature algorithms that are present in the signature algorithm pairs list that is presented by the session partner. If the certificate chain is correct, then configure the client or server or both to specify all necessary signature algorithm pairs in the environment variable `GSK_TLS_SIG_ALG_PAIRS` to allow use of the certificate chain. See Table 23 on page 695 for a list of valid 4-character signature algorithm pair definitions.

468 Certificate key algorithm not in signature algorithm pairs list.

Explanation: The certificate key algorithm of the local certificate cannot be used to generate digital signatures as it is not included in the signature algorithm pairs list. The server certificate must use a key algorithm included in the signature algorithm pairs list that is presented by the client during the TLS handshake. The client certificate must use a key algorithm included in the signature algorithm pairs list that is presented by the server during the TLS handshake.

User response: Ensure that the key algorithm of the certificate is present in the signature algorithm pairs list that is presented by the session partner. If the certificate is correct, then configure the client or server or both to specify all necessary signature algorithm pairs in the environment variable `GSK_TLS_SIG_ALG_PAIRS` that allows the use of the certificate's key for generating digital signatures. See Table 23 on page 695 for a list of valid 4-character signature algorithm pair definitions.

469 Incorrect key attribute.

Explanation: One or more PKCS #11 attributes or parameters for a key are missing or incorrect for a requested function that is being performed. For example, a signing operation requires that for the key that is being used, the PKCS #11 sign attribute is to be TRUE. Verify that the correct key is being used for the requested function, and that all required attributes are set for that key. If you are using `gsk_make_enveloped_private_key_msg0`, ensure that a recipient certificate's RSA public key is valid.

User response: Verify that a certificate's PKCS #11 key attributes are correct for the function that is being performed.

470 Certificate does not meet Suite B requirements.

Explanation: The certificate in use does not meet the requirements for the Suite B profile that is selected for the environment.

User response: Ensure that the certificate used for the connection satisfies the requirements for the chosen Suite B profile. See “Suite B cryptography support” on page 45 for more information about Suite B certificate requirements.

471 Secure private key cannot be used with a fixed ECDH key exchange.

Explanation: A handshake attempted to perform an ECDH key exchange. The certificate's private key is a label that is pointing to a secure key. This is not a supported operation.

User response: Choose a certificate that does not have a secure private key or a cipher that does not perform an ECDH key exchange.

472 Clear key support not available due to ICSF key policy.

Explanation: Unable to generate clear keys or PKCS #11 objects because of the caller's RACF access to CRYPTOZ class resource CLEARKEY.SYSTOK-SESSION-ONLY does not allow the generation of non-secure (clear) PKCS #11 keys.

User response: Ensure that the user ID of the application has appropriate access to the RACF CRYPTOZ class resource CLEARKEY.SYSTOK-SESSION-ONLY.

473 OCSP responder requires a signed request

Explanation: The OCSP responder contacted for certificate validation requires that all OCSP requests be signed.

User response: Enable OCSP request signing by specifying the label of the signing certificate (GSK_OCSP_REQUEST_SIGKEYLABEL) and the signature algorithm (GSK_OCSP_REQUEST_SIGALG).

474 HTTP response is not valid

Explanation: The HTTP response received was not properly formatted or contents are not valid.

User response: Ensure that the HTTP 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.

475 OCSP response is not valid

Explanation: The OCSP ASN.1 encoded response received was not properly formatted or contents are not valid.

User response: Ensure that the OCSP responder 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.

476 Session ID entry does not exist

Explanation: The session identifier specified for either GSK_PEER_ID by a client application or GSK_SID_VALUE by a server application does not exist or has expired. If a client application has set GSK_REQ_CACHE_SESSION to ON, the required cached session entry could not be located. If a server application has set GSK_SID_VALUE, the required cached session entry could not be located. For a client application, the maximum number of SSL environment connections has been reached by a client application so new GSK_PEER_IDS cannot be stored in the session cache.

User response: Verify that the specified session identifier is correct, is not expired, the cache is large enough to hold the cached session entry, and the maximum client connections has not been reached.

477 Client session identifier does not match the server session identifier

Explanation: The server failed the connection request because the session identifier provided by the client through the client hello does not match the server GSK_SID_VALUE that was specified by `gsk_attribute_set_buffer()`.

User response: Verify that specified session identifiers are correct.

478 Client session cache attributes do not agree

Explanation: Client application attributes GSK_ENABLE_CLIENT_SET_PEERID, GSK_REQ_CACHED_SESSION, GSK_V3_SIDCACHE_SIZE, and GSK_V3_TIMEOUT conflict.

User response: Verify that the settings for GSK_ENABLE_CLIENT_SET_PEERID, GSK_REQ_CACHED_SESSION, GSK_V3_SIDCACHE_SIZE, and GSK_V3_TIMEOUT are in agreement.

Attribute conflicts may cause the application to not behave as desired and may result in handshake failures. SID cache reuse requires that the cache be defined and active.

One or more of the following conflicts may need to be corrected:

- A client application has set GSK_ENABLE_CLIENT_SET_PEERID to OFF and GSK_REQ_CACHED_SESSION to ON.
- A client application has set GSK_ENABLE_CLIENT_SET_PEERID to ON and attributes GSK_V3_SIDCACHE_SIZE or GSK_V3_TIMEOUT to zero.
- A server has set GSK_SID_VALUE and attributes GSK_V3_SIDCACHE_SIZE or GSK_V3_TIMEOUT to zero.

479 SID VALUE is not valid

Explanation: An invalid user-supplied GSK_SID_VALUE was encountered during `gsk_secure_socket_init()`.

User response: An application should only set GSK_SID_VALUE for a new connection if cache reuse is desired. The System SSL generated value can only be retrieved from SSL using `gsk_attribute_get_buffer()`. The buffer that is returned belongs to an active System SSL connection and should not be modified or the storage freed by the application. If the connection that provided the GSK_SID_VALUE buffer has closed, the data pointed to by the buffer cannot be determined. An application must copy this data into the application's own storage in order to continue reusing a particular cached session prior to closing the originating connection.

480 PEER ID is not valid

Explanation: An invalid user-supplied GSK_PEER_ID was encountered during `gsk_secure_socket_init()`.

User response: An application should only set GSK_PEER_ID for a new connection if cache reuse is desired. The System SSL generated value can only be retrieved from SSL using `gsk_attribute_get_buffer()`. The buffer that is returned belongs to an active System SSL connection and should not be modified or the storage freed by the application. If the connection that provided the GSK_PEER_ID buffer has closed, the data pointed to by the buffer cannot be determined. An application must copy this data into the application's own storage in order to continue reusing a particular cached session.

481 OCSP request failed with internal responder error

Explanation: The OCSP responder contacted for certificate validation returned an internal error.

User response: Ensure that the OCSP responder 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.

482 OCSP response is expired

Explanation: The current time is after the OCSP response expiration time.

User response: If using the dedicated OCSP responder, ensure that the OCSP responder server is using the most recent revocation information available from the certification authority. If certificate revocation through the AIA extension is enabled, ensure that the OCSP responder servers referenced in the certificate chain are using the most recent revocation information available. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

483 Error creating OCSP request

Explanation: An internal error was encountered while creating the OCSP request to send to an OCSP responder.

User response: If OCSP request signing is enabled, verify that the signing certificate resides in the SAF key ring, SSL key database, PKCS#11 token, or PKCS#12 file and the signing certificate is valid (start time is before the current

- | time and is not yet expired) and contains a private key. The key repository is provided through the
 | GSK_KEYRING_FILE environment variable or attribute. The OCSP signing certificate is provided through the
 | GSK_OCSP_REQUEST_SIGKEYLABEL environment variable or attribute.
- | Collect a System SSL trace containing the error and then contact your service representative if the error persists.

| **484 Maximum response size exceeded**

- | **Explanation:** When attempting to retrieve revocation information, the HTTP response exceeded the maximum
 | configured response size for either an OCSP response or a HTTP CRL. The response size is provided through either
 | GSK_HTTP_CDP_MAX_RESPONSE_SIZE or GSK_OCSP_MAX_RESPONSE_SIZE environment variables or attributes.
- | **User response:** Ensure that the HTTP response maximum size is adequate for the size of the CRLs or OCSP
 | responses that are being retrieved. If necessary, increase the maximum response size until an adequate size is
 | provided to handle the CRLs or OCSP responses that are being retrieved. If unable to determine an adequate size,
 | collect a System SSL trace containing the error and then contact your service representative.

| **485 HTTP server communication error**

- | **Explanation:** Unable to establish a connection to contact the HTTP server or the OCSP responder to retrieve
 | certificate revocation information.
- | **User response:** If enabled for OCSP and a dedicated OCSP responder is enabled, ensure that the responder is
 | running and can be accessed.
- | If enabled for OCSP responders identified in the certificate AIA extension, ensure that the OCSP responders specified
 | in the extension are running and can be accessed.
- | If HTTP CRL support is enabled, ensure that the HTTP server specified in the CRL Distribution Point extension is
 | running and can be accessed.
- | If there is a firewall in place and either an HTTP proxy server or port and/or an OCSP proxy server or port has been
 | identified, ensure that the servers and ports settings are correct and the servers can be accessed. The HTTP proxy
 | server and port are specified through the GSK_HTTP_CDP_PROXY_SERVER_NAME and
 | GSK_HTTP_CDP_PROXY_SERVER_PORT attributes or environment variables. The OCSP proxy server and port are
 | specified through the GSK_OCSP_PROXY_SERVER_NAME and GSK_OCSP_PROXY_SERVER_PORT attributes or
 | environment variables.
- | Collect a System SSL trace containing the error and then contact your service representative if the error persists.

| **486 Nonce in OCSP response does not match value in OCSP request**

- | **Explanation:** When validating the nonce in the OCSP response, the value did not match the value sent in the OCSP
 | request.
- | **User response:** If OCSP is enabled for the dedicated OCSP responder (GSK_OCSP_URL), ensure that the OCSP
 | responder server is configured to send a nonce in OCSP responses.
- | Ensure that nonce checking is required. If not required, set GSK_OCSP_NONCE_CHECK_ENABLE to off.
- | Collect a System SSL trace containing the error and then contact your service representative if the error persists.

| **487 OCSP response not received within configured time limit**

- | **Explanation:** The time limit indicated in the value for GSK_OCSP_RESPONSE_TIMEOUT has been exceeded.
- | **User response:** Ensure that the HTTP server where the OCSP responder resides is available and able to process
 | OCSP requests. Verify that the value for GSK_OCSP_RESPONSE_TIMEOUT is sufficient to receive a complete
 | response from the HTTP server containing the OCSP responder.

| **488 Revocation information is not yet valid**

- | **Explanation:** The current time is earlier than the validity period of the revocation information provided though
 | either an OCSP response or CRL.
- | **User response:** Ensure that the system time is configured correctly. Collect a System SSL trace containing the error

and then contact your service representative if the problem persists.

489 HTTP server host name is not valid

Explanation: The URI value in the AIA extension or the CDP extension is not in the correct format or cannot be resolved by the Domain Name Service (DNS). The correct URI format is `http://hostname[:portNumber]`.

User response: If the `GSK_OCSP_ENABLE` parameter is enabled, verify that the certificate being verified has a URI value in the AIA extension that is properly formatted and can be resolved by the DNS. It may be necessary to obtain a new certificate or to specify the `GSK_OCSP_PROXY_SERVER_NAME` and `GSK_OCSP_PROXY_SERVER_PORT` parameters if there is a need to pass through a firewall. If the `GSK_OCSP_URL` or `GSK_OCSP_PROXY_SERVER_NAME` parameters are specified, verify that the host name and the IP address is properly formatted and can be resolved by the DNS.

If the `GSK_HTTP_CDP_ENABLE` parameter is enabled, verify that the certificate being verified has a URI value in the CDP extension that is properly formatted and can be resolved by the DNS. It may be necessary to obtain a new certificate or to specify the `GSK_HTTP_CDP_PROXY_SERVER_NAME` and `GSK_HTTP_CDP_PROXY_SERVER_PORT` parameters if there is a need to pass through a firewall.

490 PKCS #12 file content not valid

Explanation: When processing the PKCS #12 file, a format error was detected. This can occur if the file is not properly ASN.1 encoded, been modified if transferred, or the PKCS #12 file is not a Version 3 binary file. PKCS #12 Version 1 files and files in Base64 format are not supported.

User response: If the current file is either a PKCS #12 Version 1 file or a Base64 encoded file, it must be replaced with a PKCS #12 Version 3 file. If transferring the file, be sure to transfer the file in binary format. Correct the PKCS #12 file or obtain a new PKCS #12 file. If the problem persists, collect a System SSL Trace containing the error and then contact your service representative.

491 Required basic constraints certificate extension is missing

Explanation: During the establishment of an SSL/TLS secure connection (`gsk_secure_socket_init()` or `gsk_secure_soc_init()` API) with certificate validation processing set to mode ANY or 2459, an intermediate CA certificate was encountered outside of a trusted certificate source (for example, key database file, PKCS #12 file, SAF key ring, or PKCS #11 token), which does not have a basic constraints extension.

User response: Contact the connection partner to determine the certificates being utilized. Once the certificates are identified, either new valid Version 3 certificates can be obtained to replace the intermediate CA certificate or certificates causing the error, or if the usage of the intermediate CA certificates is deemed to be acceptable, a version of the CA certificate or certificates needs to be added to the application's trusted certificate source (for example, key database file, PKCS #12 file, SAF key ring, or PKCS #11 token).

If the error persists after adding the certificates, or if the certificates can not be readily obtained, collect a System SSL trace containing the error and then contact your service representative.

492 Maximum number of locations allowed to be contacted during certificate validation has been reached

Explanation: The number of locations allowed by either `GSK_MAX_SOURCE_REV_EXT_LOC_VALUES` or `GSK_MAX_VALIDATION_REV_EXT_LOC_VALUES` has been exceeded. The locations for revocation information are specified by the `accessLocation` in the AIA certificate extension for OCSP and the `distributionPoint` in the CDP extension for HTTP CRLs.

User response: Use the values in the certificate chain being validated to determine the proper value for `GSK_MAX_SOURCE_REV_EXT_LOC_VALUES`, `GSK_MAX_VALIDATION_REV_EXT_LOC_VALUES`, or both. The value for `GSK_MAX_SOURCE_REV_EXT_LOC_VALUES` must be greater than or equal to the maximum number of location values in a CDP or AIA extension used in the certificate chain being validated. The value for `GSK_MAX_VALIDATION_REV_EXT_LOC_VALUES` must be greater than or equal to the total number of location values in all CDP and AIA extensions used in the certificate chain being validated. Collect a System SSL trace containing the error and then contact your service representative if the problem persists.

493 HTTP response not received within configured time limit

Explanation: The time limit indicated in the value for `GSK_HTTP_CDP_RESPONSE_TIMEOUT` has been exceeded.

User response: Ensure that the HTTP server is available and able to process HTTP CRL requests. Verify that the value for `GSK_HTTP_CDP_RESPONSE_TIMEOUT` is sufficient to receive a complete response from the HTTP server

494 LDAP response not received within configured time limit

Explanation: The time limit indicated in the value for `GSK_LDAP_RESPONSE_TIMEOUT` has been exceeded.

User response: Ensure that the LDAP server is available and able to process LDAP CRL requests. Verify that the value for `GSK_LDAP_RESPONSE_TIMEOUT` is sufficient to receive a complete response from the LDAP server.

495 OCSP request failed with try later error

Explanation: The OCSP responder is unable to currently process the OCSP request.

User response: Contact the OCSP responder administrator to verify that the OCSP responder is working properly. Then retry the OCSP request at a later time.

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 that is 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 that is 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, TLS V1.1, or TLS V1.2.

Explanation: The requested function requires the SSL V3, TLS V1.0, TLS V1.1, or TLS V1.2 protocol.

User response: Ensure that the correct protocol is in use before issuing the request.

602 Function identifier is not valid.

Explanation: The function identifier that is specified for `gsk_secure_socket_misc()` is not valid.

User response: Specify a valid function identifier.

603 Specified function enumerator is not valid.

Explanation: The value that is specified is not a value that is enumerated as a function for the API.

User response: Ensure that the correct function enumerator is coded for the function.

604 Send sequence number is near maximum value

Explanation: While using TLS V1.1 or higher protocol, the send sequence number is near the maximum value before which it wraps. For TLS V1.1 and higher, an SSL handshake must occur to reset the send sequence number before the sequence number wrapping. System SSL is unable to automatically initiate a handshake on the current function call. This code is not returned again until after a handshake for the connection resets the send sequence number and the send sequence number is again near the maximum value.

User response: The caller should initiate a handshake by calling `gsk_secure_socket_misc`, specifying `GSK_RESET_CIPHER`. When the handshake is initiated, the previous function call that returned this code can be called again.

701 Attribute identifier is not valid.

Explanation: The attribute identifier is not valid.

User response: Specify a valid attribute identifier.

702 Attribute length is not valid.

Explanation: The attribute length is not valid.

User response: Specify a valid attribute length.

703 Enumeration is not valid.

Explanation: The enumeration value is not valid.

User response: Specify a valid enumeration value.

704 Session identifier cache callback is not valid.

Explanation: The session identifier cache callback values are not valid. All callback routines must be provided to use an application session identifier cache.

User response: Specify valid session identifier cache callback values.

705 Numeric value is not valid.

Explanation: The numeric value is not valid.

User response: Specify a valid numeric value.

706 Attribute parameter is not valid.

Explanation: The attribute parameter value is not valid.

User response: Specify a valid attribute parameter value.

707 **TLS extension type is not valid.**

Explanation: The TLS extension type is not valid or not supported.

User response: Specify a valid or supported TLS extension type value.

708 **Supplied TLS extension data is not valid.**

Explanation: TLS extension data that is submitted to the SSL environment or connection is incorrectly defined.

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

The deprecated System SSL functions return the value 0 (GSK_OK) if no error is detected. Otherwise, one of the return codes listed in the gskssl.h include file is returned.

1 Error detected while reading certificate database

Explanation: An error is detected while reading the key database, the PKCS #12 file, or retrieving entries from the SAF key ring or z/OS PKCS #11 token.

User response: Collect a System SSL trace containing the error and then contact your service representative.

2 Error detected while opening the certificate database.

Explanation: An error is detected while opening the key database, PKCS #12 file, 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. When using a PKCS #12 file, the file name cannot end with .kdb, .rdb or .sth.

User response: Verify that the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token exists and is accessible by the application. This value is case-sensitive. Ensure that the case is preserved with your request. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

3 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.

4 Key database password is not correct.

Explanation: The System SSL run time 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 and both the file and directory path are accessible to the application

9 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. If using a PKCS #12 file as the certificate database, the label is either the certificate's friendly name or the subject's distinguished name. A default key does not exist in a PKCS #12 file.

User response: Supply a valid label or define a default key in the key database. If using a PKCS #12 file, use the **gskkyman** command line option **-dc** or **-dcv** to display the contents of the PKCS #12 file. The friendly name or subject distinguished name values is displayed in the label field.

12 Key label is not found.

Explanation: The requested key label is not found in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. When using a PKCS #12 file, this error can also occur when the file is being processed during the establishment of the SSL/TLS environment when a certificate is encountered where there is no friendly name PKCS #12 attribute and the certificate's subject distinguished name is empty.

User response: Specify a label that exists in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. If encountered when establishing a SSL/TLS environment using a PKCS #12 file, verify any certificate that has no subject distinguished name is assigned a PKCS #12 friendly name attribute.

13 Duplicate subject names.

Explanation: 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.

User response: Either remove the duplicate certificates or specify a label instead of a DN in the `gsk_secure_soc_init()` initialization data.

16 Incorrect key database password.

Explanation: The System SSL run time 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. Re-create the database if the error persists.

17 Key database password is expired.

Explanation: The key database password is expired.

User response: Use the `gskkyman` utility to assign a new password for the key database.

18 No certification authority certificates.

| **Explanation:** The key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token does not contain any valid certification authority certificates. The SSL run time needs at least one CA or self-signed certificate to perform client authentication.

| **User response:** Add the necessary certificates to the key database, PKCS #12 file, 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 that the latest changes are available.

19 No certificates available.

| **Explanation:** The key database, PKCS #12 file, 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, PKCS #12 file, 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 RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that 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 that is 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 that is 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 that is 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 Appendix C, "Cipher suite definitions," on page 687 for more information. Server cipher specifications are dependent on the type of algorithms that are 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 that are 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 run time.

User response: Collect a System SSL trace containing a dump that contains 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. This code occurs if renegotiation is disabled or if the client or server rejects the renegotiation. If using the TLS protocol, and a no renegotiation alert is sent to the peer or received from the peer, then SSL processing continues using the current session parameters. If using the TLS or the SSL V3 protocol, and a handshake failure alert is sent to the peer or received from the peer, then the SSL connection is closed.

User response: If the session parameters are expected to be successfully reset, then the connection must be closed.

-9 Certificate is revoked.

Explanation: The certificate is 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 run time 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 that 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 because of 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 SSL protocol you want 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, PKCS #12 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 RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

-16 SSL protocol violation.

Explanation: The communication partner 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 run time is unable to access a file or system facility.

User response: Ensure that 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, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token.

User response: Add the self-signed certificate to the key database, PKCS #12 file, 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 that 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 that 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 run time uses 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 using a SAF key ring that is owned by another user, 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 user ID of the application or be SITE certificates. This error can occur when using z/OS PKCS #11 tokens if the user ID of the application does not have appropriate access to the CRYPTOZ class. This error can occur when using private keys associated with user certificates in a SAF key ring owned by another user if the user ID of the application does not have appropriate access to the *ringOwner.ringName.LST* resource in the RDATALIB 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 that the user ID has 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.

-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, PKCS #12 file, SAF key ring, 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, PKCS #12 file, 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 that the latest changes are available.

For more information, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

-36 Cryptographic processing error.

Explanation: An error is detected by a cryptographic function. This error might 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 19.

-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 run time 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 that is 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 because of 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 detected an internal error while performing an SSL operation and 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 sent an alert message that is not recognized by the System SSL run time.

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.
- Diffie-Hellman 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 Usage menu to create a user or server certificate.

-56 Multiple certificates exist for label.

Explanation: Access of certificate/key from 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.

| If using a PKCS #12 file, use the **gskkyman** command line option **-dc** or **-dcv** to display the contents of the PKCS #12
| file. The friendly name or subject distinguished name value is displayed as the label. Ensure that the specified label is
| unique in the PKCS #12 file.

-57 Multiple keys are marked as the default.

Explanation: Access of key from 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 suspends 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 run time.

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 encountered an error while generating random bytes.

User response: Retry 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.

-106 Required TLS Renegotiation Indication not received

Explanation: TLS Renegotiation Indication was not received on the initial handshake with the peer as required by the GSK_EXTENDED_RENEGOTIATION_INDICATOR environment variable. If a server receives this code, then the GSK_EXTENDED_RENEGOTIATION_INDICATOR is set to either SERVER or BOTH and the client did not signal TLS Renegotiation Indication on the initial client hello. If a client receives this code, then the GSK_EXTENDED_RENEGOTIATION_INDICATOR is set to either CLIENT or BOTH and the server did not signal TLS Renegotiation Indication on the initial server hello.

User response: Ensure that the peer is configured to signal TLS Renegotiation Indication. If the peer does not support TLS Renegotiation Indication, and connection is required, then adjust the local setting of the environment variable GSK_EXTENDED_RENEGOTIATION_INDICATOR to "OPTIONAL".

-107 ICSF services are unavailable

Explanation: A cryptographic process cannot complete because ICSF callable services are unavailable.

User response: Ensure that ICSF is running and operating correctly.

-108 ICSF callable service returned an error

Explanation: An ICSF callable service that is employed to facilitate a cryptographic process returned an error condition. This error can occur if the user ID of the application does not have appropriate access to the RACF CSFSERV class resource profiles.

User response: Ensure that ICSF is operating correctly and that the user ID of the application has appropriate access to the RACF CSFSERV class resource profiles. See Table 4 on page 17 or Table 5 on page 17 for information about resource profiles. Collect a System SSL trace and verify the ICSF return code and reason code relating to the error. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about ICSF return and reason codes. If the problem persists, contact your service representative.

-109 ICSF PKCS #11 not operating in FIPS mode

Explanation: While running in FIPS mode, an attempt was made to use ICSF PKCS #11 services that were not operating in FIPS mode.

User response: Ensure that ICSF is configured to run in FIPS mode.

-110 ICSF PKCS #11 services are disabled

Explanation: An attempt was made to use ICSF PKCS #11 services, which are disabled because of an ICSF FIPS self test failure.

User response: Stop and restart ICSF. System SSL might need restarting to regain the full hardware benefit from ICSF. Contact your service representative if the error persists.

-111 ICSF clear key support not available

Explanation: Unable to generate clear keys or PKCS #11 objects because of the caller's RACF access to CRYPTOZ class resource CLEARKEY.SYSTOK-SESSION-ONLY that does not allow the generation of non-secure (clear) PKCS #11 keys.

User response: Ensure that the user ID of the application has appropriate access to the RACF CRYPTOZ class resource CLEARKEY.SYSTOK-SESSION-ONLY.

-112 LDAP Response not received within configured time limit

Explanation: The time limit indicated in the value for GSK_LDAP_RESPONSE_TIMEOUT has been exceeded.

User response: Ensure that the LDAP server is available and able to process LDAP CRL requests. Verify that the value for GSK_LDAP_RESPONSE_TIMEOUT is sufficient to receive a complete response from the LDAP server.

-124 PKCS #12 file content not valid

Explanation: When processing the PKCS #12 file, a format error was detected. This can occur if the file is not properly ASN.1 encoded, been modified if transferred, or the PKCS #12 file is not a Version 3 binary file. PKCS #12 Version 1 files and files in Base64 format are not supported.

User response: If the current file is either a PKCS #12 Version 1 file or a Base64 encoded file, it must be replaced with a PKCS #12 Version 3 file. If transferring the file, be sure to transfer the file in binary format. Correct the PKCS #12 file or obtain a new PKCS #12 file. If the problem persists, collect a System SSL Trace containing the error and then contact your service representative.

-125 Required basic constraints certificate extension is missing

Explanation: During the establishment of an SSL/TLS secure connection (`gsk_secure_socket_init()` or `gsk_secure_soc_init()` API) with certificate validation processing set to mode ANY or 2459, an intermediate CA certificate was encountered outside of a trusted certificate source (for example, key database file, SAF key ring, or PKCS #11 token), which does not have a basic constraints extension.

User response: Contact the connection partner to determine the certificates being utilized. Once the certificates are identified, either new valid Version 3 certificates can be obtained to replace the intermediate CA certificate or certificates causing the error, or if the usage of the intermediate CA certificates is deemed to be acceptable, a version of the CA certificate or certificates needs to be added to the application's trusted certificate source (for example, key database file, SAF key ring or PKCS #11 token).

If the error persists after adding the certificates, or if the certificates can not be readily obtained, collect a System SSL trace containing the error and then contact your service representative.

ASN.1 status codes (014CExxx)

ASN.1 status codes have the prefix "014CE". These status codes identify ASN.1 encoding and decoding errors.

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 supported 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: Contact your service representative if this error occurs while decoding a bit string. Correct the application if this error occurs while encoding 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 is already 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 code set 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 run time. 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. See RFC 2253: *UTF-8 String Representation of Distinguished Names* for more information about 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)

Certificate Management Services (CMS) status codes have the prefix "03353". These status codes include informational messages, including errors that require a user response.

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 run time 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 run time 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 run time. 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 that is 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 run time 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 that 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 key ring can be accessed and is not damaged. If creating or updating a file, verify that the file system containing the file is not full. 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 key ring cannot be opened because it is not found.

User response: Verify that the correct name is specified. This value is case-sensitive. Ensure that the case is preserved with your request. 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 run time.

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 run time 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 that is passed to the System SSL run time is not valid. This error can occur if the handle is 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 run time 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: If using the `gskkyman` utility, use option 4 from the Database Menu to enlarge the database record length. Applications using the System SSL APIs can use the `gsk_change_database_record_length` API to enlarge the database 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. Also, this error can occur if the key that is being exported is a secure private key in the TKDS and the specified password length is greater than 63 bytes.

User response: Specify the correct password.

03353017 Access denied.

Explanation: The database or key ring 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 key ring.

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: The key entry does not contain a private key or the private key is not usable. This error might 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.
- This error can occur when using a SAF key ring if:
 - The key ring is owned by another user.
 - Using a private key that is associated with a user certificate in a SAF key ring that is owned by another user, and if the user ID of the application does not have appropriate access to the ringOwner.ringName.LST resource in the RDATA LIB class.
 - Certificates meant to represent a server or client must be connected to a SAF key ring with a USAGE value of PERSONAL, and either owned by the user ID of the application or SITE certificates.
- This error can occur when using z/OS PKCS #11 tokens if:
 - The user ID of the application does not have appropriate access to the CRYPTOZ class.
 - The label name is not valid for a certificate's PKCS #11 TKDS secure key.
 - The PKCS #11 key object does not exist.
 - The certificate's PKCS #11 TKDS secure key algorithm is not supported.
 - Using `gsk_make_enveloped_private_key_msg()` and the PKCS #11 secure key object that is used as input exists in the PKDS instead of the TKDS.

User response: Verify that the ICSF started task is running if the private key is stored in ICSF. Otherwise, repeat the failing request by using a database entry containing a private key. If using z/OS PKCS #11 tokens, ensure that the user ID has appropriate access to the CRYPTOZ class.

If executing in FIPS mode, ensure that the certificate that is being used does not have its private key stored in ICSF.

If using PKCS # 11 tokens:

- Verify that the certificate's PKCS #11 secure key label name is valid within the TKDS.
- Verify that the PKCS #11 TKDS secure key algorithm is supported.
- If you are using `gsk_make_enveloped_private_key_msg()`, verify that the input PKCS #11 key object exists in the TKDS.

0335301B Record label is not valid.

Explanation: The record label is not valid. A label may contain letters, numbers, and punctuation. 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 key database file and its associated request database.

User response: Verify the labels already in use by a key database file and its associated request database and 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 (for example, -----BEGIN CERTIFICATE-----), encoded text, and a footer line (for example, -----END CERTIFICATE-----). 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.

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.

If importing a certificate from a Base64 file, the first and last lines contain readable data. The first line in the file contains '-----BEGIN CERTIFICATE-----' and the last line in the file contains '-----END CERTIFICATE-----'. If data is not correct, ensure that the file was transferred successfully.

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. See "Database menu" on page 524 for information about displaying the contents of an external certificate file 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 that is specified in the certification authority certificate.

User response: Report the problem to the certification authority.

03353026 Incorrect key usage.

Explanation: The key usage certificate extension does not permit the requested key operation.

User response: Obtain a certificate, which allows the requested key operation.

03353027 Issuer is not a certification authority.

Explanation: The issuer of an X.509 certificate is not a certification authority. This indicates that the basic constraints certificate extension in the issuer certificate does not contain the certification authority indicator.

User response: Report the problem to the issuer of the certificate.

03353028 Export file format is not supported.

Explanation: The requested export file format is not supported for the specified database record. Certificates can be exported using the DER and PKCS #7 formats. Certificates and keys can be exported using the PKCS #12 formats.

User response: Select an appropriate export file format.

03353029 Cryptographic algorithm is not available.

Explanation: An X.509 cryptographic algorithm is not available. Because of government export regulations, strong encryption is not available on the local system.

User response: Select an algorithm that is available.

0335302A Record type cannot be changed.

Explanation: The record type cannot be changed when replacing a database record.

User response: Create a new database entry for the record.

0335302B Subject name cannot be changed.

Explanation: The subject name cannot be changed when replacing a database record where the database record has no private key or is used as a signing certificate for other user or server certificates.

User response: Create a new database entry for the record.

0335302C Public key cannot be changed.

Explanation: The subject public key cannot be changed when replacing a database record.

User response: Create a new database entry for the record.

0335302D Default key cannot be changed

Explanation: The default key for the database cannot be changed using the `gsk_replace_record()` routine.

User response: Use the `gsk_set_default_key()` routine to change the default key for the database.

0335302E Database contains certificates signed by the certificate.

Explanation: A CA certificate cannot be deleted because the database still contains certificates that were signed using that certificate. A certificate renewal for a signing certificate fails with this error code if the certificate's subject name has changed.

User response: Delete all certificates that are signed by the CA certificate before deleting the certificate. To renew a signing certificate with a changed subject name all dependent certificates must be resigned with the new certificate:

- Create certificate renewal requests for each dependent certificate and delete the dependent certificates and keys.
- Receive the new signing certificate.
- Sign any dependent certificate requests with the new signing certificate.
- Receive the signed dependent certificate renewals.

0335302F Certificate chain is not trusted.

Explanation: A certification authority (CA) certificate in the certification chain is not trusted.

User response: Set the trust status for the CA certificate if the certificate can be used for authentication purposes.

03353030 Key not supported by encryption or signature algorithm.

Explanation: The supplied key is not supported by the requested encryption or signature algorithm. For example, an RSA key cannot be used to verify that a DSA signature and a DSA key cannot be used to encrypt data.

User response: Provide the appropriate key for the encryption or signature algorithm.

03353031 Signer certificate not found.

Explanation: A 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: See 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 run time.

User response: See 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 2048 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 between 1024 and 2048 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. See 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 that is 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 run time 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 run time is unable to open the 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 that 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 Version is not supported.

Explanation: An unsupported version is encountered.

User response: See the Programming Reference for the failing routine to determine the supported 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. See RFC 2631: *Diffie-Hellman Key Agreement Method* for more information about 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. See RFC 2631: *Diffie-Hellman Key Agreement Method* for more information about 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. See FIPS 186-2: *DIGITAL SIGNATURE STANDARD (DSS)* for more information about 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 host name you want.

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 that are 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 that are 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 that are supplied from 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 that are 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` on the function call.

03353058 Modulus not supplied.

Explanation: A gsk_buffer structure containing the modulus for the RSA key was either not supplied on the call or not supplied in the gsk_private_key or gsk_public_key structure.

User response: Ensure that a gsk_buffer structure containing the modulus for the RSA key is supplied on the function call, or is defined in the gsk_private_key or gsk_public_key structure.

03353059 Public exponent not supplied.

Explanation: A gsk_buffer structure containing the public exponent for the RSA key was not supplied on the call or not supplied in the gsk_private_key or gsk_public_key structure.

User response: Ensure that a gsk_buffer structure containing the public exponent for the RSA key is supplied on the function call, or is defined in the gsk_private_key or gsk_public_key structure.

0335305A Private exponent not supplied.

Explanation: A gsk_buffer structure containing the private exponent for the RSA key was not supplied on the call, or not supplied in the gsk_private_key structure.

User response: Ensure that a gsk_buffer structure containing the private exponent for the RSA key is supplied on the function call, or is defined in the gsk_private_key structure.

0335305B First prime not supplied.

Explanation: A gsk_buffer structure containing the first prime for the RSA key was not supplied on the call, or not supplied in the gsk_private_key structure.

User response: Ensure that a gsk_buffer structure containing the first prime exponent for the RSA key is supplied on the function call, or is defined in the gsk_private_key structure.

0335305C Second prime not supplied.

Explanation: A gsk_buffer structure containing the second prime for the RSA key was not supplied on the call, or not supplied in the gsk_private_key structure.

User response: Ensure that a gsk_buffer structure containing the second prime for the RSA key is supplied on the function call, or is defined in the gsk_private_key structure.

0335305D First prime exponent not supplied.

Explanation: A gsk_buffer structure containing the first prime exponent for the RSA key was not supplied on the call, or not supplied in the gsk_private_key structure.

User response: Ensure that a gsk_buffer structure containing the prime exponent for the RSA key is supplied on the function call, or is defined in the gsk_private_key structure.

0335305E Second prime exponent not supplied.

Explanation: A gsk_buffer structure containing the second prime exponent for the RSA key was not supplied on the call, or not supplied in the gsk_private_key structure.

User response: Ensure that a gsk_buffer structure containing the second prime exponent for the RSA key is supplied on the function call, or is defined in the gsk_private_key structure.

0335305F CRT coefficient not supplied.

Explanation: A gsk_buffer structure containing the CRT coefficient for the RSA key was not supplied on the call, or not supplied in the gsk_private_key structure.

User response: Ensure that a gsk_buffer structure containing the CRT coefficient for the RSA key is supplied on the function call, or is defined in the gsk_private_key structure.

03353060 Certificate revocation list cannot be found.

Explanation: The required certificate revocation list (CRL) cannot be found in the specified LDAP server when the `gsk_crl_security_level` is set to HIGH or the CRL cannot be found in the HTTP server indicated in the CRL distribution points extension and the `gskcms_revocation_security_level` is set to MEDIUM or HIGH.

User response: If contacting an LDAP server to retrieve the CRL, verify that the CRL is present in the LDAP entry being searched and is valid. Verify that the certificate's issuer is the same as the CRL issuer. Contact the certification authority and obtain the required CRL.

If contacting an HTTP server to retrieve the CRL, verify that the CRL is present on the HTTP server. Contact the HTTP server administrator to verify that the CRL is present on the server. If there are `crlIssuers` present in the CRL distribution point extension, verify that there is at least one match between those and the CRL issuer. If a match cannot be found in the `crlIssuers` in the CRL distribution point extension or there are no `crlIssuers` present, verify that the certificate's issuer is the same as the CRL issuer. The HTTP server administrator may need to contact the certification authority to obtain the required CRL.

Collect a System SSL trace containing the error and then contact your service representative if the problem persists.

03353061 Multiple certificates exist for label.

Explanation: Access of certificate/key from 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 from 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 produced identical consecutive blocks of output data. If in FIPS mode, any further attempts to use System SSL continues 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 failed to match the expected results. Any further attempts to use System SSL continues 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 used is supported in the mode in which the application is executing. If you are invoking a FIPSONly API, you must 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 that the certificate chain is valid and the user certificate is intended to be used for the required purpose.

0335306E Variable argument count is not valid.

Explanation: The specified variable argument count is not valid.

User response: Specify a valid variable argument count.

0335306F 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 that 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.

| If performing a PKCS #7 operation to encode a signer identifier, ensure that the certificate has a subject key identifier extension.

03353070 Certificate extension data is incorrect.

Explanation: A certificate extension has incorrect data or has a necessary field missing.

User response: Ensure that 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.

03353071 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 that 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.

03353072 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 that 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.

03353073 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 that 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.

03353074 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 continues 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.

03353076 Prime not supplied.

Explanation: A `gsk_buffer` structure containing the prime for the DSA or Diffie-Hellman key was not supplied in the `gsk_private_key` or `gsk_public_key` structure.

User response: Ensure that the prime value for the DSA or Diffie-Hellman key is defined in the `gsk_private_key` or `gsk_public_key` structure.

03353077 Subprime not supplied.

Explanation: A `gsk_buffer` structure containing the `sub_prime` for the DSA key was not supplied in the `gsk_private_key` or `gsk_public_key` structure.

User response: Ensure that the sub prime value for the DSA key is defined in the `gsk_private_key` or `gsk_public_key` structure.

03353078 Base not supplied.

Explanation: A `gsk_buffer` structure containing the base for the DSA or Diffie-Hellman key was not supplied in the `gsk_private_key` or `gsk_public_key` structure.

User response: Ensure that the base value for the DSA or Diffie-Hellman key is defined in the `gsk_private_key` or `gsk_public_key` structure.

03353079 Private value not supplied.

Explanation: A `gsk_buffer` structure containing the private value for the DSA or Diffie-Hellman key was not supplied in the `gsk_private_key` structure.

User response: Ensure that the private value for the DSA or Diffie-Hellman key is defined in the `gsk_private_key` structure.

0335307A Public value not supplied.

Explanation: A `gsk_buffer` structure containing the public value for the DSA or Diffie-Hellman key was not supplied in the `gsk_public_key` structure.

User response: Ensure that the public value for the DSA or Diffie-Hellman key is defined in the `gsk_public_key` structure.

0335307B Private key structure not supplied.

Explanation: The structure containing the private key components was not supplied on the call.

User response: Supply the structure containing the private key components on the function call.

0335307C Public key structure not supplied.

Explanation: The structure containing the public key components was not supplied on the call.

User response: Supply the structure containing the public key components on the function call.

0335307D Size specified for supplied structure is too small.

Explanation: The value of the size field in the structure indicates that the size of the structure is insufficient.

User response: Ensure that the size field in the structure being used is initialized to the size of structure.

0335307E Elliptic Curve is not supported.

Explanation: The elliptic curve domain parameters that are defined for the elliptic curve public or private key are not supported.

User response: Ensure the elliptic curve public/private key pair uses a supported elliptic curve. See Chapter 3, "Using cryptographic features with System SSL," on page 11 for the list of elliptic curves that are supported by System SSL.

0335307F EC Parameters not supplied.

Explanation: A `gsk_buffer` structure containing the EC domain parameters was not supplied on the call.

User response: Supply a `gsk_buffer` structure containing the EC domain parameters on the function call.

03353080 Signature not supplied.

Explanation: A `gsk_buffer` structure containing the signature was not supplied on the call.

User response: Supply a `gsk_buffer` structure containing the signature on the function call.

03353081 Elliptic Curve parameters are not valid.

Explanation: The EC domain parameters that are defined for the elliptic curve public or private key are not valid. Either no parameters could be found or the parameters could not be successfully decoded.

User response: Ensure the elliptic curve public/private key pair uses a valid elliptic curve.

03353082 Elliptic Curve not supported in FIPS mode.

Explanation: The EC domain parameters that are defined for the elliptic curve public or private key are not approved in FIPS mode.

User response: Ensure the elliptic curve for the public or private key is valid in FIPS mode. See Chapter 4, “System SSL and FIPS 140-2,” on page 19 for a list of elliptic curves that are supported by System SSL when running in FIPS mode.

03353083 ICSF services are unavailable.

Explanation: A cryptographic process cannot be completed because of ICSF callable services being unavailable.

User response: Ensure that ICSF is running and operating correctly.

03353084 ICSF callable service returned an error.

Explanation: Ensure that ICSF is operating correctly and if access to the ICSF callable services are protected with CSFSERV class profiles that the user ID of the application has READ access to the profiles protecting the ICSF callable services. See Table 4 on page 17 or Table 5 on page 17 for information about the required resource profile access. If the problem persists, collect a System SSL trace and contact your service representative.

User response: Ensure that ICSF is operating correctly and that the user ID of the application has appropriate access to the CSFSERV class RACF resource profiles. See Table 4 on page 17 or Table 5 on page 17 for information about required resource profile access. Collect a System SSL trace and verify the ICSF return code and reason code relating to the error. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about ICSF return and reason codes. If the problem persists contact your service representative.

03353085 ICSF PKCS #11 not operating in FIPS mode.

Explanation: While running in FIPS mode, an attempt was made to use ICSF PKCS #11 services, which were not operating in FIPS mode.

User response: Ensure that ICSF is configured to run in FIPS mode.

03353086 Incorrect key algorithm.

Explanation: A supplied key uses an algorithm type that is not suitable for the requested function. This error can occur if a non-ECC key has been supplied to an ECC related function, or if incompatible keys are supplied for certificate creation, such as a certificate containing a Diffie-Hellman key to be signed with an ECDSA key.

User response: Ensure the key supplied uses a suitable key algorithm type. Collect a System SSL trace containing the error to verify the key algorithms. Contact your service representative if the error persists.

03353087 Certificate revocation list is expired.

Explanation: The current time is after the nextUpdate time specified in the CRL.

User response: Obtain the latest copy of the CRL from the certification authority.

03353088 Cryptographic hardware does not support service or algorithm.

Explanation: A call requiring cryptographic hardware was made to ICSF. The current installation hardware does not support the service or algorithm being used.

User response: Ensure that the correct protocol is in use for your installation or that cryptographic hardware that is required for this service or algorithm is available to ICSF.

03353089 ICSF PKCS #11 services are disabled.

Explanation: An attempt was made to use ICSF PKCS #11 services, which are disabled because of an ICSF FIPS self-test failure.

User response: Stop and restart ICSF. System SSL may need restarting to regain the full hardware benefit from ICSF. Contact your service representative if the error persists.

0335308A Known Answer Test has failed when attempting to use ICSF.

Explanation: A Known Answer Test failed because of ICSF returning an error. Any further attempts to use System SSL continues to fail until the application is restarted or the executing process is reinitialized.

User response: Ensure that ICSF is running and operating correctly and that the user ID of the application has appropriate access to the CSFSERV class RACF resource profiles. See Table 4 on page 17 for information about required resource profile access. Collect a System SSL trace and verify the ICSF return code and reason code relating to the error. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about ICSF return and reason codes. If the problem persists contact your service representative.

0335308B Variable argument validate root is not valid.

Explanation: The specified variable argument validate root is not valid.

User response: Specify a valid variable argument validate root.

0335308C PKCS #11 label name not valid.

Explanation: The PKCS #11 secure key label name is not valid. This might be because the label is NULL, an empty string, or has only an equal sign (=).

User response: Verify that input label is correct.

0335308D Incorrect key attribute.

Explanation: One or more PKCS #11 attributes or parameters for a key are missing or incorrect for a requested function that is being performed. For example, a signing operation requires that for the key that is being used, the PKCS #11 sign attribute is to be TRUE. Verify that the correct key is being used for the requested function, and that all required attributes are set for that key. If using `gsk_make_enveloped_private_key_msg()`, ensure that a recipient certificate's RSA public key is valid.

User response: Verify that a certificate's PKCS #11 key attributes are correct for the function that is being performed.

0335308E PKCS #11 object was not found.

Explanation: PKCS #11 token, token object, or session object are not found.

User response: Verify that a PKCS #11 token or token object is in the TKDS data set. Also, verify that the session object is not lost because of ICSF restarting after the object is created.

0335308F An algorithm or key size is not FIPS approved for an ICSF operation.

Explanation: ICSF is in FIPS mode. A call to ICSF for cryptographic or signing support failed because the input key algorithm or size is not supported in FIPS mode. For example, an RSA key size of 512 is not supported in FIPS mode.

User response: Verify that the certificate key that is being used is a supported algorithm and size when ICSF is in FIPS mode. See Table 6 on page 19 for more information about supported algorithms and key sizes.

03353090 PKCS #11 key cannot be extracted.

Explanation: An attempt to export a PKCS #11 secure key failed because PKCS #11 attribute CKA_EXTRACTABLE is set to CK_FALSE.

User response: Verify that input label is correct. If it is correct, then the key cannot be exported.

03353093 Clear key support not available due to ICSF key policy.

Explanation: Unable to generate clear keys or PKCS #11 objects because of the caller's RACF access to CRYPTOZ class resource CLEARKEY.SYSTOK-SESSION-ONLY or CLEARKEY.token_name not permitting the generation of non-secure (clear) PKCS #11 keys.

User response: Ensure that the user ID of the application has appropriate access to the RACF CRYPTOZ class resource CLEARKEY.SYSTOK-SESSION-ONLY. If using **gskkyman**, ensure issuer also has access to resource CLEARKEY.token_name. token_name is the name of the PKCS #11 token that is being managed by **gskkyman**.

03353094 OCSF responder requires a signed request

Explanation: The OCSF responder contacted for certificate validation requires that all OCSF requests be signed.

User response: Enable OCSF request signing by specifying a valid database handle (ocspDbHandle), label of signing certificate (ocspReqlabel), and the signature algorithm (ocspReqSignatureAlgorithm) within the OCSF data source structure.

03353095 HTTP response is not valid

Explanation: The HTTP response received was not properly formatted or contents are not valid.

User response: Ensure that the HTTP 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.

03353096 OCSF response is not valid

Explanation: The OCSF ASN.1 encoded response received was not properly formatted or contents are not valid.

User response: Ensure that the OCSF responder 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.

03353097 OCSF request failed with internal responder error

Explanation: The OCSF responder contacted for certificate validation returned an internal error.

User response: Ensure that the OCSF responder 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.

03353098 OCSF response is expired

Explanation: The current time is after the OCSF response expiration time.

User response: If using the dedicated OCSF responder, ensure that the OCSF responder server is using the most recent revocation information available from the certification authority. If certificate revocation through the AIA extension is enabled, ensure that the OCSF responder servers referenced in the certificate chain are using the most recent revocation information available. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

03353099 Numeric value is not valid

Explanation: A numeric value specified as either a parameter to or as a field within the data structure is not valid.

User response: Verify that all numeric values specified contain numeric values that are within acceptable ranges.

0335309A A PKCS #7 CMS Version is not supported

Explanation: The PKCS #7 CMS version is not supported.

User response: Verify that the PKCS #7 CMS version is supported by System SSL.

0335309B Input certificate not supplied

Explanation: The required input field, certificate, is missing.

User response: Verify that the input certificate is not NULL.

0335309C Error creating OCSP request

Explanation: An internal error was encountered while creating the OCSP request to send to an OCSP responder.

User response: If OCSP request signing is enabled, verify that the signing certificate resides in the handle returned from `gsk_open_keyring()` or `gsk_open_database()` and the signing certificate is valid (start time is before the current time and is not yet expired) and contains a private key. The handle is provided through the `ocspDbHandle` field within the `gskdb_ocsp_source` structure provided on the `gsk_create_revocation_source()` routine. The signing key label is provided through the `ocspReqLabel` field within the `gskdb_ocsp_source` structure provided on the `gsk_create_revocation_source()` routine.

Collect a System SSL trace containing the error and then contact your service representative if the error persists.

0335309D Maximum response size exceeded

Explanation: When attempting to retrieve revocation information, the HTTP response exceeded the maximum configured response size for either an OCSP response or a HTTP CRL. The response size is provided through either the `ocspMaxResponseSize` or `httpCdpMaxResponseSize` fields within the `gskdb_ocsp_source` or `gskdb_cdp_source` structures provided on the `gsk_create_revocation_source()` routine.

User response: Ensure that the HTTP response maximum size is adequate for the size of the CRLs or OCSP responses that are being retrieved. If necessary, increase the maximum response size until an adequate size is provided to handle the CRLs or OCSP responses that are being retrieved. If unable to determine an adequate size, collect a System SSL trace containing the error and then contact your service representative.

0335309E HTTP server communication error

Explanation: Unable to establish a connection to contact the HTTP server or the OCSP responder to retrieve certificate revocation information.

User response: If enabled for OCSP and a dedicated OCSP responder is enabled, ensure that the responder is running and can be accessed.

If enabled for OCSP responders identified in the certificate AIA extension, ensure that the OCSP responders specified in the extension are running and can be accessed.

If HTTP CRL support is enabled, ensure that the HTTP server specified in the CRL Distribution Point extension is running and can be accessed.

If there is a firewall in place and either an HTTP proxy server or port and/or an OCSP proxy server or port has been identified, ensure that the servers and ports settings are correct and the servers can be accessed. The proxy server and ports are specified through the `gskdb_ocsp_source` and `gskdb_cdp_source` structures provided on the call to the `gsk_create_revocation_source()` routine.

Collect a System SSL trace containing the error and then contact your service representative if the error persists.

0335309F Variable argument security level is not valid

Explanation: The specified variable argument security level is not valid.

User response: Specify a valid variable argument security level.

033530A0 Extended key usage input count is not valid

Explanation: The extended key usage input count must be at least 1.

User response: Verify that the extended key usage count is equal to the number of extended key usages provided. The count may not be less than 1.

-
- | **033530A1 Extended key usage input is not supplied**
- | **Explanation:** The extended key usage structure `x509_key_usages` is NULL or the `x509_key_purpose` structure within `x509_key_usages` is NULL.
- | **User response:** Verify that `x509_key_usages` is not NULL and that `x509_key_purpose` is not NULL within `x509_key_usages`.
-
- | **033530A2 Extended key usage comparison failed**
- | **Explanation:** The extended key usages comparison failed. The user-supplied validation parameter and extended key purpose list did not compare favorably to the certificate's extended key usage extension values.
- | **User response:** The extended key usage comparison failed. Verify that the desired validation option is correct for your use.
-
- | **033530A3 Extended key usage type is not supported for this operation**
- | **Explanation:** The extended key usage type should be a value defined within `x509_purpose_type`. Purpose types `x509_purpose_unknown` and `x509_purpose_maximum` are invalid. Use special case `GSKCMS_VALIDATE_EXTENDED_KEY_USAGE_CHECK_OID` only if OID comparison is required.
- | **User response:** Verify that the input `x509_purpose_type` is defined within the `x509_purpose_type` and is not `x509_purpose_unknown`, `x509_purpose_maximum`, or special case `GSKCMS_VALIDATE_EXTENDED_KEY_USAGE_CHECK_OID`.
-
- | **033530A4 Certificate does not have an extended key usage extension**
- | **Explanation:** The specified certificate does not have an extended key usage extension.
- | **User response:** Verify that the input `x509_certificate` has an extended key usage extension defined.
-
- | **033530A5 Nonce in OCSP response does not match value in OCSP request**
- | **Explanation:** When validating the nonce in the OCSP response, the value did not match the value sent in the OCSP request.
- | **User response:** If OCSP is enabled for the dedicated OCSP responder, ensure that the OCSP responder server is configured to send a nonce in OCSP responses.
- | Ensure that nonce checking is required. If not required, set `ocspCheckNonce` to FALSE within the `gskdb_ocsp_source` structure provided on the `gsk_create_revocation_source()` routine.
- | Collect a System SSL trace containing the error and then contact your service representative if the error persists.
-
- | **033530A6 OCSP response not received within configured time limit**
- | **Explanation:** The time limit indicated in the value for `ocspResponseTimeout` in the OCSP data source has been exceeded.
- | **User response:** Ensure that the HTTP server where the OCSP responder resides is available and able to process OCSP requests. Verify that the value for `ocspResponseTimeout` in the OCSP data source is sufficient to receive a complete response from the HTTP server containing the OCSP responder.
-
- | **033530A7 Revocation information is not yet valid**
- | **Explanation:** The current time is earlier than the validity period of the revocation information provided through either an OCSP response or CRL.
- | **User response:** Ensure that the system time is configured correctly. Collect a System SSL trace containing the error and then contact your service representative if the problem persists.
-

033530A8 HTTP server host name is not valid

Explanation: The URI value in the AIA extension or the CDP extension is not in the correct format or cannot be resolved by the Domain Name Service (DNS). The correct URI format is `http://hostname[:portNumber]`.

User response: If an OCSP data source has been provided and the `ocspEnable` parameter is enabled, verify that the certificate being verified has a URI value in the AIA extension that is properly formatted and can be resolved by the DNS. It may be necessary to obtain a new certificate or to specify the `ocspProxyServerName` and `ocspProxyServerPort` parameters if there is a need to pass through a firewall. If the `ocspURL` or `ocspProxyServerName` parameters are specified, verify that the host name and the IP address is properly formatted and can be resolved by the DNS.

If an CDP data source has been provided and the `cdpEnableFlags` parameter is enabled for HTTP URIs, verify that the certificate being verified has a URI value in the CDP extension that is properly formatted and can be resolved by the DNS. It may be necessary to obtain a new certificate or to specify the `httpCdpProxyServerName` and `httpCdpProxyServerPort` parameters if there is a need to pass through a firewall.

033530A9 An internal error has occurred.

Explanation: The System SSL runtime library detected an internal processing error.

User response: Collect a System SSL trace containing the error and then contact your service representative.

033530AA Required basic constraints certificate extension is missing

Explanation: During the certificate validation processing for mode RFC 2459 or ANY, an intermediate CA certificate was encountered outside of a trusted certificate source which does not have a basic constraints extension.

User response: The specified untrusted certificate sources that was passed on either the `gsk_validate_certificate_mode()` or `gsk_validate_certificate()` API needs to be examined to determine which intermediate CA certificate is being disallowed. Either the intermediate CA certificate should be replaced with a valid Version 3 certificate or if the usage of the CA certificate is acceptable, the certificate needs to be moved to a trusted certificate source and removed from the untrusted certificate source.

If the error persists after adding the certificates, or if the certificates can not be readily obtained, collect a System SSL trace containing the error and then contact your service representative.

033530AB PKCS #12 input certificate has no subject DN or friendly name

Explanation: When reading the certificates of the specified PKCS #12 file, a certificate was encountered that had no subject distinguished name or PKCS #12 friendly name.

User response: Verify that all certificates within the provided PKCS #12 file have either a subject distinguished name or a friendly name attribute. The friendly name attribute or the subject distinguished name is used to create the certificate's label.

033530AC PKCS #12 file name may not end with .kdb, .rdb or .sth

Explanation: A PKCS #12 file name cannot end with .kdb, .rdb or .sth.

User response: Verify that the PKCS #12 file name does not end with .kdb, .rdb or .sth. If it does, it needs to be renamed.

033530AD Required parameter is not set

Explanation: One or more required parameters in the `gskdb_ocsp_source`, `gskdb_cdp_source`, or `gskdb_extended_directory_source` structures within the `gskdb_source` structure are not correctly set.

User response: If using the `gsk_create_revocation_source()` routine to create an OCSP data source, the `ocspEnable` parameter must be set to TRUE or the `ocspURL` parameter must be specified. If `ocspReqLabel` is set, the `ocspDbHandle` must be specified. The `ocspDbHandle` is the database handle returned by the `gsk_open_database()` routine or the `gsk_open_keyring()` routine that contains the name of the certificate specified in the `ocspReqLabel` parameter.

If using the `gsk_create_revocation_source()` routine to create an CDP data source, the `httpCdpEnableFlags` must be set to either `GSKCMS_CDP_ENABLE_HTTP` or `GSKCMS_CDP_ENABLE_ALL`.

| If using the `gsk_create_revocation_source()` routine to create an LDAP extended directory source, the `ldapServerName` parameter must be specified.

| **033530AE** **Maximum number of locations allowed to be contacted during certificate validation has been reached**

| **Explanation:** The number of locations allowed by either the `max_source_rev_ext_loc_values` or `max_validation_rev_ext_loc_values` parameters to `gsk_validate_certificate_mode()` has been exceeded. The locations for revocation information are specified by the `accessLocation` in the AIA certificate extension for OCSP and the `distributionPoint` in the CDP extension for HTTP CRLs.

| **User response:** Use the values in the certificate chain being validated to determine the proper value for the `max_source_rev_ext_loc_values` parameter, the `max_validation_rev_ext_loc_values` parameter, or both. The value for `max_source_rev_ext_loc_values` must be greater than or equal to the maximum number of location values in a certificate CDP or AIA extensions. The value for `max_validation_rev_ext_loc_values` must be greater than or equal to the total number of location values in all CDP and AIA extensions used in the certificate chain being validated. Collect a System SSL trace containing the error and then contact your service representative if the problem persists.

| **033530AF** **HTTP response not received within configured time limit**

| **Explanation:** The time limit indicated in the value for `httpCdpResponseTimeout` in the CDP data source has been exceeded.

| **User response:** Ensure that the HTTP server is available and able to process HTTP CRL requests. Verify that the value for `httpCdpResponseTimeout` in the CDP data source is sufficient to receive a complete response from the HTTP server.

| **033530B0** **LDAP response not received within configured time limit**

| **Explanation:** The time limit indicated in the value for `ldapResponseTimeout` in the extended directory data source or the time out value specified for the basic directory data source has been exceeded. The basic directory data source time out is specified on the `gsk_set_directory_numeric_value()` routine

| **User response:** Ensure that the LDAP server is available and able to process LDAP CRL requests. Verify that the time out value for the basic or extended directory data source is sufficient to receive a complete response from the LDAP server.

| **033530B1** **An unknown error has occurred**

| **Explanation:** System SSL has detected an unknown processing error.

| **User response:** Collect a System SSL trace containing the error and then contact your service representative.

| **033530B2** **OCSP request failed with try later error**

| **Explanation:** The OCSP responder is unable to currently process the OCSP request.

| **User response:** Contact the OCSP responder administrator to verify that the OCSP responder is working properly. Then retry the OCSP request at a later time.

SSL started task messages (GSK01nnn)

Messages from the SSL started task (GSKSRVR) have the prefix "GSK01". These status codes include informational messages, including errors that require a user response.

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 try the request again.

GSK01003I SSL server initialization complete.

Explanation: The server initialization is complete.

User response: None

GSK01004I SSL server shutdown requested.

Explanation: The system operator entered a STOP command for the SSL server.

User response: None

GSK01005E 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 off 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 pdsname.SIEALNKE data set to the list of APF-authorized data sets and then restart the GSKSRVR started task. If you are using a STEPLIB or JOBLIB for the GSKSRVR started task, verify that all data sets 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 that is returned by the SYSEVENT system service.

User response: Verify that the GSKSRVR started task is APF-authorized. See the SYSEVENT description in *z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO* for more information. Contact your service representative if the error persists.

GSK01013I SSL server restart registration complete on system.

Explanation: The GSKSRVR started task successfully registered with ARM (Automatic Restart Management) on the indicated system. The GSKSRVR started task is automatically restarted if it fails unexpectedly (it does not restart 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 that are 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: See the IXCARM description in *z/OS MVS Programming: Sysplex Services Reference* for more information. Contact your service representative if the error persists.

GSK01016E • GSK01022E

GSK01016E Unable to unregister for restart: Error *error-code*, **Reason** *reason-code*.

Explanation: The GSKSRVR started task is unable to unregister with ARM (Automatic Restart Management). The IXCARM request failed with the indicated error and reason codes.

User response: See the IXCARM description in *z/OS MVS Programming: 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 successfully deregistered with ARM (Automatic Restart Management) on the indicated system. The SSL server is no longer 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: See the `pthread_mutex_init()` description in *z/OS XL C/C++ Runtime 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: See the `pthread_mutex_lock()` description in *z/OS XL C/C++ Runtime 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: See the `pthread_create()` description in *z/OS XL C/C++ Runtime 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 that the failing system function and the reason code are the error code that is 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 ETCR request failed.
- 8 = The IEANTCR request failed.
- 9 = The CTRACE DEFINE request failed.

User response: Verify that the GSKSRVR started task is APF-authorized. See the system function description in *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG* 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: See the DSPSERV or ALESERV description in *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN* 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: See the IXCJOIN or IXCQUERY description in *z/OS MVS Programming: 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 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 that are allowed for a multi-line write-to-operator message is 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. See the DSPSERV description in *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN* 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 that is 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 = IXCMMSGO failed. The reason code contains the IXCMMSGO return code in the upper halfword and the IXCMMSGO reason code in the lower halfword.
- 9 = IXCMMSGI failed on the target system. The reason code contains the IXCMMSGI return code in the upper halfword and the IXCMMSGI 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 = IXCMISGI failed on the local system. The reason code contains the IXCMISGI return code in the upper halfword and the IXCMISGI reason code in the lower halfword.

User response: The request is not processed. See the IXCMISGI or IXCMISGO description in *z/OS MVS Programming: Sysplex Services Reference* for more information. Contact your service representative if the error persists.

GSK01035E SSL server is not available.

Explanation: The SSL server task is not available. This error occurs 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 try the failing request again.

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 *errorcode*, Reason *reasoncode*.

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: Verify that the GSKSRVR started task is running on the local system. Contact your service representative if the error persists.

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. See Appendix A, "Environment variables," on page 667 for the description of the GSK_TRACE environment variable for more information about 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 that is 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 that are 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 that are 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 data set. Trace entries are discarded until the trace writer emptied one of the trace buffers.

User response: Increase the trace buffer size that is 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 user ID of the GSKSRVR started task must be defined to the started procedure. See "Configuring the SSL started task" on page 588 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 started for the indicated process. This message is displayed each time component trace is started for each SSL process whose job name matches one of the job names that are 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 encountered a severe hardware encryption error during ICSF hardware processing. Encryption functions are processed in software. See message GSK01052W in 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 encountered a severe hardware encryption error. Hardware processing for the specified algorithm has been disabled. Any future encryption or decryption using this algorithm is 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 *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: See “CMS status codes (03353xxx)” on page 633 for information about 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. Status *status-code*

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 that is specified for the environment variable was not GSK_FIPS_STATE_ON, so 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 continues 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, see “CMS status codes (03353xxx)” on page 633 for information about 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 executes 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

GSK01064I

GSK01064I GSK_FIPS_ICSF_TRACKING environment variable is no longer supported.

Explanation: The GSK_FIPS_ICSF_TRACKING environment variable is not supported after z/OS V1R13, and the setting that is specified in the SSL started task environment variable file is ignored.

User response: Remove the GSK_FIPS_ICSF_TRACKING environment variable from the SSL started task environment variable file.

Utility messages (GSK00nnn)

System SSL utility messages have the prefix "GSK00". These messages identify conditions from utilities (such as **gsktrace** and System SSL run time) that require a system operator response.

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 file system 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 that 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 run time.

GSK00004R Enter password:

Explanation: The System SSL run time needs a database or certificate password.

User response: Enter the requested password.

GSK00005R Re-enter password:

Explanation: The System SSL run time 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 run time.

User response: Process the trace file using the **gsktrace** command that is at the same level as the System SSL run time which created the trace file.

GSK00007R Enter new password:

Explanation: The System SSL run time 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 parentheses. See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information about the function and return code value.

User response: See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information about the reported function and return code. If the problem cannot be resolved, contact your service representative.

GSK00009E

GSK00009E Problem encountered with the installation of the gskkyman utility.

Explanation: During installation, the sticky bit is set on for the **gskkyman** utility. If the sticky bit is turned off, attempts to invoke **gskkyman** fails.

User response: Verify that the sticky bit is set. If not set, set the sticky bit.

To check the sticky bit setting, issue:

```
ls -l /usr/lpp/gskssl/bin/gskkyman
```

The first part of the output should be:

```
-rwxr-xr-t
```

The t indicates that the sticky bit is on.

To set the sticky bit on, issue the following command from an authorized ID:

```
chmod +t /usr/lpp/gskssl/bin/gskkyman
```

Appendix A. Environment variables

These tables contain all the environment variables used by the System SSL application and read during the startup of the application.

Table 16. SSL-Specific environment variables

Environment variables	Usage	Valid values
GSK_AIA_CDP_PRIORITY	Specifies the priority order that the AIA and the CDP extensions are checked for certificate revocation information.	<p>A value of 1 or ON indicates that the AIA extension is queried before examining the CDP extension. This means that any OCSF responders specified in the AIA extension or the OCSF responder specified in GSK_OCSP_URL is contacted before attempting to contact the HTTP servers specified in the URI values of the CDP extension.</p> <p>A value of 0 or OFF indicates that the CDP extension is queried before examining the AIA extension. This means that the HTTP servers specified in the URI values of the CDP extension is contacted before attempting to contact the OCSF responders in the AIA extension or the OCSF responder specified in GSK_OCSP_URL.</p> <p>The default value is ON.</p>
GSK_CERT_VALIDATE_KEYRING_ROOT	Specifies how certificates in a SAF key ring are validated.	<p>A value of ON or 1 specifies that SAF key ring certificates must be validated to the root CA certificate.</p> <p>Specify OFF or 0 if SAF key ring certificates are only validated to the trust anchor certificate. If a sole intermediate certificate is found in a SAF key ring and the next issuer is not found in the same SAF key ring, the intermediate certificate acts as a trust anchor and the certificate chain is considered complete. By default, SAF key ring certificates are only validated to the trust anchor certificate. This setting does not affect the validation of SSL key database file, PKCS #12 file, or PKCS #11 token certificates because these certificates are always validated to the root CA certificate. The default value is OFF.</p>
GSK_CERT_VALIDATION_MODE	Specifies which Internet standard is to be used for certificate validation.	<p>A value of 2459 specifies certificate validation against RFC 2459 only. A value of 3280 specifies certificate validation against RFC 3280 only. A value of 5280 specifies certificate validation against RFC 5280 only. A value of ANY specifies certificate validation against RFC 2459 initially - if that fails, validate against RFC 3280 - if that fails, validate against RFC 5280. The default value is ANY.</p>

Environment variables

Table 16. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_CLIENT_AUTH_NOCERT_ALERT	Specifies whether the SSL server application accepts a connection from a client where client authentication is requested and the client fails to supply an X.509 certificate.	A value of OFF or 0 allows connections with clients where client authentication is requested and the client fails to supply an X.509 certificate. A value of ON or 1 terminates connections with clients where client authentication is requested and the client fails to supply an X.509 certificate. The default value is OFF.
GSK_CLIENT_ECURVE_LIST	<p>Specifies the list of elliptic curves that are supported by the client as a string consisting of 1 or more 4-character values in order of preference for use. The list is used by the client to guide the server as to which elliptic curves are preferred when using ECC-based cipher suites for TLS V1.0 and higher protocols.</p> <p>Only NIST recommended curves can be specified. To use Brainpool standard curves for an SSL environment or connection, set GSK_CLIENT_ECURVE_LIST to "" or use gsk_attribute_set_buffer() to re-initialize the GSK_CLIENT_ECURVE_LIST buffer to NULL.</p> <p>See Table 22 on page 695 for a list of valid 4-character elliptic curve specifications.</p>	The default specification is 00210023002400250019.
GSK_CRL_CACHE_ENTRY_MAXSIZE	Specifies the maximum size in bytes of a CRL to be kept in the LDAP CRL cache.	<p>The valid cache entry sizes are 0 through 2147483647.</p> <p>The default value is 0, which means there is no limit on the size of a CRL that is allowed to be stored in the LDAP CRL cache.</p> <p>The size must be greater than or equal to 0.</p>

Table 16. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_CRL_CACHE_EXTENDED	<p>Specifies that LDAP extended CRL cache support is enabled.</p> <p>Enabling extended support:</p> <ul style="list-style-type: none"> LDAP CRLs are only cached when there is an expiration time present and it is greater than the current time. Limits the number of CRLs that can be stored in the LDAP cache to 32. This can be overridden by specifying GSK_CRL_CACHE_SIZE. Disables caching of temporary CRLs. This can be enabled by specifying GSK_CRL_CACHE_TEMP_CRL. Ignores GSK_CRL_CACHE_TIMEOUT. <p>When disabled, LDAP basic CRL caching can be used and retrieved LDAP CRLs are only cached when GSK_CRL_CACHE_TIMEOUT is greater than 0 and GSK_CRL_CACHE_SIZE is set to a non-zero number.</p>	<p>A value of ON or 1 enables LDAP extended CRL caching.</p> <p>A value of OFF or 0 disables LDAP extended CRL caching.</p> <p>The default value is OFF.</p>
GSK_CRL_CACHE_SIZE	<p>Specifies the maximum number of CRLs that are allowed to be stored in the LDAP CRL cache.</p>	<p>The valid cache sizes are -1 through 32000.</p> <p>A value of -1 means unlimited while a value of 0 means caching is not enabled.</p> <p>If LDAP extended CRL cache support is enabled, the default is 32 and caching only occurs if the CRL contains an expiration time that is later than the current time.</p> <p>If LDAP basic CRL cache support is enabled, the default is unlimited or -1 and caching only occurs when GSK_CRL_CACHE_TIMEOUT is greater than 0.</p>
GSK_CRL_CACHE_TEMP_CRL	<p>Specifies if a temporary LDAP CRL cache entry is added to the LDAP CRL cache when the CRL does not reside on the LDAP server.</p>	<p>A value of ON or 1 indicates that a temporary LDAP CRL cache entry is added to the LDAP CRL cache.</p> <p>A value of OFF or 0 indicates that a temporary LDAP CRL cache entry is not to be added to the LDAP CRL cache.</p> <p>If LDAP extended CRL cache support is enabled, the default value is OFF.</p> <p>If LDAP basic CRL cache support is enabled, the default value is ON.</p>

Environment variables

Table 16. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_CRL_CACHE_TEMP_CRL_TIMEOUT	<p>Specifies the time in hours that a temporary CRL cache entry resides in the LDAP extended CRL cache when caching of temporary CRLs is enabled.</p> <p>A temporary LDAP CRL cache entry is added to the LDAP CRL cache when the CRL does not reside on the LDAP server.</p>	The range is 1 through 720 hours and defaults to 24 hours.
GSK_CRL_CACHE_TIMEOUT	Specifies the number of hours that a cached LDAP CRL remains valid.	The valid timeout values are 0 through 720 and defaults to 24. A value of 0 disables the LDAP CRL cache.
GSK_CRL_SECURITY_LEVEL	<p>Specifies the level of security to be used when contacting LDAP servers to check CRLs for revoked certificates during certificate validation.</p> <p>An attempt to contact the LDAP server is performed when the CRL is not found in the LDAP cache. To enforce contact with the LDAP server for each CRL being checked, CRL caching must be disabled.</p> <p>For LDAP basic CRL caching, see the GSK_CRL_CACHE_TIMEOUT or GSK_CRL_CACHE_SIZE settings.</p> <p>For LDAP extended CRL caching, see the GSK_CRL_CACHE_SIZE setting.</p>	<p>LOW - Certificate validation does not fail if the LDAP server cannot be contacted.</p> <p>MEDIUM - Certificate validation requires the LDAP server to be contactable, but does not require a CRL to be defined. This is the default.</p> <p>HIGH - Certificate validation requires revocation information to be provided by the LDAP server.</p>
GSK_EXC_ABEND_DUMP	Specifies whether the SSL condition handler should call the cdump() service to dump the current thread before resuming the failing routine. The dump is placed in the current directory unless LE is instructed to use a different directory by the _CEE_DMPTARG environment variable. See <i>z/OS Language Environment Programming Guide</i> for more information about LE callable services.	A value of 1 enables SSL dumps and a value of 0 disables SSL dumps. The default is 0. The export file contains just the requested certificate when the DER format is selected.
GSK_EXTENDED_RENEGOTIATION_INDICATOR	Specifies the level of enforcement of renegotiation indication as specified by RFC 5746 during the initial handshake.	<p>A value of OPTIONAL does not require the renegotiation indicator during initial handshake. This is the default.</p> <p>A value of CLIENT allows the client initial handshake to proceed only if the server indicates support for RFC 5746 Renegotiation.</p> <p>A value of SERVER allows the server initial handshake to proceed only if the client indicates support for RFC 5746 Renegotiation.</p> <p>A value of BOTH will allow the server and client initial handshakes to proceed only if partner indicates support for RFC 5746 Renegotiation.</p>

Table 16. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_HTTP_CDP_CACHE_ENTRY_MAXSIZE	Specifies the maximum size in bytes of a CRL that is allowed to be stored in the HTTP CDP CRL cache. Any CRLs larger than this size are not cached.	The valid sizes are 0 through 2147483647. The default value is 0, which means there is no limit on the size of the CRL stored in the HTTP CDP CRL cache.
GSK_HTTP_CDP_CACHE_SIZE	Specifies the maximum number of CRLs that are allowed to be stored in the HTTP CDP CRL cache.	The valid sizes are 0 through 32000. The default value is 32. If set to 0, HTTP CDP CRL caching is disabled.
GSK_HTTP_CDP_ENABLE	Specifies if certificate revocation checking with the HTTP URI values in the CDP extension is enabled.	A value of 0, OFF, or DISABLED indicates that certificate revocation checking with the HTTP URI values in the CDP extension is not enabled. A value of 1, ON, or ENABLED indicates certificate revocation checking with the HTTP URI values in the CDP extension is enabled. The default value is OFF.
GSK_HTTP_CDP_MAX_RESPONSE_SIZE	Specifies the maximum size in bytes accepted as a response from an HTTP server when retrieving a CRL. Setting the maximum response size too small could implicitly disable HTTP CRL support.	The valid sizes are 0 through 2147483647. The default value is 204800 (200K). A value of 0 disables checking the size and allows a CRL of any size.
GSK_HTTP_CDP_PROXY_SERVER_NAME	Specifies the DNS name or IP address of the HTTP proxy server for HTTP CDP CRL retrieval.	The default value is NULL.
GSK_HTTP_CDP_PROXY_SERVER_PORT	Specifies the HTTP proxy server port for HTTP CDP CRL retrieval.	Port must be between 1 and 65535. The default port value is 80.
GSK_HTTP_CDP_RESPONSE_TIMEOUT	Specifies the time in seconds to wait for a response from the HTTP server.	The valid time limits are 0 through 43200 seconds (12 hours). The default value is 15 seconds and a value of 0 indicates that there is no time limit.

Environment variables

Table 16. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_HW_CRYPTO	<p>Specifies whether the hardware cryptographic support is used. Note that ICSF (Integrated Cryptographic Service Facility) must be configured and running in order for System SSL to use the hardware cryptographic support that is available in the cryptographic cards.</p> <p>SHA-1, SHA-2, DES, Triple DES, and AES hardware functions can be used without ICSF if the zArchitecture message-security assist is installed.</p> <p>For more information about hardware cryptographic support, see Chapter 3, "Using cryptographic features with System SSL," on page 11.</p> <p>Selected hardware cryptographic functions can be disabled by setting the appropriate bits to zero in the GSK_HW_CRYPTO value. The corresponding software algorithms are used when a hardware function is disabled. These bit assignments are defined:</p> <ul style="list-style-type: none"> 1 = SHA-1 digest generation 2 = 56-bit DES encryption/decryption 4 = 168-bit Triple DES encryption/decryption 8 = Public key encryption/decryption 16 = AES 128-bit encryption/decryption 32 = SHA-256 digest generation 64 = AES-256-bit encryption/decryption 128 = SHA-224 digest generation 256 = SHA-384 digest generation 512 = SHA-512 digest generation <p>Note: If a hardware function bit is set on and the hardware function is unavailable, processing takes place in software.</p>	<p>A value of 0 disables the use of hardware support while a value of 65535 enables the use of hardware support. The default value is 65535 and only available hardware support is used.</p>
GSK_KEY_LABEL	<p>Specifies the label of the key that is used to authenticate the application.</p>	<p>Any key label. The default key is used if a key label is not specified.</p>

Table 16. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_KEYRING_FILE	<p>Specifies the name of the key database file, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. A key database or PKCS #12 file is used if the GSK_KEYRING_PW environment variable is also specified. A key database file is used if GSK_KEYRING_STASH environment variable is also specified. Otherwise, a SAF key ring or z/OS PKCS #11 token is used.</p> <p>Note that certificate private keys are not available when using a SAF key ring owned by another user.</p> <p>The user must have READ access to resource USER.tokenname in the CRYPTOZ class when using a z/OS PKCS #11 token.</p>	<p>The SAF key ring name is specified as userid/keyring. The current user ID is used if the user ID is omitted.</p> <p>The z/OS PKCS #11 token name is specified as *TOKEN*/token-name.</p> <p>If no certificate source is specified, defaults to NULL.</p>
GSK_KEYRING_PW	Specifies the password for the key database or PKCS #12 file.	<p>NULL or value consisting of up to 128 characters.</p> <p>The default value is NULL</p>
GSK_KEYRING_STASH	Specifies the name of the key database password stash file.	<p>The stash file name always has an extension of .sth and the supplied name is changed if it does not have the correct extension. The GSK_KEYRING_PW environment variable is used instead of the GSK_KEYRING_STASH environment variable if it is also specified.</p> <p>The default value is NULL.</p>
GSK_LDAP_PASSWORD	Specifies the password to use when connecting to the LDAP server.	The default value is NULL.
GSK_LDAP_PORT	Specifies the LDAP server port.	Port must be between 1 and 65535. Port 389 is used if no LDAP server port is specified.
GSK_LDAP_RESPONSE_TIMEOUT	Specifies the time in seconds to wait for a response from the LDAP server.	<p>The valid time limits are 0 through 43200 seconds (12 hours).</p> <p>The default value is 15 seconds and a value of 0 indicates that there is no time limit.</p>
GSK_LDAP_SERVER	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.	<p>Each host name can contain an optional port number that is separated from the host name by a colon.</p> <p>The default value is NULL.</p>
GSK_LDAP_USER	Specifies the distinguished name to use when connecting to the LDAP server.	The default value is NULL.

Environment variables

Table 16. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_MAX_SOURCE_REV_EXT_LOC_VALUES	Specifies the maximum number of location values that are contacted per data source when attempting validation of a certificate. The locations for revocation information are specified by the <i>accessLocation</i> in the AIA certificate extension for OCSP and the <i>distributionPoint</i> in the CDP extension for HTTP CRLs. When an HTTP URI is present in an AIA or CDP extension, validation attempts to contact the remote HTTP server to obtain revocation information. Both of these extensions can contain multiple location values and therefore have the potential to impact performance when there be a very large number of locations present.	The valid values are 0 through 256. The default value is 10 and a value of 0 indicates there is no limit on the number of locations contacted.
GSK_MAX_VALIDATION_REV_EXT_LOC_VALUES	Specifies the maximum number of locations values that are contacted when performing validation of a certificate. The locations for revocation information are specified by the <i>accessLocation</i> in the AIA certificate extension for OCSP and the <i>distributionPoint</i> in the CDP extension for HTTP CRLs. When an HTTP URI is present in an AIA or CDP extension, validation attempts to contact the remote HTTP server to obtain revocation information. Both of these extensions can contain multiple location values and therefore has the potential to negatively impact performance when there be a very large number of locations present.	The valid values are 0 through 1024. The default value is 100 and a value of 0 indicates there is no limit on the number of locations contacted.
GSK_OCSP_CLIENT_CACHE_ENTRY_MAXSIZE	Specifies the maximum number of OCSP responses or cached certificate statuses that are allowed to be kept in the OCSP response cache for an issuing CA certificate.	The valid sizes are 0 through 32000. The default value is 0 which indicates that there is no limit on the number of cached certificate statuses allowed for a specific issuing CA certificate other than the limit imposed by GSK_OCSP_CLIENT_CACHE_SIZE. This cache size is rounded up to the nearest multiple of 16 with a minimum size of 16.
GSK_OCSP_CLIENT_CACHE_SIZE	Specifies the maximum number of OCSP responses or cached certificate statuses to be kept in the OCSP response cache.	The valid cache sizes are 0 through 32000 and defaults to 256. The OCSP response cache is disabled if 0 is specified. The OCSP response cache is allocated using the requested size rounded up to the nearest multiple of 16 with a minimum size of 16.

Table 16. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_OCSP_ENABLE	<p>Specifies whether the AIA extensions are to be used for revocation checking.</p> <p>If GSK_OCSP_URL is specified, GSK_OCSP_ENABLE is set to ON and GSK_OCSP_URL_PRIORITY is set to ON, then the order the responders are used is GSK_OCSP_URL defined responder first and then the responders identified in the AIA extension.</p> <p>If GSK_OCSP_URL is specified, GSK_OCSP_ENABLE is set to ON and GSK_OCSP_URL_PRIORITY is set to OFF, then the order that responders are used is the responders identified in the AIA extension first and then the GSK_OCSP_URL defined responder.</p>	<p>A value of 0, OFF, or DISABLED disables OCSP revocation checking via the AIA extension.</p> <p>A value of 1, ON, or ENABLED enables OCSP revocation checking via the AIA extension.</p> <p>The default value is OFF.</p>
GSK_OCSP_MAX_RESPONSE_SIZE	<p>Specifies the maximum size in bytes that is accepted as a response from an OCSP responder. Setting the maximum response size too small could implicitly disable OCSP support.</p>	<p>The valid response sizes are 0 through 2147483647.</p> <p>The default value is 20480 (20K).</p> <p>A value of 0 disables checking of the OCSP response size and allows an OCSP response of any size.</p>
GSK_OCSP_NONCE_CHECK_ENABLE	<p>Specifies if OCSP response nonce checking is enabled. Nonce checking ensures the nonce in the OCSP response matches the nonce sent in the OCSP request.</p> <p>Note: Setting to ON sets GSK_OCSP_NONCE_GENERATION_ENABLE to ON.</p>	<p>A value of 0, OFF, or DISABLED disables OCSP nonce checking.</p> <p>A value of 1, ON, or ENABLED enables OCSP nonce checking.</p> <p>The default value is OFF.</p>
GSK_OCSP_NONCE_GENERATION_ENABLE	<p>Specifies if OCSP requests include a generated nonce.</p>	<p>A value of 0, OFF, or DISABLED disables OCSP nonce generation.</p> <p>A value of 1, ON, or ENABLED enables OCSP nonce generation.</p> <p>The default value is OFF.</p>
GSK_OCSP_NONCE_SIZE	<p>Specifies the size in bytes for the value of the nonce to be sent in OCSP requests.</p>	<p>The valid OCSP nonce sizes are 8 through 256 and defaults to 8.</p>
GSK_OCSP_PROXY_SERVER_NAME	<p>Specifies the DNS name or IP address of the OCSP proxy server.</p>	<p>The default value is NULL.</p>
GSK_OCSP_PROXY_SERVER_PORT	<p>Specifies the OCSP responder proxy server port.</p>	<p>Port must be between 1 and 65535. The default port value is 80.</p>
GSK_OCSP_REQUEST_SIGALG	<p>Specifies the hash and signature algorithm pair used to sign OCSP requests.</p> <p>Only requests sent to the OCSP responder identified by GSK_OCSP_URL are signed and not the ones selected from a certificate AIA extension.</p> <p>See Table 23 on page 695 for a list of valid 4-character signature algorithm pairs specifications.</p>	<p>Default is 0401 (RSA with SHA256).</p>

Environment variables

Table 16. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_OCSP_REQUEST_SIGKEYLABEL	Specifies the label of the key used to sign OCSF requests. Only requests sent to the OCSF responder identified by GSK_OCSP_URL are signed.	Any key label. OCSF requests are not signed if a key label is not specified.
GSK_OCSP_RESPONSE_TIMEOUT	Specifies the time in seconds to wait for a response from the OCSF responder server.	The valid time limits are 0 through 43200 seconds (12 hours). The default value is 15 seconds and a value of 0 indicates that there is no time limit.
GSK_OCSP_RETRIEVE_VIA_GET	Specifies if the HTTP GET method should be used when sending an OCSF request.	A value of 0 or OFF sends the OCSF request via the HTTP POST method. A value of 1 or ON sends the OCSF request via the HTTP GET method when the total request size after Base64 encoding is less than 255 bytes. The default value is OFF.
GSK_OCSP_URL	Specifies the URI of an OCSF responder. The OCSF responder is used to obtain certificate revocation status during certificate validation. A certificate does not need an AIA extension if a responder URL is configured using this option. If GSK_OCSP_URL is specified, GSK_OCSP_ENABLE is set to ON, and GSK_OCSP_URL_PRIORITY is set to ON, the order that responders are used is GSK_OCSP_URL defined responder first and then the responders identified in the AIA extension. If GSK_OCSP_URL is specified, GSK_OCSP_ENABLE is set to ON, and GSK_OCSP_URL_PRIORITY is set to OFF, the order that responders are used is the responders identified in the AIA extension first and then the GSK_OCSP_URL defined responder.	The value must conform to the definition of an HTTP url: <pre>http_URL = "http:" "/" host [":" port] [abs_path ["?" query]]</pre> where <i>host</i> can be an IPv4 or IPv6 IP address, or a domain name. The default value is NULL.
GSK_OCSP_URL_PRIORITY	Specifies the priority order for contacting OCSF responder locations if both GSK_OCSP_URL and GSK_OCSP_ENABLE are active.	A value of 1 or ON indicates that the order that responders are used is the GSK_OCSP_URL defined responder first and then the responders identified in the AIA extension. A value of 0 or OFF indicates that the order that responders are used is the responders identified in the AIA extension first and then the GSK_OCSP_URL defined responder. The default value is ON.

Table 16. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_PROTOCOL_SSLV2	Specifies whether the SSL V2 protocol is supported. The SSL V2 and SSL V3 protocols should be disabled whenever possible because the TLS V1.0, TLS V1.1, and TLS V1.2 protocols provide significant security enhancements. This variable has no effect when operating in FIPS mode.	A value of 0, OFF or DISABLED disables the SSL V2 protocol while a value of 1, ON or ENABLED enables the SSL V2 protocol. The default value is OFF.
GSK_PROTOCOL_SSLV3	Specifies whether the SSL V3 protocol is supported. The SSL V2 and SSL V3 protocols should be disabled whenever possible because the TLS V1.0, TLS V1.1, and TLS V1.2 protocols provide significant security enhancements. This variable has no effect when operating in FIPS mode.	A value of 0, OFF or DISABLED disables the SSL V3 protocol while a value of 1, ON or ENABLED enables the SSL V3 protocol. The default value is OFF.
GSK_PROTOCOL_TLSV1	Specifies whether the TLS V1.0 protocol is supported.	A value of 0, OFF or DISABLED disables the TLS V1.0 protocol while a value of 1, ON or ENABLED enables the TLS V1.0 protocol. The default value is ON.
GSK_PROTOCOL_TLSV1_1	Specifies whether the TLS V1.1 protocol is supported.	A value of 0, OFF or DISABLED disables the TLS V1.1 protocol while a value of 1, ON or ENABLED enables the TLS V1.1 protocol. The default value is OFF.
GSK_PROTOCOL_TLSV1_2	Specifies whether the TLS V1.2 protocol is supported.	A value of 0, OFF or DISABLED disables the TLS V1.2 protocol. A value of 1, ON or ENABLED enables the TLS V1.2 protocol. The default value is OFF.
GSK_RENEGOTIATION	Specifies the type of session renegotiation allowed for an SSL environment.	<p>A value of NONE disables SSL V3 and TLS handshake renegotiation as a server and allow RFC 5746 renegotiation. This is the default.</p> <p>A value of DISABLED disables SSL V3 and TLS handshake renegotiation as a server and also disable RFC 5746 renegotiation.</p> <p>A value of ALL allows SSL V3 and TLS handshake renegotiation as a server while also allowing RFC 5746 renegotiation.</p> <p>A value of ABBREVIATED allows SSL V3 and TLS abbreviated handshake renegotiation as a server for resuming the current session only, while disabling SSL V3 and TLS full handshake renegotiation as a server. With this value specified, the System SSL session ID cache is not checked when resuming the current session. RFC 5746 renegotiation is allowed if this value is specified.</p>

Environment variables

Table 16. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_RENEGOTIATION_PEER_CERT_CHECK	Specifies if the peer certificate is allowed to change during renegotiation.	<p>A value of OFF or 0 does not perform an identity check against the peer's certificate during renegotiation. This allows the peer certificate to change during renegotiation. This is the default.</p> <p>A value of ON or 1 performs a comparison against the peer's certificate to ensure that certificate does not change during renegotiation.</p>
GSK_REVOCATION_SECURITY_LEVEL	<p>Specifies the level of security to be used when contacting an OCSP responder or an HTTP server specified in a URI value of the CDP extension.</p> <p>An attempt to contact either an OCSP responder or HTTP server is performed when revocation information is not found in cache. To enforce contact with either the OCSP responder or HTTP server for each validation, caching must be disabled.</p> <p>For OCSP caching, see GSK_OCSP_CLIENT_CACHE_SIZE.</p> <p>For HTTP CRL caching, see GSK_HTTP_CDP_CACHE_SIZE.</p>	<p>A value of LOW indicates that certificate validation does not fail if the OCSP responder or HTTP server specified in the URI value of the CDP extension cannot be contacted.</p> <p>A value of MEDIUM requires the OCSP responder or the HTTP server in a URI value in the CDP extension to be contactable. For an OCSP responder, it must be able to provide a valid certificate revocation status. If the certificate status is revoked or unknown, certificate validation fails. For an HTTP server in a CDP extension, it must be contactable and able to provide an CRL.</p> <p>A value of HIGH requires revocation information to be provided by the OCSP responder or HTTP server. If OCSP revocation checking with the AIA extension is enabled, there must be HTTP URI values present in the certificate that are able to be contactable and able to provide a valid certificate revocation status. If HTTP CRL checking is enabled, there must be HTTP URI values in the CDP extension that are able to be contactable and able to provide a CRL.</p> <p>The default value is MEDIUM.</p>
GSK_RNG_ALLOW_ZERO_BYTES	<p>Specifies whether the SSL random number generator, gsk_generate_random_bytes includes bytes with a zero value in the random byte output stream, or remove them.</p> <p>The GSK_RNG_ALLOW_ZERO_BYTES environment variable is processed during System SSL initialization and is not checked afterward.</p>	<p>A value of TRUE, ON or 1 sets the random number generator to retain bytes with a zero value in the output stream. A value of FALSE, OFF or 0 results in bytes with a zero value being removed. The default setting is TRUE.</p>
GSK_SSL_HW_DETECT_MESSAGE	Setting this environment variable to 1 causes a series of messages to be written to stderr during System SSL initialization. These messages displays 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.	Specify 1 to have messages written. Any other value is ignored, which is the default.

Table 16. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_SSL_ICSF_ERROR_MESSAGE	Setting this environment variable to 1 causes a message to be written to stderr when an ICSF callable service returns an error. These messages are intended for diagnostic use only and are not translated based on the setting of the LANG environment variable.	Specify 1 to have messages written. Any other value is ignored, which is the default.
GSK_STDERR_FILE	Specifies the fully-qualified name of the file to receive standard error messages generated using SSL message services. Messages displayed from externally documented messages is written to stderr if this environment variable is not defined.	If fully qualified file not specified, the default action is to write standard errors to stderr.
GSK_STDOUT_FILE	Specifies the fully-qualified name of the file to receive standard output messages generated using SSL message services. Messages displayed from externally documented messages is written to stdout if this environment variable is not defined.	If fully qualified file not specified, the default action is to write standard output to stdout.
GSK_SUITE_B_PROFILE	<p>Specifies the Suite B profile to be applied to TLS sessions.</p> <p>A Suite B compliant TLS V1.2 or later client must offer only the following cipher suites when conversing with a TLS V1.2 Suite B compliant server.</p> <p>128-bit security level:</p> <ul style="list-style-type: none"> • C023 = 128-bit AES encryption with SHA-256 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate. • C02B = 128-bit AES in Galois Counter Mode encryption with SHA-256 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate. <p>192-bit security level:</p> <ul style="list-style-type: none"> • C024 = 256-bit AES encryption with SHA-384 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate. • C02C = 256-bit AES in Galois Counter Mode encryption with SHA-384 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate. 	<p>A value of OFF specifies that Suite B compliant profiles are not in use for TLS sessions. This is the default value.</p> <p>A value of 128 specifies that only ciphers defined within 128-bit Suite B compliant profile can be used for a TLS session.</p> <p>A value of 192 specifies that only ciphers defined within 192-bit Suite B compliant profile can be used for a TLS session.</p> <p>A value of ALL specifies that ciphers defined within both the 128-bit and 192-bit Suite B compliant profiles can be used for a TLS session.</p>
GSK_SYSPLEX_SIDCACHE	Specifies whether sysplex session caching is supported for this application.	A value of 0, OFF or DISABLED disables sysplex session caching while a value of 1, ON or ENABLED enables sysplex session caching. The default value is OFF.

Environment variables

Table 16. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_T61_AS_LATIN1	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.	If the GSK_T61_AS_LATIN1 environment variable is set to YES or 1, the 8-bit ISO8859-1 character set is 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 is 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 afterward. Note that selecting the incorrect character set can cause strings to be converted incorrectly.
GSK_TLS_CBC_PROTECTION_METHOD	Specifies an optional SSL V3.0 or TLS V1.0 CBC IV protection method when writing application data.	<p>A value of NONE indicates that no CBC protection is enabled. This is the default.</p> <p>A value of ZEROBYTEFRAGMENT indicates that zero byte record fragmenting is enabled. When this value is specified, a zero byte record fragment is sent before the application data records are sent.</p> <p>A value of ONEBYTEFRAGMENT indicates that one byte record fragmenting is enabled. When this value is specified, the first record is sent in two record fragments with the first record fragment containing only one byte of application data. The rest of the application data in the first record is sent in the second record fragment. All following records are written whole.</p>
GSK_TLS_SIG_ALG_PAIRS	<p>Specifies the list of hash and signature algorithm pair specifications supported by the client or server as a string consisting of 1 or more 4-character values in order of preference for use.</p> <p>The signature algorithm pair specifications are sent by either the client or server to the session partner to indicate which signature/hash algorithm combinations are supported for digital signatures.</p> <p>The signature algorithm pair specification only has relevance for sessions using TLS V1.2 or higher protocols.</p> <p>See Table 23 on page 695 for a list of valid 4-character signature algorithm pairs specifications.</p>	<p>If executing in non-FIPS mode, the default is: "060106030501050304010403030103030201020302020101"</p> <p>If executing in FIPS mode, the default is: "06010603050105030401040303010303020102030202"</p>

Table 16. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
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 The default value is 0x00.
GSK_TRACE_FILE	Specifies the name of the trace file. 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 is /tmp/gskssl.247.trc.	Must be set to the name of an UNIX System Services file in a directory for which the executing application has write permission. The default trace file is /tmp/gskssl.%.trc.
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. See Table 18 on page 687 for the list of the supported ciphers.	If United States only encryption is enabled (System SSL Security Level 3 FMID is installed), the default is 7364. Otherwise, the default is 64.
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 is disabled if 0 is specified. The session identifier cache is allocated using the requested size rounded up to a power of 2 with a minimum size of 16.

Environment variables

Table 16. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_V3_CIPHER_SPECS	<p>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 higher protocols.</p> <p>For protocols TLS V1.1 and higher, export cipher suites is not used. 40-bit ciphers are ignored if these security protocols are negotiated.</p> <p>For protocols TLS V1.2 and higher, 56-bit DES cipher suites are not used. DES ciphers are ignored if these communications protocols are negotiated.</p> <p>Any ciphers that use SHA-256 or greater message authentication or use AES-GCM encryption can only be used if TLS V1.2 or higher is the negotiated protocol.</p> <p>See Table 19 on page 687 for the list of the supported 2-character ciphers.</p>	<p>If executing in non-FIPS mode and United States only encryption is enabled (System SSL Security Level 3 FMID is installed), the default is: "35363738392F303132330A1613100D0915120F0C"</p> <p>If executing in non-FIPS mode and United States only encryption is not enabled (System SSL Security Level 3 FMID is not installed), the default is: "0915120F0C"</p> <p>If executing in FIPS mode, the default is: "35363738392F303132330A1613100D"</p>
GSK_V3_CIPHER_SPECS_EXPANDED	<p>Specifies the SSL V3 cipher specifications in order of preference as a string consisting of 1 or more 4-character values. The SSL V3 cipher specifications are used for the SSL V3, TLS V1.0, and higher protocols.</p> <p>For protocols TLS V1.1 and higher export cipher suites are not used. 40-bit ciphers are ignored if these security protocols are negotiated.</p> <p>For protocols TLS V1.2 and higher, 56-bit DES cipher suites are not used. DES ciphers are ignored if these communications protocols are negotiated.</p> <p>Any ciphers that use SHA-256 or greater message authentication or use AES-GCM encryption can only be used if TLS V1.2 or higher is the negotiated protocol.</p> <p>See Table 20 on page 691 for the list of the supported 4-character ciphers.</p>	<p>If executing in non-FIPS mode and United States only encryption is enabled (System SSL Security Level 3 FMID is installed), the default is: "00350036003700380039002F0030003100320033000A001600130010000D000900150012000F000C"</p> <p>If executing in non-FIPS mode and United States only encryption is not enabled (System SSL Security Level 3 FMID is not installed), the default is: "000900150012000F000C"</p> <p>If executing in FIPS mode, the default is: "00350036003700380039002F0030003100320033000A001600130010000D"</p>
GSK_V3_SESSION_TIMEOUT	<p>Specifies the session timeout value in seconds for the SSL V3, TLS V1.0 and higher protocols.</p>	<p>The valid timeout values are 0 through 86400 and defaults to 86400. The timeout is disabled if 0 is specified.</p>
GSK_V3_SIDCACHE_SIZE	<p>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 higher protocols.</p>	<p>The valid cache sizes are 0 through 64000 and defaults to 512. The SSL V3 cache is disabled if 0 is specified. The session identifier cache is allocated by using the requested size rounded up to a power of 2 with a minimum size of 16.</p>

Table 16. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
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 17 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*.

Table 17. System environment variables used by SSL

System environment variables	Usage	Valid values
LIBPATH	Used to specify the directory to search for a DLL (Dynamic Link Library) file name. If it is not set, the working directory is searched.	
NLSPATH	Specifies where the message catalogs are to be found.	The default location is /usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/En_US.IBM-1047/%N
PATH	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).	
STEPLIB	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.	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 that 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.

Environment variables

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 three programs: **client**, **server**, and **display_certificate**. These sample files are 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. File name and password attributes are hard-coded in the sample files.

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**.

To get information about the parameters accepted when invoking the server program, issue `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.

To get information about the parameters accepted when invoking the client program, issue `client -?`

display_certificate (source file: display_certificate.c) is a program that can display an X.509 certificate stored in a file. The **display_certificate** program is only supported as a 31-bit application.

The files included in the examples are:

Sample C++ SSL files

Makefile

This file builds the example programs and DLLs. The resulting executable DLLs are **client**, **server** and **display_certificate**.

To build the examples as a 31-bit application (default), issue:

```
/bin/make
```

To build the client and server examples as a 64-bit application, issue:

```
/bin/make AMODE=64
```

Remove all compiled *.o* and *.x* artifacts, issue:

```
/bin/make clean
```

Remove all compiled *.o*, *.x* and DLL artifacts, issue:

```
/bin/make clobber
```

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.

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.

Appendix C. Cipher suite definitions

The following tables outline:

- Cipher suite definitions for SSL V2
- 2-character and 4-character cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2.
- Cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 by supported protocol, symmetric algorithm, and message authentication algorithm
- Cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 by key-exchange method and signing certificate
- Supported elliptic curve definitions for TLS V1.0, TLS V1.1, and TLS V1.2.

Table 18. Cipher suite definitions for SSL V2

Cipher number	Description	FIPS 140-2	Base security level FMID HCPT420	Security level 3 FMID JCPT421
1	128-bit RC4 encryption with MD5 message authentication (128-bit secret key)			X
2	128-bit RC4 export encryption with MD5 message authentication (40-bit secret key)		X	X
3	128-bit RC2 encryption with MD5 message authentication (128-bit secret key)			X
4	128-bit RC2 export encryption with MD5 message authentication (40-bit secret key)		X	X
6	56-bit DES encryption with MD5 message authentication (56-bit secret key)		X	X
7	168-bit Triple DES encryption with MD5 message authentication (168-bit secret key)			X

Table 19. 2-character and 4-character cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2

2-character cipher number	4-character cipher number	Short name	Description ¹	FIPS 140-2	Base security level FMID HCPT420	Security level 3 FMID JCPT421
00	0000	TLS_NULL_WITH_NULL_NULL	No encryption or message authentication and RSA key exchange		X	X
01	0001	TLS_RSA_WITH_NULL_MD5	No encryption with MD5 message authentication and RSA key exchange		X	X
02	0002	TLS_RSA_WITH_NULL_SHA	No encryption with SHA-1 message authentication and RSA key exchange		X	X
03	0003	TLS_RSA_EXPORT_WITH_RC4_40_MD5	40-bit RC4 encryption with MD5 message authentication and RSA (export) key exchange		X	X
04	0004	TLS_RSA_WITH_RC4_128_MD5	128-bit RC4 encryption with MD5 message authentication and RSA key exchange			X
05	0005	TLS_RSA_WITH_RC4_128_SHA	128-bit RC4 encryption with SHA-1 message authentication and RSA key exchange			X
06	0006	TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5	40-bit RC2 encryption with MD5 message authentication and RSA (export) key exchange		X	X
09	0009	TLS_RSA_WITH_DES_CBC_SHA	56-bit DES encryption with SHA-1 message authentication and RSA key exchange		X	X
0A	000A	TLS_RSA_WITH_3DES_EDE_CBC_SHA	168-bit Triple DES encryption with SHA-1 message authentication and RSA key exchange	X		X
0C	000C	TLS_DH_DSS_WITH_DES_CBC_SHA	56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate		X	X
0D	000D	TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate	X		X
0F	000F	TLS_DH_RSA_WITH_DES_CBC_SHA	56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate		X	X
10	0010	TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA	168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate	X		X

Cipher suite definitions

Table 19. 2-character and 4-character cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 (continued)

2-character cipher number	4-character cipher number	Short name	Description ¹	FIPS 140-2	Base security level FMID HCPT420	Security level 3 FMID JCPT421
12	0012	TLS_DHE_DSS_WITH_DES_CBC_SHA	56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate		X	X
13	0013	TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate	X		X
15	0015	TLS_DHE_RSA_WITH_DES_CBC_SHA	56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate		X	X
16	0016	TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate	X		X
2F	002F	TLS_RSA_WITH_AES_128_CBC_SHA	128-bit AES encryption with SHA-1 message authentication and RSA key exchange	X		X
30	0030	TLS_DH_DSS_WITH_AES_128_CBC_SHA	128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate	X		X
31	0031	TLS_DH_RSA_WITH_AES_128_CBC_SHA	128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate	X		X
32	0032	TLS_DHE_DSS_WITH_AES_128_CBC_SHA	128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate	X		X
33	0033	TLS_DHE_RSA_WITH_AES_128_CBC_SHA	128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate	X		X
35	0035	TLS_RSA_WITH_AES_256_CBC_SHA	256-bit AES encryption with SHA-1 message authentication and RSA key exchange	X		X
36	0036	TLS_DH_DSS_WITH_AES_256_CBC_SHA	256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate	X		X
37	0037	TLS_DH_RSA_WITH_AES_256_CBC_SHA	256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate	X		X
38	0038	TLS_DHE_DSS_WITH_AES_256_CBC_SHA	256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate	X		X
39	0039	TLS_DHE_RSA_WITH_AES_256_CBC_SHA	256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate	X		X
3B	003B	TLS_RSA_WITH_NULL_SHA256	No encryption with SHA-256 message authentication and RSA key exchange		X	X
3C	003C	TLS_RSA_WITH_AES_128_CBC_SHA256	128-bit AES encryption with SHA-256 message authentication and RSA key exchange	X		X
3D	003D	TLS_RSA_WITH_AES_256_CBC_SHA256	256-bit AES encryption with SHA-256 message authentication and RSA key exchange	X		X
3E	003E	TLS_DH_DSS_WITH_AES_128_CBC_SHA256	128-bit AES encryption with SHA-256 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate	X		X
3F	003F	TLS_DH_RSA_WITH_AES_128_CBC_SHA256	128-bit AES encryption with SHA-256 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate	X		X
40	0040	TLS_DHE_DSS_WITH_AES_128_CBC_SHA256	128-bit AES encryption with SHA-256 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate	X		X
67	0067	TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	128-bit AES encryption with SHA-256 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate	X		X
68	0068	TLS_DH_DSS_WITH_AES_256_CBC_SHA256	256-bit AES encryption with SHA-256 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate	X		X
69	0069	TLS_DH_RSA_WITH_AES_256_CBC_SHA256	256-bit AES encryption with SHA-256 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate	X		X
6A	006A	TLS_DHE_DSS_WITH_AES_256_CBC_SHA256	256-bit AES encryption with SHA-256 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate	X		X

Table 19. 2-character and 4-character cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 (continued)

2-character cipher number	4-character cipher number	Short name	Description ¹	FIPS 140-2	Base security level FMID HCPT420	Security level 3 FMID JCPT421
6B	006B	TLS_DHE_RSA_WITH_AES_256_CBC_SHA256	256-bit AES encryption with SHA-256 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate	X		X
9C	009C	TLS_RSA_WITH_AES_128_GCM_SHA256	128-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and RSA key exchange	X		X
9D	009D	TLS_RSA_WITH_AES_256_GCM_SHA384	256-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and RSA key exchange	X		X
9E	009E	TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	128-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate	X		X
9F	009F	TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	256-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate	X		X
A0	00A0	TLS_DH_RSA_WITH_AES_128_GCM_SHA256	128-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate	X		X
A1	00A1	TLS_DH_RSA_WITH_AES_256_GCM_SHA384	256-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate	X		X
A2	00A2	TLS_DHE_DSS_WITH_AES_128_GCM_SHA256	128-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate	X		X
A3	00A3	TLS_DHE_DSS_WITH_AES_256_GCM_SHA384	256-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate	X		X
A4	00A4	TLS_DH_DSS_WITH_AES_128_GCM_SHA256	128-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate	X		X
A5	00A5	TLS_DH_DSS_WITH_AES_256_GCM_SHA384	256-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate	X		X
	C001	TLS_ECDH_ECDSA_WITH_NULL_SHA	NULL encryption with SHA-1 message authentication and fixed ECDH key exchange signed with an ECDSA certificate		X	X
	C002	TLS_ECDH_ECDSA_WITH_RC4_128_SHA	128-bit RC4 encryption with SHA-1 message authentication and fixed ECDH key exchange signed with an ECDSA certificate			X
	C003	TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA	168-bit Triple DES encryption with SHA-1 message authentication and fixed ECDH key exchange signed with an ECDSA certificate	X		X
	C004	TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA	128-bit AES encryption with SHA-1 message authentication and fixed ECDH key exchange signed with an ECDSA certificate	X		X
	C005	TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA	256-bit AES encryption with SHA-1 message authentication and fixed ECDH key exchange signed with an ECDSA certificate	X		X
	C006	TLS_ECDHE_ECDSA_WITH_NULL_SHA	NULL encryption with SHA-1 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate		X	X
	C007	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	128-bit RC4 encryption with SHA-1 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate			X
	C008	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	168-bit Triple DES encryption with SHA-1 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate	X		X
	C009	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	128-bit AES encryption with SHA-1 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate	X		X
	C00A	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	256-bit AES encryption with SHA-1 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate	X		X

Cipher suite definitions

Table 19. 2-character and 4-character cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 (continued)

2-character cipher number	4-character cipher number	Short name	Description ¹	FIPS 140-2	Base security level FMID HCPT420	Security level 3 FMID JCPT421
	C00B	TLS_ECDH_RSA_WITH_NULL_SHA	NULL encryption with SHA-1 message authentication and fixed ECDH key exchange signed with an RSA certificate		X	X
	C00C	TLS_ECDH_RSA_WITH_RC4_128_SHA	128-bit RC4 encryption with SHA-1 message authentication and fixed ECDH key exchange signed with an RSA certificate			X
	C00D	TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA	168-bit Triple DES encryption with SHA-1 message authentication and fixed ECDH key exchange signed with an RSA certificate	X		X
	C00E	TLS_ECDH_RSA_WITH_AES_128_CBC_SHA	128-bit AES encryption with SHA-1 message authentication and fixed ECDH key exchange signed with an RSA certificate	X		X
	C00F	TLS_ECDH_RSA_WITH_AES_256_CBC_SHA	256-bit AES encryption with SHA-1 message authentication and fixed ECDH key exchange signed with an RSA certificate	X		X
	C010	TLS_ECDHE_RSA_WITH_NULL_SHA	NULL encryption with SHA-1 message authentication and ephemeral ECDH key exchange signed with an RSA certificate		X	X
	C011	TLS_ECDHE_RSA_WITH_RC4_128_SHA	128-bit RC4 encryption with SHA-1 message authentication and ephemeral ECDH key exchange signed with an RSA certificate			X
	C012	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	168-bit Triple DES encryption with SHA-1 message authentication and ephemeral ECDH key exchange signed with an RSA certificate	X		X
	C013	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	128-bit AES encryption with SHA-1 message authentication and ephemeral ECDH key exchange signed with an RSA certificate	X		X
	C014	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	256-bit AES encryption with SHA-1 message authentication and ephemeral ECDH key exchange signed with an RSA certificate	X		X
	C023	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	128-bit AES encryption with SHA-256 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate	X		X
	C024	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	256-bit AES encryption with SHA-384 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate	X		X
	C025	TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256	128-bit AES encryption with SHA-256 message authentication and fixed ECDH key exchange signed with an ECDSA certificate	X		X
	C026	TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384	256-bit AES encryption with SHA-384 message authentication and fixed ECDH key exchange signed with an ECDSA certificate	X		X
	C027	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	128-bit AES encryption with SHA-256 message authentication and ephemeral ECDH key exchange signed with an RSA certificate	X		X
	C028	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	256-bit AES encryption with SHA-384 message authentication and ephemeral ECDH key exchange signed with an RSA certificate	X		X
	C029	TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256	128-bit AES encryption with SHA-256 message authentication and fixed ECDH key exchange signed with an RSA certificate	X		X
	C02A	TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384	256-bit AES encryption with SHA-384 message authentication and fixed ECDH key exchange signed with an RSA certificate	X		X
	C02B	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	128-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and ephemeral ECDH key exchange signed with an ECDSA certificate	X		X
	C02C	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	256-bit AES in Galois Counter Mode encryption with 128-bit AEAD message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate	X		X
	C02D	TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256	128-bit AES in Galois Counter Mode encryption with 128-bit AEAD message authentication and fixed ECDH key exchange signed with an ECDSA certificate	X		X
	C02E	TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384	256-bit AES in Galois Counter Mode encryption with 128-bit AEAD message authentication and fixed ECDH key exchange signed with an ECDSA certificate	X		X

Table 19. 2-character and 4-character cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 (continued)

2-character cipher number	4-character cipher number	Short name	Description ¹	FIPS 140-2	Base security level FMID HCPT420	Security level 3 FMID JCPT421
	C02F	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	128-bit AES in Galois Counter Mode encryption with 128-bit AEAD message authentication and ephemeral ECDH key exchange signed with an RSA certificate	X		X
	C030	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	256-bit AES in Galois Counter Mode encryption with 128-bit AEAD message authentication and ephemeral ECDH key exchange signed with an RSA certificate	X		X
	C031	TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256	128-bit AES in Galois Counter Mode encryption with 128-bit AEAD message authentication and fixed ECDH key exchange signed with an RSA certificate	X		X
	C032	TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384	256-bit AES in Galois Counter Mode encryption with 128-bit AEAD message authentication and fixed ECDH key exchange signed with an RSA certificate	X		X

¹ See Table 21 on page 693 for more information about the signing algorithm required for the key exchanges.

Table 20. Cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 by supported protocol, symmetric algorithm, and message authentication algorithm

Cipher suite		Protocol support				Symmetric algorithm						Message MAC				
4 Char	2 Char	SSL V3	TLS V1.0	TLS V1.1	TLS V1.2	RC2 or RC4	DES or 3DES	AES-CBC 128	AES-CBC 256	AES-GCM 128	AES-GCM 256	MD5	SHA 1	SHA 256	SHA 384	AEAD
0000	00	X	X	X	X											
0001	01	X	X	X	X							X				
0002	02	X	X	X	X								X			
0003	03	X	X			RC4						X				
0004	04	X	X	X	X	RC4						X				
0005	05	X	X	X	X	RC4							X			
0006	06	X	X			RC2						X				
0009	09	X	X	X			DES						X			
000A	0A	X	X	X	X		3DES						X			
000C	0C	X	X	X			DES						X			
000D	0D	X	X	X	X		3DES						X			
000F	0F	X	X	X			DES						X			
0010	10	X	X	X	X		3DES						X			
0012	12	X	X	X			DES						X			
0013	13	X	X	X	X		3DES						X			
0015	15	X	X	X			DES						X			
0016	16	X	X	X	X		3DES						X			
002F	2F	X	X	X	X			X					X			
0030	30	X	X	X	X			X					X			
0031	31	X	X	X	X			X					X			
0032	32	X	X	X	X			X					X			
0033	33	X	X	X	X			X					X			
0035	35	X	X	X	X				X				X			
0036	36	X	X	X	X				X				X			
0037	37	X	X	X	X				X				X			
0038	38	X	X	X	X				X				X			
0039	39	X	X	X	X				X				X			
003B	3B				X									X		

Cipher suite definitions

Table 20. Cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 by supported protocol, symmetric algorithm, and message authentication algorithm (continued)

Cipher suite		Protocol support				Symmetric algorithm						Message MAC				
4 Char	2 Char	SSL V3	TLS V1.0	TLS V1.1	TLS V1.2	RC2 or RC4	DES or 3DES	AES-CBC 128	AES-CBC 256	AES-GCM 128	AES-GCM 256	MD5	SHA 1	SHA 256	SHA 384	AEAD
003C	3C				X			X						X		
003D	3D				X				X					X		
003E	3E				X			X						X		
003F	3F				X			X						X		
0040	40				X			X						X		
0067	67				X			X						X		
0068	68				X				X					X		
0069	69				X				X					X		
006A	6A				X				X					X		
006B	6B				X				X					X		
009C	9C				X					X						X
009D	9D				X						X					X
009E	9E				X					X						X
009F	9F				X						X					X
00A0	A0				X					X						X
00A1	A1				X						X					X
00A2	A2				X					X						X
00A3	A3				X						X					X
00A4	A4				X					X						X
00A5	A5				X						X					X
C001			X	X	X								X			
C002			X	X	X	RC4							X			
C003			X	X	X		3DES						X			
C004			X	X	X			X					X			
C005			X	X	X				X				X			
C006			X	X	X								X			
C007			X	X	X	RC4							X			
C008			X	X	X		3DES						X			
C009			X	X	X			X					X			
C00A			X	X	X				X				X			
C00B			X	X	X								X			
C00C			X	X	X	RC4							X			
C00D			X	X	X		3DES						X			
C00E			X	X	X			X					X			
C00F			X	X	X				X				X			
C010			X	X	X								X			
C011			X	X	X	RC4							X			
C012			X	X	X		3DES						X			
C013			X	X	X			X					X			
C014			X	X	X				X				X			
C023					X			X						X		
C024					X				X						X	
C025					X			X						X		
C026					X				X						X	
C027					X			X						X		
C028					X				X						X	
C029					X			X						X		

Cipher suite definitions

Table 20. Cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 by supported protocol, symmetric algorithm, and message authentication algorithm (continued)

Cipher suite		Protocol support				Symmetric algorithm						Message MAC				
4 Char	2 Char	SSL V3	TLS V1.0	TLS V1.1	TLS V1.2	RC2 or RC4	DES or 3DES	AES-CBC 128	AES-CBC 256	AES-GCM 128	AES-GCM 256	MD5	SHA 1	SHA 256	SHA 384	AEAD
C02A					X				X						X	
C02B					X					X						X
C02C					X						X					X
C02D					X					X						X
C02E					X						X					X
C02F					X					X						X
C030					X						X					X
C031					X					X						X
C032					X						X					X

Table 21. Cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 by key-exchange method and signing certificate

Cipher suite		RSA key exchange	Fixed Diffie-Hellman key exchange		Ephemeral Diffie-Hellman key exchange		Fixed EC Diffie-Hellman key exchange		Ephemeral EC Diffie-Hellman key exchange	
4 Char	2 Char		Signed by RSA ¹	Signed by DSA ¹	Signed by RSA ¹	Signed by DSA ¹	Signed by RSA ¹	Signed by ECDSA ¹	Signed by RSA ¹	Signed by ECDSA ¹
0000	00	X								
0001	01	X								
0002	02	X								
0003	03	X								
0004	04	X								
0005	05	X								
0006	06	X								
0009	09	X								
000A	0A	X								
000C	0C			X						
000D	0D			X						
000F	0F		X							
0010	10		X							
0012	12					X				
0013	13					X				
0015	15				X					
0016	16				X					
002F	2F	X								
0030	30			X						
0031	31		X							
0032	32					X				
0033	33				X					
0035	35	X								
0036	36			X						
0037	37		X							
0038	38					X				
0039	39				X					
003B	3B	X								
003C	3C	X								
003D	3D	X								
003E	3E			X						

Cipher suite definitions

Table 21. Cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 by key-exchange method and signing certificate (continued)

Cipher suite		RSA key exchange	Fixed Diffie-Hellman key exchange		Ephemeral Diffie-Hellman key exchange		Fixed EC Diffie-Hellman key exchange		Ephemeral EC Diffie-Hellman key exchange	
4 Char	2 Char		Signed by RSA ¹	Signed by DSA ¹	Signed by RSA ¹	Signed by DSA ¹	Signed by RSA ¹	Signed by ECDSA ¹	Signed by RSA ¹	Signed by ECDSA ¹
003F	3F		X							
0040	40					X				
0067	67				X					
0068	68			X						
0069	69		X							
006A	6A					X				
006B	6B				X					
009C	9C	X								
009D	9D	X								
009E	9E				X					
009F	9F				X					
00A0	A0		X							
00A1	A1		X							
00A2	A2					X				
00A3	A3					X				
00A4	A4			X						
00A5	A5			X						
C001							X			
C002							X			
C003							X			
C004							X			
C005							X			
C006										X
C007										X
C008										X
C009										X
C00A										X
C00B							X			
C00C							X			
C00D							X			
C00E							X			
C00F							X			
C010									X	
C011									X	
C012									X	
C013									X	
C014									X	
C023										X
C024										X
C025							X			
C026							X			
C027									X	
C028									X	
C029							X			
C02A							X			
C02B										X

Table 21. Cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 by key-exchange method and signing certificate (continued)

Cipher suite		RSA key exchange	Fixed Diffie-Hellman key exchange		Ephemeral Diffie-Hellman key exchange		Fixed EC Diffie-Hellman key exchange		Ephemeral EC Diffie-Hellman key exchange	
			Signed by RSA ¹	Signed by DSA ¹	Signed by RSA ¹	Signed by DSA ¹	Signed by RSA ¹	Signed by ECDSA ¹	Signed by RSA ¹	Signed by ECDSA ¹
4 Char	2 Char									
C02C										X
C02D								X		
C02E								X		
C02F									X	
C030									X	
C031							X			
C032							X			

¹ SSL V3, TLS V1.0, and TLS V1.1 imposed restrictions on the signing algorithm that must be used to sign a server certificate when using any cipher suites that use a Diffie-Hellman based key-exchange. The TLS V1.2 protocol does not impose such restriction. If the server certificate signing algorithm is listed in the signature algorithm pairs that are specified by the client, the certificate can be used.

Table 22. Supported elliptic curve definitions for TLS V1.0, TLS V1.1, and TLS V1.2

I.A.N.A Elliptic curve enumerator (decimal)	Named curve by standards organizations		
	SECG	ANSI X9.62	NIST
0019	secp192r1	prime192v1	NIST P-192
0021	secp224r1		NIST P-224
0023	secp256r1	prime256v1	NIST P-256
0024	secp384r1		NIST P-384
0025	secp521r1		NIST P-521

Table 23. Signature algorithm pair definitions for TLS V1.2 and OCSP request signing

Signature algorithm enumerator	Hash and signature algorithm
0101	MD5 with RSA
0201	SHA-1 with RSA
0202	SHA-1 with DSA
0203	SHA-1 with ECDSA
0301	SHA-224 with RSA
0302	SHA-224 with DSA
0303	SHA-224 with ECDSA
0401	SHA-256 with RSA
0402	SHA-256 with DSA
0403	SHA-256 with ECDSA
0501	SHA-384 with RSA
0503	SHA-384 with ECDSA
0601	SHA-512 with RSA
0603	SHA-512 with ECDSA

Cipher suite definitions

Appendix D. Object identifiers

The following table shows the object identifiers (OIDs) supported by System SSL.

Table 24. System SSL supported object identifiers (OIDs)

Type	Description	OID
Digest Algorithms	MD2	1.2.840.113549.2.2
	MD5	1.2.840.113549.2.5
	SHA-1	1.3.14.3.2.26
	SHA-224	2.16.840.1.101.3.4.2.4
	SHA-256	2.16.840.1.101.3.4.2.1
	SHA-394	2.16.840.1.101.3.4.2.2
	SHA-512	2.16.840.1.101.3.4.2.3
Asymmetric Encryption Algorithms	RSA Encryption	1.2.840.113549.1.1.1
	DSA	1.2.840.10040.4.1
	Diffie-Hellman (dhPublicNumber)	1.2.840.10046.2.1
	ECC (ecPublicKey)	1.2.840.10045.2.1
Signature Algorithms	md2WithRsaEncryption	1.2.840.113549.1.1.2
	md5WithRsaEncryption	1.2.840.113549.1.1.4
	sha1WithRsaEncryption	1.2.840.113549.1.1.5
	sha224WithRsaEncryption	1.2.840.113549.1.1.14
	sha256WithRsaEncryption	1.2.840.113549.1.1.11
	sha384WithRsaEncryption	1.2.840.113549.1.1.12
	sha512WithRsaEncryption	1.2.840.113549.1.1.13
	dsaWithSha1	1.2.840.10040.4.3
	dsaWithSha224	2.16.840.1.101.3.4.3.1
	dsaWithSha256	2.16.840.1.101.3.4.3.2
	ecdsaWithSha1	1.2.840.10045.4.1
	ecdsaWithSha224	1.2.840.10045.4.3.1
	ecdsaWithSha256	1.2.840.10045.4.3.2
	ecdsaWithSha384	1.2.840.10045.4.3.3
ecdsaWithSha512	1.2.840.10045.4.3.4	
Password Base Encryption Algorithms	pbeWithMd2AndDesCbc	1.2.840.113549.1.5.1
	pbeWithMd5AndDesCbc	1.2.840.113549.1.5.3
	pbeWithSha1AndDesCbc	1.2.840.113549.1.5.10
	pbeWithMd2AndRc2Cbc	1.2.840.113549.1.5.4
	pbeWithMd5AndRc2Cbc	1.2.840.113549.1.5.6
	pbeWithSha1AndRc2Cbc	1.2.840.113549.1.5.11
	pbeWithSha1And40BitRc2Cbc	1.2.840.113549.1.12.1.6
	pbeWithSha1And128BitRc2Cbc	1.2.840.113549.1.12.1.5
	pbeWithSha1And40BitRc4	1.2.840.113549.1.12.1.2
	pbeWithSha1And128BitRc4	1.2.840.113549.1.12.1.1
	pbeWithSha1And3DesCbc	1.2.840.113549.1.12.1.3
	Deprecated Password Based Encryption Algorithms	pb1WithSha1And128BitRc4
pb1WithSha1And40BitRc4		1.2.840.113549.1.12.5.1.2
pb1WithSha1And3DesCbc		1.2.840.113549.1.12.5.1.3
pb1WithSha1And128BitRc2Cbc		1.2.840.113549.1.12.5.1.4
pb1WithSha1And40BitRc2Cbc		1.2.840.113549.1.12.5.1.5
Symmetric Encryption Algorithms	DES CBC	1.3.14.3.2.7
	3DES CBC	1.2.840.113549.3.7
	RC2	1.2.840.113549.3.2
	ArcFour	1.2.840.113549.3.4
	AES CBC 128	2.16.840.1.101.3.4.1.2
	AES CBC 256	2.16.840.1.101.3.4.1.42

Object identifiers

Table 24. System SSL supported object identifiers (OIDs) (continued)

Type	Description	OID
x.500 Distinguished Name Attributes	name	2.5.4.41
	surname	2.5.4.4
	given name	2.5.4.42
	initials	2.5.4.43
	generation qualifier	2.5.4.44
	common name	2.5.4.3
	locality name	2.5.4.7
	state or province name	2.5.4.8
	organization name	2.5.4.10
	organizational unit name	2.5.4.11
	title	2.5.4.12
	dnQualifier	2.5.4.46
	country name	2.5.4.6
	email address	1.2.840.113549.1.9.1
	domain component	0.9.2342.19200300.100.1.25
	street address	2.5.4.9
	postal code	2.5.4.17
mail	0.9.2342.19200300.100.1.3	
serial number	2.5.4.5	
ECC Name Curves	secp192r1	1.2.840.10045.3.1.1
	secp224r1	1.3.132.0.33
	secp256r1	1.2.840.10045.3.1.7
	secp384r1	1.3.132.0.34
	secp521r1	1.3.132.0.35
	brainpoolP160r1	1.3.36.3.3.2.8.1.1.1
	brainpoolP192r1	1.3.36.3.3.2.8.1.1.3
	brainpoolP224r1	1.3.36.3.3.2.8.1.1.5
	brainpoolP256r1	1.3.36.3.3.2.8.1.1.7
	brainpoolP320r1	1.3.36.3.3.2.8.1.1.9
	brainpoolP384r1	1.3.36.3.3.2.8.1.1.11
	brainpoolP512r1	1.3.36.3.3.2.8.1.1.13

Appendix E. Accessibility

Accessible publications for this product are offered through IBM Knowledge Center (<http://www.ibm.com/support/knowledgecenter/SSLTBW/welcome>).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the "Contact us" web page for z/OS (<http://www.ibm.com/systems/z/os/zos/webqs.html>) or use the following mailing address.

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

Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS V2R2 ISPF User's Guide Vol I*

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM Knowledge Center with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that the screen reader is set to read out

punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, your syntax can include either USERID or SYSTEMID, but not both.

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

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

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

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

The following symbols are used next to the dotted decimal numbers.

? indicates an optional syntax element

The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark symbol (?) indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that the syntax elements NOTIFY and UPDATE are optional. That is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

! indicates a default syntax element

The exclamation mark (!) symbol indicates a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the dotted decimal number can specify the ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the

default option for the FILE keyword. In the example, if you include the FILE keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, the default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP applies only to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

*** indicates an optional syntax element that is repeatable**

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

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write HOST STATE, but you cannot write HOST HOST.
3. The * symbol is equivalent to a loopback line in a railroad syntax diagram.

+ indicates a syntax element that must be included

The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loopback line in a railroad syntax diagram.

Notices

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

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

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

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

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel
IBM Corporation
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

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

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

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

COPYRIGHT LICENSE:

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

Policy for unsupported hardware

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

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

Minimum supported hardware

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

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

Trademarks

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

Index

A

- accepting a secure socket connection 497
- accessibility 699
 - contact IBM 699
 - features 699
- accessing DLLs 522
- APIs
 - `gsk_attribute_get_buffer()` 60
 - `gsk_attribute_get_cert_info()` 65
 - `gsk_attribute_get_data()` 70
 - `gsk_attribute_get_enum()` 72
 - `gsk_attribute_get_numeric_value()` 79
 - `gsk_attribute_set_buffer()` 82
 - `gsk_attribute_set_callback()` 87
 - `gsk_attribute_set_enum()` 92
 - `gsk_attribute_set_numeric_value()` 102
 - `gsk_attribute_set_tls_extension()` 108
 - `gsk_construct_private_key()` 171
 - `gsk_construct_public_key()` 175
 - `gsk_environment_close()` 111
 - `gsk_environment_init()` 112
 - `gsk_environment_open()` 114
 - `gsk_factor_private_key()` 287
 - `gsk_factor_public_key()` 289
 - `gsk_free_cert_data()` 121
 - `gsk_free_memory()` 488
 - `gsk_free_private_key()` 307
 - `gsk_free_public_key()` 309
 - `gsk_get_all_cipher_suites()` 122
 - `gsk_get_cert_by_label()` 123
 - `gsk_get_cipher_info` 489
 - `gsk_get_dn_by_label()` 490
 - `gsk_get_ec_parameters_info()` 342
 - `gsk_get_update()` 130
 - `gsk_initialize()` 491
 - `gsk_list_free()` 131
 - `gsk_secure_soc_close()` 496
 - `gsk_secure_soc_init()` 497
 - `gsk_secure_soc_read()` 505
 - `gsk_secure_soc_reset()` 508
 - `gsk_secure_soc_write()` 509
 - `gsk_secure_socket_close()` 132
 - `gsk_secure_socket_init()` 133
 - `gsk_secure_socket_misc()` 142
 - `gsk_secure_socket_open()` 144
 - `gsk_secure_socket_read()` 145
 - `gsk_secure_socket_write()` 150
 - `gsk_uninitialize()` 515
 - `gsk_user_set()` 516
 - `GSKSRBRD()` 513
 - `GSKSRBWT()` 514, 577
- using in an System SSL program 6
- assistive technologies 699

B

- building a z/OS System SSL application 29
- building an System SSL application 34

C

- callback routine for I/O 39
- certificate
 - removing 562
 - self-signed, creating 547
- certificate management
 - introduction 519
- Certificate Management Services (CMS) API reference 153
- Certificate/Key management 519
- cipher information
 - querying 489
- Cipher suite definitions 687
- client, authentication certificate selection 38
- compiling an System SSL application 34
- component trace support 590
- Configuring the SSL started task 588
- contact
 - z/OS 699
- creating
 - SSL environment 29

D

- diagnostic information 591
- Diffie-Hellman key agreement 16
- distinguished name
 - returning pointer for 490
- DLLs, accessing 522

E

- elements of an System SSL program 6
- Elliptic Curve cryptography support 14
- ending secure socket connection 496
- environment variables 667
- establishing System SSL environment 491
- examples
 - parts shipped in UNIX system services file system 2

F

- FIPS 140-2 19
- FIPS mode
 - algorithms and key sizes 19
 - application changes 26
 - certificate stores 25
 - certificates 21
 - SAF key rings and PKCS #11 tokens 25
 - SSL started task 26
 - SSL/TLS protocol 21
 - system setup and requirements 22
- FMID
 - Cryptographic Services Security Level 3 1
 - Cryptographic Services System SSL 1
 - Japanese 1

G

`gsk_add_record()` 159
`gsk_attribute_get_buffer()` 60
`gsk_attribute_get_cert_info()` 65
`gsk_attribute_get_data()` 70
`gsk_attribute_get_enum()` 72
`gsk_attribute_get_numeric_value()` 79
`gsk_attribute_set_buffer()` 82
`gsk_attribute_set_callback()` 87
`gsk_attribute_set_enum()` 92
`gsk_attribute_set_numeric_value()` 102
`gsk_attribute_set_tls_extension()` 108
`gsk_change_database_password()` 162
`gsk_change_database_record_length()` 164
`gsk_close_database()` 165
`gsk_close_directory()` 166
`gsk_construct_certificate()` 167
`gsk_construct_private_key_rsa()` 173
`gsk_construct_private_key()` 171
`gsk_construct_public_key_rsa()` 177
`gsk_construct_public_key()` 175
`gsk_construct_renewal_request()` 178
`gsk_construct_self_signed_certificate()` 181
`gsk_construct_signed_certificate()` 184
`gsk_copy_attributes_signers()` 188
`gsk_copy_buffer()` 189
`gsk_copy_certificate_extension()` 191
`gsk_copy_certificate()` 190
`gsk_copy_certification_request()` 192
`gsk_copy_content_info()` 193
`gsk_copy_crl()` 194
`gsk_copy_name()` 195
`gsk_copy_private_key_info()` 196
`gsk_copy_public_key_info()` 197
`gsk_copy_record()` 198
`gsk_create_certification_request()` 199
`gsk_create_database_renewal_request()` 205
`gsk_create_database_signed_certificate()` 208
`gsk_create_database()` 203
`gsk_create_renewal_request()` 214
`gsk_create_revocation_source()` 216, 313
`gsk_create_self_signed_certificate()` 222
`gsk_create_signed_certificate_record()` 229
`gsk_create_signed_certificate_set()` 234
`gsk_create_signed_certificate()` 226
`gsk_create_signed_crl_record()` 242
`gsk_create_signed_crl()` 239
`gsk_decode_base64()` 246
`gsk_decode_certificate_extension()` 248
`gsk_decode_certificate()` 247
`gsk_decode_certification_request()` 250
`gsk_decode_crl()` 251
`gsk_decode_import_certificate()` 252
`gsk_decode_import_key()` 253
`gsk_decode_issuer_and_serial_number()` 255
`gsk_decode_name()` 256
`gsk_decode_private_key()` 257
`gsk_decode_public_key()` 258
`gsk_decode_signer_identifiers()` 259
`gsk_delete_record()` 260
`gsk_dn_to_name()` 261
`gsk_encode_base64()` 264
`gsk_encode_certificate_extension()` 265
`gsk_encode_ec_parameters()` 267
`gsk_encode_export_certificate()` 268
`gsk_encode_export_key()` 270
`gsk_encode_export_request()` 273
`gsk_encode_issuer_and_serial_number()` 274
`gsk_encode_name()` 275
`gsk_encode_private_key()` 276
`gsk_encode_public_key()` 277
`gsk_encode_signature()` 278
`gsk_encode_signer_identifiers()` 279
`gsk_environment_close()` 111
`gsk_environment_init()` 112
`gsk_environment_open()` 114
`gsk_export_certificate()` 280
`gsk_export_certification_request()` 282
`gsk_export_key()` 284
`gsk_factor_private_key_rsa()` 288
`gsk_factor_private_key()` 287
`gsk_factor_public_key_rsa()` 290
`gsk_factor_public_key()` 289
`gsk_fips_state_query()` 291
`gsk_fips_state_set()` 292
`gsk_free_attributes_signers()` 294
`gsk_free_buffer()` 295
`gsk_free_cert_data()` 121
`gsk_free_certificate_extension()` 298
`gsk_free_certificate()` 296
`gsk_free_certificates()` 297
`gsk_free_certification_request()` 299
`gsk_free_content_info()` 300
`gsk_free_crl()` 301
`gsk_free_crls()` 302
`gsk_free_decoded_extension()` 303
`gsk_free_issuer_and_serial_number()` 304
`gsk_free_memory()` API 488
`gsk_free_name()` 305
`gsk_free_oid()` 306
`gsk_free_private_key_info()` 308
`gsk_free_private_key()` 307
`gsk_free_public_key_info()` 310
`gsk_free_public_key()` 309
`gsk_free_record()` 311
`gsk_free_records()` 312
`gsk_free_signer_identifiers()` 314
`gsk_free_string()` 315
`gsk_free_strings()` 316
`gsk_generate_key_agreement_pair()` 317
`gsk_generate_key_pair()` 319
`gsk_generate_key_parameters()` 322
`gsk_generate_random_bytes()` 324
`gsk_generate_secret()` 325
`gsk_get_all_cipher_suites()` 122
`gsk_get_cert_by_label()` 123
`gsk_get_certificate_algorithms()` 326
`gsk_get_certificate_info()` 327
`gsk_get_cipher_info()` API 489
`gsk_get_cms_vector()` 329
`gsk_get_content_type_and_cms_version()` 331
`gsk_get_default_key()` 332
`gsk_get_default_label()` 333
`gsk_get_directory_certificates()` 334
`gsk_get_directory_crls()` 336
`gsk_get_directory_enum()` 339
`gsk_get_directory_numeric_value()` 341
`gsk_get_dn_by_label()` API 490
`gsk_get_ec_parameters_info()` 342
`gsk_get_record_by_id()` 343
`gsk_get_record_by_index()` 344
`gsk_get_record_by_label()` 345
`gsk_get_record_by_subject()` 346
`gsk_get_record_labels()` 347

gsk_get_update_code() 348
gsk_get_update() 130
gsk_import_certificate() 349
gsk_import_key() 352
gsk_initialize() API 491
 Environment variables supported 494
gsk_list_free() 131
gsk_make_content_msg() 355
gsk_make_data_content() 356
gsk_make_data_msg() 357
gsk_make_encrypted_data_content() 358
gsk_make_encrypted_data_msg() 360
gsk_make_enveloped_data_content_extended() 365
gsk_make_enveloped_data_content() 362
gsk_make_enveloped_data_msg_extended() 371
gsk_make_enveloped_data_msg() 368
gsk_make_signed_data_content_extended() 380
gsk_make_signed_data_content() 377
gsk_make_signed_data_msg_extended() 387
gsk_make_signed_data_msg() 384
gsk_make_wrapped_content() 391
gsk_mktime() 392
gsk_name_compare() 395
gsk_name_to_dn() 396
gsk_open_database_using_stash_file() 400
gsk_open_database() 398
gsk_open_directory() 402
gsk_open_keyring() 403
gsk_perform_kat() 405
gsk_query_crypto_level() 406
gsk_query_database_label() 407
gsk_query_database_record_length() 408
gsk_rdttime() 409
gsk_read_content_msg() 410
gsk_read_data_content() 411
gsk_read_data_msg() 412
gsk_read_encrypted_data_content() 413
gsk_read_encrypted_data_msg() 415
gsk_read_enveloped_data_content_extended() 419
gsk_read_enveloped_data_content() 417
gsk_read_enveloped_data_msg_extended() 423
gsk_read_enveloped_data_msg() 421
gsk_read_signed_data_content_extended() 428
gsk_read_signed_data_content() 425
gsk_read_signed_data_msg_extended() 434
gsk_read_signed_data_msg() 431
gsk_read_wrapped_content() 438
gsk_receive_certificate() 439
gsk_replace_record() 440
gsk_secure_soc_close API 496
gsk_secure_soc_init() API 497
gsk_secure_soc_read() API 505
gsk_secure_soc_reset() API 508
gsk_secure_soc_write() API 509
gsk_secure_socket_close() 132
gsk_secure_socket_init() 133
gsk_secure_socket_misc() 142
gsk_secure_socket_open() 144
gsk_secure_socket_read() 145
gsk_secure_socket_write() 150
gsk_set_default_key() 443
gsk_set_directory_enum() 445
gsk_set_directory_numeric_value() 447
gsk_sign_certificate() 449
gsk_sign_crl() 452
gsk_sign_data() 455
gsk_soc_init_data structure 497
gsk_uninitialize() API 515
gsk_user_set() API 516
gsk_validate_certificate_mode() 464
gsk_validate_certificate() 458
gsk_validate_extended_key_usage() 472
gsk_validate_hostname() 474
gsk_validate_server() 476
gsk_verify_certificate_signature() 477
gsk_verify_crl_signature() 480
gsk_verify_data_signature() 483
gskkyman utility
 accessing DLLs 522
 being your own certificate authority 574
 certificate, self signed
 creating 547
 certificates
 removing 562
 database menu 524
 key database files 523
 key management menu 527
 overview 521
 private key
 removing 562
 setting LANG environment variable 522
 setting NLSPATH environment variable 522
 setting PATH environment variable 522
 setting STEPLIB environment variable 522
 setting up the environment 522
 token management menu 527
 UNIX system services file system location 2
 using 519
 z/OS PKCS #11 tokens 523
GSKSRBRD() 513
GSKSRBWT() 514, 577
GSKSRVR environment variables 587
gskssl.h header file
 gsk_soc_init_data structure 497

H

handshake process 497
hardware cryptographic features and System SSL 12
hardware cryptography failure notification 590
header file, gskssl.h 2
HTTP CDP support 48

I

initializing data areas for System SSL 497
initiating a secure socket connection 497
installation information 2
installation PDS and PDSE
 members of 2
 name of 1

K

key database file
 reading 491
 uninitialize 515
key management 519
key ring 519
keyboard
 navigation 699
 PF keys 699
 shortcut keys 699

L

LANG environment variable, setting 522
LDAP CRL support 50

M

managing PKI private keys and certificates 519
Messages and codes 599
 ASN.1 status codes (014CExxx) 629
 CMS status codes (03353xxx) 633
 Deprecated SSL function return codes 618
 SSL function return codes 599
 SSL started task messages (GSK01nnn) 656
 Utility messages (GSK00nnn) 665
migrating from deprecated SSL interfaces 55

N

navigation
 keyboard 699
NLSPATH environment variable, setting 522
Notices 703

O

object identifiers 697
obtaining System SSL trace information 591
OCSP support 46

P

PATH environment variable, setting 522
PDS
 identified in STEPLIB 35
PDS and PDSE, installation
 members of 2
 name of 1
PKCS #11 and setting CLEARKEY resource within CRYPTOZ
 class 18
PKCS #11 Cryptographic operations using ICSF handles 18
PKCS #12 files 25
private keys
 removing 562
programming interfaces
 using in an System SSL program 6

Q

querying cipher information 489

R

RACDCERT command 519
RACF CSFSERV resource requirements 16
RACF key ring
 reading 491
 uninitialize 515
Random byte generation support 14
receiving data on secure socket connection 505
refreshing security parameters 508
removing
 certificate/private key from key database 562
removing settings for the System SSL environment 515
returning distinguished name 490

running an System SSL application 35

S

SAF
 access levels 524
sample files
 list of 685
secure socket connection
 accepting 497
 ending 496
 initiating 497
 receiving data 505
 sending data 509
Secure Sockets Layer (SSL) 1
sending comments to IBM xv
sending data on secure socket connection 509
Server operator commands 589
server, System SSL program 31
session ID (SID) 40
session ID cache replacement 41
session renegotiation notification 42
setting
 gskkyman environment 522
 LANG environment variable 522
 NLSPATH environment variable 522
 PATH environment variable 522
 STEPLIB environment variable 522
shortcut keys 699
software dependencies 1
SSL (Secure Sockets Layer)
 description 1
SSL environment
 creating 29
SSL started task 587
SSL System
 callback routine for I/O 39
SSL/TLS partner certificate revocation checking 46
STEPLIB environment variable, setting 522
structure
 gsk_soc_init_data 497
structure of a System SSL program 6
summary of changes xvii
Summary of changes xx
Sysplex session cache support 590
System SSL
 APIs 57
 client authentication certificate selection 38
 elements of a program 6
 environment variables 667
 establishing environment 491
 FIPS 140-2 19
 how it works 5
 migrating 55
 object identifiers 697
 obtaining trace information 591
 parts shipped in PDS and PDSE 2
 parts shipped in UNIX file system 2
 refreshing security parameters 508
 removing settings for the environment 515
 session ID (SID) cache 40
 using hardware cryptographic features 11
System SSL application
 building 34
 overview 6
 writing a server program 31
 writing a source program 29

System SSL application (*continued*)
 writing and building 29
System SSL application programming considerations 35
System SSL client program 33

T

trademarks 705

U

UNIX file system
 parts shipped 2
user interface
 ISPF 699
 TSO/E 699
using hardware cryptographic features with System SSL 11

W

writing
 System SSL server program 31
 System SSL source program 29
 z/OS System SSL application 29

X

x.509 certificate revocation 520
 enabling HTTP CDP support checking 48
 enabling LDAP CRL support 50
 enabling OCSP support checking 46
 examples 52
 security enforcement 51
 source checking 51
 SSL/TLS partner certificate revocation checking 46



Product Number: 5650-ZOS

Printed in USA

SC14-7495-01

