z/OS

APAR OA26457
System Secure Sockets Layer Programming

August 25, 2009

z/OS

APAR OA26457
System Secure Sockets Layer Programming

August 25, 2009

Contents

Summary of changes

Summary of Changes for APAR OA26457

This book contains information for APAR OA26457. It is based on information previously presented in *z/OS System Secure Sockets Layer Programming, SC24-5901-07*, which supports z/OS Version 1 Release 10.

New information

- Information in Chapter 2, “System SSL and FIPS 140-2,” on page 3
- Added Certificate Management Services (CMS) APIs:
  - gsk_fips_state_query
  - gsk_fips_state_set
  - gsk_perform_kat
- Added Function Return Codes:
  - 445
- Added Deprecated Function Return Code:
  - -105
- Added CMS Status Codes:
  - 03353067
  - 03353068
  - 03353069
  - 0335306A
  - 0335306B
  - 0335306C
  - 03353074
- Added Messages:
  - GSK01053E
  - GSK01054E
  - GSK01057I

Changed information

- Changed API Reference
  - gsk_attribute_get_data
  - gsk_attribute_set_enum
  - gsk_environment_open
  - gsk_get_cipher_suites
  - gsk_secure_socket_init
- Changed Certificate Management Services (CMS) APIs:
  - gsk_add_record
  - gsk_change_database_password
  - gsk_change_database_record_length
  - gsk_construct_certificate
  - gsk_construct_renewal_request
  - gsk_construct_self_signed_certificate
  - gsk_construct_signed_certificate

- gsk_create_certification_request
- gsk_create_database
- gsk_create_database_renewal_request
- gsk_create_database_signed_certificate
- gsk_create_renewal_request
- gsk_create_self_signed_certificate
- gsk_create_signed_certificate_record
- gsk_create_signed_certificate_set
- gsk_create_signed_crl_record
- gsk_decode_import_key
- gsk_delete_record
- gsk_encode_export_key
- gsk_export_key
- gsk_generate_key_agreement_pair
- gsk_generate_key_pair
- gsk_generate_key_parameters
- gsk_generate_secret
- gsk_import_certificate
- gsk_import_key
- gsk_make_encrypted_data_content
- gsk_make_encrypted_data_msg
- gsk_make_enveloped_data_content
- gsk_make_enveloped_data_content_extended
- gsk_make_enveloped_data_msg
- gsk_make_enveloped_data_msg_extended
- gsk_make_signed_data_content
- gsk_make_signed_data_content_extended
- gsk_make_signed_data_msg
- gsk_make_signed_data_msg_extended
- gsk_open_database
- gsk_open_database_using_stash_file
- gsk_read_encrypted_data_content
- gsk_read_encrypted_data_msg
- gsk_read_enveloped_data_content
- gsk_read_enveloped_data_content_extended
- gsk_read_enveloped_data_msg
- gsk_read_enveloped_data_msg_extended
- gsk_read_signed_data_content
- gsk_read_signed_data_content_extended
- gsk_read_signed_data_msg
- gsk_read_signed_data_msg_extended
- gsk_replace_record
- gsk_set_default_key
- gsk_sign_certificate
- gsk_sign_crl

  - gsk_sign_data
  - gsk_validate_certificate
  - gsk_verify_certificate_signature
  - gsk_verify_crl_signature
  - gsk_verify_data_signature
- Changed Deprecated Secure Socket Layer APIs:
  - gsk_get_cipher_info
  - gsk_initialize
  - gsk_secure_soc_init
  - gsk_srb_initialize
- Changed Function Return Codes:
  - 8
  - 9
  - 402
  - 407
  - 412
  - 428
- Changed Deprecated Function Return Codes:
  - -27
  - -35
  - -36
- Changed CMS Status Codes:
  - 03353003
  - 03353010
  - 03353034
- Updated gskyyman
- Changed GSK_PROTOCOL_SSLV2 in Appendix A. Environment Variables
- Changed GSK_PROTOCOL_SSLV3 in Appendix A. Environment Variables
- Changed GSK_V3_CIPHER_SPECS in Appendix A. Environment Variables

Chapter 1. Overview

This book update contains alterations to information previously presented in z/OS System Secure Sockets Layer Programming, SC24-5901-07, which supports z/OS Version 1 Release 10.

Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

These updates relate to the enhancements made to the System SSL product by the application of APAR OA26457, which provides a version of System SSL that is designed to meet the Federal Processing Standard - FIPS 140-2 Level 1 criteria.

Software Dependencies

The System SSL enhancements have the following software dependencies:

- z/OS Version 1 Release 10 Security Server (RACF) - HCR7750 - OA26109 z/OS Version 1 Release 10
- Cryptographic Services (System SSL Security Level 3) - JCPT3A1

Your External Security Manager must support the R_PgmSignVer callable service. RACF supports this callable service through APAR OA26109.

Note: The System SSL DLLs cannot be added to the LPA if System SSL is to be utilized in FIPS mode.

Chapter 2. System SSL and FIPS 140-2

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

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

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

Algorithms and key sizes

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

Table 1 summarizes the differences between FIPS mode and non-FIPS mode algorithm support. Hardware availability depends on the processor and CPACF feature installed. See *z/OS Cryptographic Services System SSL Programming*, SC24-5901 for more information about processors, CPACF algorithm availability and cryptographic card support.

Table 1. Algorithm support: FIPS and non-FIPS

	Non-FIPS		FIPS	
Algorithm	**Sizes**	**Hardware**	**Sizes**	**Hardware**
RC2	40 and 128			
RC4	40 and 128			
DES	56	X		
TDES	168	X	168	X
AES	128 and 256	X	128 and 256	X
MD5	48			
SHA-1	160	X	160	X
SHA-2	224, 256, 384 and 512	X	224, 256, 384 and 512	X
RSA	512–4096	X	1024–4096	X
DSA	512–1024		1024	
DH	512–2048		2048	

Certificates

When executing in FIPS mode, System SSL can only use certificates that use the algorithms and key sizes shown in Table 1. During X.509 certificate validation (including CA certificates from untrusted data sources, that is, certificates flowing during the SSL/TLS handshake), if an algorithm that is incompatible with FIPS mode is encountered, then the certificate cannot be used and is treated as not valid.

SSL/TLS Protocol

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

050435363738392F303132330A1613100D0915120F0C0306020100

The algorithm restrictions (see Table 1 on page 3) result in the following default cipher specifications string in FIPS mode:

35363738392F303132330A1613100D

All cipher suites other than those in the preceding string are incompatible with the restrictions in Table 1 on page 3 and are therefore not supported while executing in FIPS mode.

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

For more information about ciphers and their two-character values, see “gsk_environment_open()” on page 15.

System SSL Module Verification Setup

System SSL requires Security Level 3 FMID (JCPT3A1) and APAR OA26457 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 78.

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

These steps involve:

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

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

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

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

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

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

```
RACDCERT CERTAUTH LIST(LABEL('STG Code Signing CA'))
RACDCERT CERTAUTH ALTER (LABEL('STG Code Signing CA')) TRUST
```

2. Create a key ring to hold the STG Code Signing CA certificate and connect the certificate to the key ring.

   The key ring needs to be owned by a valid RACF id and the key ring must be defined in uppercase. It is recommended that the id be an id of a security administrator. In our example the security administrator id is RACFADM.

   There can only be one designated signature verification key ring active at one time. If already active, add the CA certificate to the key ring. If not already active create the key ring. The recommended key ring name is CODE.SIGNATURE.VERIFICATION.KEYRING.

   - Determine if signature verification key ring already active:

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

     The key ring will be present in the APPLICATION DATA field

   - Create key ring if needed and connect CA certificate:

     ```
     RACDCERT ID(RACFADM) ADDRING(CODE.SIGNATURE.VERIFICATION.KEYRING)
     RACDCERT ID(RACFADM) CONNECT(RING(CODE.SIGNATURE.VERIFICATION.KEYRING) CERTAUTH
     LABEL('STG Code Signing CA') USAGE(CERTAUTH))
     ```

   - If a key ring exists, verify the CA certificate is connected to the key ring. If not connected, connect the certificate:

     ```
     RACDCERT ID(RACFADM) LISTRING(CODE.SIGNATURE.VERIFICATION.KEYRING)
     RACDCERT ID(RACFADM) CONNECT(RING(CODE.SIGNATURE.VERIFICATION.KEYRING) CERTAUTH
     LABEL('STG Code Signing CA') USAGE(CERTAUTH))
     ```

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

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

   ```
   RLIST FACILITY IRR.PROGRAM.SIGNATURE.VERIFICATION
   RDEFINE FACILITY IRR.PROGRAM.SIGNATURE.VERIFICATION APPLDATA('RACFADM/
   CODE.SIGNATURE.VERIFICATION.KEYRING')
   ```

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

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

5. Activate PROGRAM control, if not already active.

   ```
   SETROPTS WHEN(PROGRAM)
   ```

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

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

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

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

7. Refresh the PROGRAM class.

   ```
   SETROPTS WHEN(PROGRAM) REFRESH
   ```

8. Contact your system programmer to complete this step.
   a. Notify your system programmer to initialize program signature verification by running the IRRVERLD program which loads and verifies the program verification module IRRPVERS. For programming information, refer to APAR OA26109 (RACF).
   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 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 information provided by APAR OA26109 (RACF) for customizing the SIGAUDIT operand within the SIGVER segment.

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

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

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

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

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

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

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

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

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

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

RDEFINE PROGRAM 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 on using VLF see ″VLF considerations for program signature verification″ in the documentation provided by RACF APAR OA26109.

Certificate Stores

To use FIPS mode, certificates can be stored in either a SAF key ring, PKCS #11 token, or a FIPS mode key database. All certificates in a certificate chain to be used by a FIPS enabled application must use algorithms and key sizes as specified in Table 1 on page 3.

SAF keyrings and PKCS #11 tokens

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

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

Key database files

To use a key database in FIPS mode, it needs to be created as a FIPS mode database. Key databases created through gskkyman not explicitly specifying FIPS during creation, or created through an application not executing in FIPS mode, cannot be used by an application executing in FIPS mode. To create a FIPS mode key database using gskkyman, see “Creating, Opening and Deleting a Key Database File” on page 184. 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 78).

The following are key points when using FIPS key databases:

- Only certificates that meet the requirements for FIPS (see Table 1 on page 3) can be added to a FIPS key database.
- A FIPS key database may only be modified if executing in FIPS mode. When opening an existing FIPS key database, gskkyman ensures that it is executing in FIPS mode. If an application modifies the key database via the Certificate Management Services (CMS) APIs, then it too must ensure it is executing in FIPS mode.
- A FIPS key database can be used in non-FIPS mode if it is opened for read only.
- A non-FIPS key database cannot be opened while executing in FIPS mode.

gskkyman automatically detects when a FIPS mode key database is opened, and executes in FIPS mode. This ensures that only certificates or certificate requests that meet the FIPS mode requirements in Table 1 on page 3 may be added to the key database.

Application Changes

In order to use System SSL in FIPS mode, application changes are required. By default, all applications that use System SSL execute in non-FIPS mode. The application needs to request System SSL execute in FIPS mode in the very early stages of interaction with the System SSL API. The application does this by invoking the function gsk_fips_state_set (see “gsk_fips_state_set()” on page 78). In order to set FIPS mode, gsk_fips_state_set must be executed prior to all other System SSL functions with the exception of gsk_get_cms_vector, gsk_get_ssl_vector and gsk_fips_state_query. It is possible to switch to non-FIPS mode at a later time. It is not possible to switch from non-FIPS mode to FIPS mode at any time.

When executing in FIPS mode and a severe cryptographic problem is encountered, one of the following return codes will be returned from the API executing at the time of failure. These return codes should be treated as severe and the application should be terminated and restarted. If execution continues, the failing cryptographic function will continue to fail.

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

The sample files (see *z/OS Cryptographic Services System SSL Programming*, SC24-5901) client.cpp and server.cpp demonstrate the use of gsk_fips_state_set to set the application to run in FIPS mode. In both cases, the gsk_fips_state_set function is invoked prior to any other System SSL function.

SSL Started Task

The System SSL started task (GSKSRVR) executes in non-FIPS mode as default. In order for the GSKSRVR started task to execute in FIPS mode, environment variable GSK_FIPS_STATE must be specified and set to GSK_FIPS_STATE_ON in the envar file in the GSKSRVR home directory. If the GSKSRVR is unable to execute in FIPS mode (for example, the Level 3 FMID JCPT3A1 is not installed), it will execute in non-FIPS mode after issuing message GSK01054E (see “SSL Started Task Messages (GSK01nnn)” on page 210).

Sysplex Session ID Cache

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

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

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

Chapter 3. API Reference

This topic describes a subset of application programming interfaces (APIs) affected by APAR OA26457.

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_INSUFFICIENT_STORAGE]
Insufficient storage is available.

[GSK_INVALID_HANDLE]
The connection handle is not valid.

[GSK_INVALID_STATE]
The connection is not initialized.

Usage

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

These data identifiers are supported:

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

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.

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 returned by **gsk_environment_open()** or an SSL connection handle returned by **gsk_secure_socket_open()**.

enum_id
Specifies the enumeration identifier.

enum_value
Specifies the enumeration value.

Results

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

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

[GSK_INVALID_HANDLE]
The handle is not valid.

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

Usage

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

These enumeration identifiers are supported:

GSK_CRL_SECURITY_LEVEL

Specify the level of security to be used when contacting an LDAP server. In order to check for revoked certificates in a Certificate Revocation List (CRL). CRLs located will be cached according to the GSK_CRL_CACHE_TIMEOUT setting of the SSL environment. To enforce contact with the LDAP server for each CRL check, CRL caching must be disabled. If a CRL is not defined an empty CRL will be placed in the CRL cache to prevent repeated calls to the LDAP server. This entry will not be cleared until the CRL cache timeout is reached. GSK_CRL_SECURITY_LEVEL can only be specified at the environment level.

Three levels of security are available:

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

- GSK_CRL_SECURITY_LEVEL_HIGH - Certificate validation requires the LDAP server to be contactable, and a CRL to be defined.

GSK_CLIENT_AUTH_ALERT

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

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

GSK_CLIENT_AUTH_TYPE

Specifies GSK_CLIENT_AUTH_FULL_TYPE to validate client certificates. If a certificate is not valid, the connection is not started and an error code is returned by the **gsk_secure_socket_init()** routine. If an LDAP server is specified, the LDAP server is queried for CA certificates and certificate revocation lists. If the LDAP server is not available, only local validation will be performed. If no client certificate is received, 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.

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

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

GSK_PROTOCOL_SSLV2

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

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

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

GSK_PROTOCOL_SSLV3

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

GSK_PROTOCOL_SSLV3 can be specified for an SSL environment or an SSL connection.

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

GSK_PROTOCOL_TLSV1

Specifies GSK_PROTOCOL_TLSV1_ON to enable the TLS Version 1 protocol or GSK_PROTOCOL_TLSV1_OFF to disable the TLS Version 1 protocol.

GSK_PROTOCOL_TLSV1 can be specified for an SSL environment or an SSL connection.

GSK_SESSION_TYPE

Specifies GSK_CLIENT_SESSION to perform the SSL handshake as a client, GSK_SERVER_SESSION to perform the SSL handshake as a server, or GSK_SERVER_SESSION_WITH_CL_AUTH to perform the SSL handshake as a server requiring client authentication.

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

GSK_SYSPLEX_SIDCACHE

Returns GSK_SYSPLEX_SIDCACHE_ON if sysplex session caching is enabled for this application

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

GSK_T61_AS_LATIN1

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

gsk_environment_open()

Creates a SSL environment.

Format

```
#include <gskssl.h>

gsk_status gsk_environment_open (
                                 gsk_handle *     env_handle)
```

Parameters

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

Results

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

[GSK_ATTRIBUTE_INVALID_ENUMERATION]
The value of an environment variable is not valid.

[GSK_ATTRIBUTE_INVALID_LENGTH]
The length of an environment variable value is not valid.

[GSK_ATTRIBUTE_INVALID_NUMERIC_VALUE]
The value of an environment variable is not valid.

[GSK_INSUFFICIENT_STORAGE]
Insufficient storage is available.

Usage

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

When not executing in FIPS mode, the following default values are set:

- SSL V2, SSL V3, and TLS V1 are enabled
- 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 domestic encryption is enabled and ″642″ otherwise

- The SSL V3 cipher specification is set to ″050435363738392F303132330A1613100D0915120F0C0306020100″ if domestic encryption is enabled and ″0915120F0C0306020100″ otherwise

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

- TLS V1 is enabled
- The connection type is set to CLIENT
- The connection timeout is set to 86400 seconds
- The cache size is set to 512
- The sysplex session cache is disabled
- The default key will be used
- No revoked certificate checking performed
- The default callback routines will be used
- The cipher specification is set to ″35363738392F303132330A1613100D″

These SSL V2 cipher specifications are supported:

- ″1″ = 128-bit RC4 encryption with MD5 message authentication (128-bit secret key)
- ″2″ = 128-bit RC4 export encryption with MD5 message authentication (40-bit secret key)
- ″3″ = 128-bit RC2 encryption with MD5 message authentication (128-bit secret key)
- ″4″ = 128-bit RC2 export encryption with MD5 message authentication (40-bit secret key)
- ″6″ = 56-bit DES encryption with MD5 message authentication (56-bit secret key)
- ″7″ = 168-bit Triple DES encryption with MD5 message authentication (168-bit secret key)

These SSL V3 cipher specifications are supported:

- ″00″ = No encryption or message authentication and RSA key exchange
- ″01″ = No encryption with MD5 message authentication and RSA key exchange
- ″02″ = No encryption with SHA-1 message authentication and RSA key exchange
- ″03″ = 40-bit RC4 encryption with MD5 message authentication and RSA key exchange
- ″04″ = 128-bit RC4 encryption with MD5 message authentication and RSA key exchange
- ″05″ = 128-bit RC4 encryption with SHA-1 message authentication and RSA key exchange
- ″06″ = 40-bit RC2 encryption with MD5 message authentication and RSA key exchange
- ″09″ = 56-bit DES encryption with SHA-1 message authentication and RSA key exchange
- ″0A″ = 168-bit Triple DES encryption with SHA-1 message authentication and RSA key exchange
- ″0C″ = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate
- ″0D″ = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate
- ″0F″ = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- ″10″ = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- ″12″ = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate
- ″13″ = 168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate
- ″15″ = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate

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

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

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

| These environment variables are processed:

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

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

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

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

GSK_KEYRING_PW
Specifies the password for the key database.

GSK_KEYRING_STASH
Specifies the name of the key database password stash file. The stash file name always has an

extension of ″.sth″ and the supplied name will be changed if it does not have the correct extension. The GSK_KEYRING_PW environment variable will be used instead of the GSK_KEYRING_STASH environment variable if it is also specified.

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

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

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

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

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

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

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

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

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

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

GSK_V2_CIPHER_SPECS
Specifies the SSL V2 cipher specifications in order of preference as a null-terminated string consisting of 1 or more 1-character values. Valid cipher specifications that are not supported due to the installed cryptographic level will be skipped when the connection is initialized.

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

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

GSK_V3_CIPHER_SPECS
Specifies the SSL V3 cipher specifications in order of preference as a null-terminated string consisting of 1 or more 2-character values. The SSL V3 cipher specifications are used for both the SSL V3 and the TLS V1 protocols. Valid cipher specifications that are not supported due to the installed cryptographic level will be skipped when the connection is initialized.

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

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

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 runtime is Version 3 Release 20. 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.

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_ERR_BAD_CERT]
Certificate is not valid.

[GSK_ERR_BAD_DATE]
Certificate is not valid yet or is expired.

[GSK_ERR_BAD_KEYFILE_LABEL]
The specified key is not found in the key database or the key is not trusted.

[GSK_ERR_BAD_MAC]
Message verification failed.

[GSK_ERR_BAD_MESSAGE]
Incorrectly-formatted message received from peer application.

[GSK_ERR_BAD_PEER]
Peer application has violated the SSL protocol.

[GSK_ERR_CERT_VALIDATION]
Certificate validation error.

[GSK_ERR_CERTIFICATE_REVOKED]
Peer certificate is revoked.

[GSK_ERR_CRYPTO]
Cryptographic error detected.

[GSK_ERR_INCOMPATIABLE_KEY]
Certificate key is not compatible with cipher suite.

[GSK_ERR_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_MULTIPLE_DEFAULT]
Multiple keys are marked as the default.

[GSK_ERR_MULTIPLE_LABEL]
Multiple certificates exist for label.

[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_SELF_SIGNED]
A self-signed certificate cannot be validated.

[GSK_ERR_SOCKET_CLOSED]
Socket connection closed by peer application.

[GSK_ERR_RNG]
Error encountered when generating random bytes.

[GSK_ERR_UNKNOWN_CA]
A certification authority certificate is missing.

[GSK_ERR_UNSUPPORTED_CERTIFICATE_TYPE]
The certificate type is not supported by System SSL.

[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.

An SSL handshake is performed as part of the processing of the **gsk_secure_socket_init()** routine. This establishes the server identity and optionally the client identity. It also negotiates the cryptographic parameters to be used for the connection. The client and server will attempt to use the highest available protocol version as determined by the intersection of the enabled protocol versions for the client and the

server. Thus, TLS V1 will be used if it is enabled on both the client and the server, dropping back to SSL V3 if TLS V1 cannot be used and SSL V3 is enabled, then dropping back to SSL V2 if SSL V3 cannot be used and SSL V2 is enabled. Note that SSL V2 is not as secure as SSL V3 or TLS V1 and should be disabled whenever possible to avoid attacks which force the client and server to drop back to SSL V2 even though they are capable of using SSL V3 or TLS V1. When operating in FIPS mode, only the TLS V1 protocol is used.

The client sends a list of ciphers it supports during the SSL handshake. The server application uses this list, and the defined ciphers supported by the server, to determine the cipher to be used during the SSL handshake. If the client is operating in FIPS mode, then the list provided only contains FIPS ciphers. A server executing in FIPS mode will only use FIPS ciphers. The cipher selection is done by looking through the server's cipher list for a match in the client's list. The first matching cipher is used.

The server certificate can use either RSA or DSA as the public/private key algorithm. In FIPS mode, the RSA or DSA key size must be at least 1024 bits. An RSA certificate can be used with an RSA, fixed Diffie-Hellman or ephemeral Diffie-Hellman key exchange. A DSA certificate can be used with either a fixed or ephemeral Diffie-Hellman key exchange. In FIPS mode, Diffie-Hellman key size must be 2048 bits. If the server's certificate contains a key usage extension during the SSL handshake, it must allow key usage as follows:

- RSA certificates using export restricted ciphers (40-bit RC4 encryption and 40-bit RC2 encryption) with a public key size greater than 512 bits must allow digital signature. If operating in FIPS mode, export restricted ciphers cannot be selected.
- RSA or DSA certificates using fixed Diffie-Hellman key exchange must allow key agreement.
- Other RSA certificates must allow key encipherment.
- DSA certificates using ephemeral Diffie-Hellman key exchange must allow digital signature.

The client certificate must support digital signatures. This means the certificate key usage extension (if any) must allow digital signature. The key algorithm can be either the RSA encryption algorithm or the Digital Signature Standard algorithm (DSA).

The SSL server always provides its certificate to the SSL client as part of the handshake. The client always performs server authentication using the certificate provided by the server. Depending upon the server handshake type, the server may ask the client to provide its certificate. The key label stored in the connection is used to retrieve the certificate from the key database, key ring or token. The default key will be used if no label is set. The key record must contain both an X.509 certificate and a private key. Refer to the **gsk_validate_certificate()** routine for a description of the steps which are performed during certificate validation.

The **gsk_secure_socket_init()** routine can return GSK_WOULD_BLOCK_READ or GSK_WOULD_BLOCK_WRITE if the socket is in non-blocking mode. The connection is not initialized in this case and the application must call **gsk_secure_socket_init()** again when the socket is ready to accept a read request (GSK_WOULD_BLOCK_READ) or a write request (GSK_WOULD_BLOCK_WRITE). The application must provide its own callback routine for **io_setsocketoptions()** in order to have the SSL handshake processed in non-blocking mode (the default **io_setsocketoptions()** routine will place the socket into blocking mode during the handshake processing).

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

Chapter 4. Certificate Management Services (CMS) API Reference

This topic describes the Certificate Management Services (CMS) APIs affected by APAR OA26457.

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_INCORRECT_DBTYPE]
The record type is not supported for the database type.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
No private key is provided for a record type that requires a private key.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_RECTYPE_NOT_VALID]
The record type is not valid.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_add_record()** routine adds a record to a key or request database. The database must be open for update in order to add records. Unused and reserved fields in the gskdb_record structure must be initialized to zero. An error will be returned when adding a certificate to a key database if the database already contains the certificate. If the record has a private key, the encrypted private key will be generated from the private key supplied in the database record.

The *recordType* field identifies the database record type as follows:

gskdb_rectype_certificate
The record contains an X.509 certificate

gskdb_rectype_certKey
The record contains an X.509 certificate and private key

gskdb_rectype_keyPair
The record contains a PKCS #10 certification request and private key

The *recordFlags* field is a bit field with these values:

GSKDB_RECFLAG_TRUSTED
The certificate is trusted

GSKDB_RECFLAG_DEFAULT
This is the default key

A unique record identifier is assigned when the record is added to the database and will be returned to the application in the *recordId* field. If the record contains an X.509 certificate, the *issuerRecordId* field will be set to the record identifier of the certificate issuer.

The record label is used as a friendly name for the database entry and is in the local code page. It can be set to any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be set to an empty string.

If the record contains an X.509 certificate, the certificate will be validated and the record will not be added to the database if the validation check fails. If the database is a FIPS key database, then the certificate must use only FIPS algorithms and key sizes.

With the exception of the record label, all character strings are specified using UTF-8.

The database file is updated as part of the **gsk_add_record()** processing. A temporary database file is created using the same name as the database file with ″.new″ appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.

gsk_change_database_password()

Changes the database password.

Format

```
#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.

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_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_CA_NOT_SUPPLIED]
Signing Certificate Authority Certificate not supplied.

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_SUBJECT_NAME]
The subject name is not valid.

[CMSERR_EXPIRED]
The signer certificate is expired.

[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 2459 (X.509 Public Key Infrastructure). The certificate will be signed using the certificate as supplied by the *issuer_certificate* parameter. A certification authority (CA) certificate will have basic contraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have basic constraints and key usage extensions as follows:

- An RSA key can be used for authentication, digital signature, and data encryption. An RSA key can be used for both CA certificates and end user certificates.
- A DSS key can be used for authentication and digital signature. A DSS key can be used for both CA certificates and end user certificates.
- A Diffie_Hellman key can be used for key agreement. A Diffie-Hellman key can be used only for end user certificates.

The new certificate is returned in the supplied x509_certificate structure.

These signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

A CA certificate will have SubjectKeyIdentifier, KeyUsage and BasicConstraints extensions while an end user certificate will have SubjectKeyIdentifier and KeyUsage extensions. The application can supply additional extensions through the *extensions* parameter. A KeyUsage or BasicConstraints extension provided by the application will replace the default extension constructed for the certificate. A SubjectKeyIdentifier extension will be created if the CA certificate has a SubjectKeyIdentifier extension.

gsk_construct_renewal_request()

Constructs a certification renewal request as described in PKCS #10.

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()** function constructs a certification request as described in PKCS #10 (Certification Request Syntax Standard) and returns the constructed request in the pkcs_cert_request structure *request*.

gsk_construct_renewal_request()

The **gsk_encode_export_request()** function can be called to create an export file containing the request for transmission to the certification authority.

The certification request will be signed using the key specified by the *private_key* parameter and the signature algorithm specified by the *signature_algorithm* parameter.

These signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

The *extensions* parameter can be used to provide certificate extensions for inclusion in the certification request. Whether or not a particular certificate extension will be included in the new certificate is determined by the certification authority.

gsk_construct_self_signed_certificate()

Constructs a self-signed certificate and returns it to the caller.

Format

```
#include <gskcms.h>

gsk_status gsk_construct_self_signed_certificate (
                                    x509_algorithm_type     signature_algorithm,
                                    const_char *            subject_name,
                                    int                     num_days,
                                    gsk_boolean             ca_certificate,
                                    x509_extensions *       extensions,
                                    x509_public_key_info *  public_key,
                                    pkcs_private_key_info * private_key,
                                    x509_certificate *      subject_certificate)
```

Parameters

signature_algorithm
Specifies the signature algorithm used to sign the constructed certificate.

subject_name
Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

num_days
Specifies the number of days for the certificate validity period as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified).

ca_certificate
Specify TRUE if this is a certification authority certificate or FALSE if this is an end user certificate.

extensions
Specifies the certificate extensions for the new certificate. Specify NULL for this parameter if no certificate extensions are supplied.

public_key
Specifies the public key for the constructed certificate.

private_key
Specifies the private key for the constructed certificate.

subject_certificate
Contains the constructed certificate.

Results

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

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_SUBJECT_NAME]
The subject name is not valid.

[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 2459 (X.509 Public Key Instrastructure). 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 509_certificate structure *subject_certificate*.

These signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

If not in FIPS mode, an RSA key size must be between 512 and 4096 bits and will be rounded up to a multiple of 16 bits. A DSA key size must be between 512 and 1024 bits and will be rounded up to a multiple of 64 bits.

In FIPS mode, an RSA key size must be between 1024 and 4096 bits and will be rounded up to a multiple of 16 bits. A DSA key size must be 1024 bits.

Note: A self-signed end-entity certificate (server or client certificate) is not recommended for use in production environments and should only be used to facilitate test environments prior to production. Self-signed certificates do not imply any level of security or authenticity of the certificate because, as their name implies, they are signed by the same key that is contained in the certificate. On the other hand, certificates that are signed by a certificate authority indicate that, at least at the time of signature, the certificate authority approved the information contained in the certificate.

gsk_construct_signed_certificate()

Constructs a signed certificate for a certificate request.

Format

```
#include <gskcms.h>

gsk_status gsk_construct_signed_certificate (
                                    pkcs_cert_key *      signer_certificate,
                                    pkcs_cert_request *  request,
                                    x509_algorithm_type  signature_algorithm,
                                    int                  num_days,
                                    gsk_boolean          ca_certificate,
                                    x509_extensions *    extensions,
                                    x509_certificate *   certificate)
```

Parameters

signer_certificate
Specifies the signing certificate with private key.

request
Specifies the PKCS #10 certification request stream in either binary DER-encoded format or in Base64 format. A Base64 stream is in the local code page.

signature_algorithm
Specifies the signature algorithm used to sign the constructed certificate.

num_days
Specifies the number of days for the certificate validity period as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified).

ca_certificate
Specify TRUE if this is a certification authority certificate or FALSE if this is an end user certificate.

extensions
Specifies the certificate extensions for the new certificate. Specify NULL for this parameter if no certificate extensions are supplied.

certificate
Contains the constructed signed certificate.

Results

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

[CMSERR_ALG_NOT_SUPPORTED]
The key algorithm or the signature algorithm is not valid.

[CMSERR_BAD_ENCODING]
The certificate request stream is not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_SIGNATURE]
The request signature is not correct.

CMSERR_CA_NOT_SUPPLIED[]
CA certificate is not supplied.

[CMSERR_EXPIRED]
The signer certificate is expired.

[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 2459 (X.509 Public Key Infrastructure). The new certificate will be signed using the certificate specified by the *signer_certificate* parameter. A certification authority certificate will have basic constraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have basic constraints and key usage extensions which allow the certificate to be used for authentication, digital signatures, and data encryption (except for a DSA key which cannot be used for data encryption). The certificate expiration will be set to the earlier of the requested expiration date and the expiration date of the signing certificate.

The signing certificate must have an associated private key, the Basic Constraints extension must either be omitted or must have the CA indicator set, and the KeyUsage extension must either be omitted or must allow signing certificates. The subject key identifier from the signing certification will be copied to the signed certificate as the authority key identifier.

A CA certificate will have SubjectKeyIdentifier, KeyUsage and BasicConstraints extensions while an end user certificate will have SubjectKeyIdentifier and KeyUsage extensions. An end user certificate 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, KeyUsage or BasicConstraints extension provided by the application will replace the default extension created for the certificate. A SubjectKeyIdentifier extension provided by the application will be ignored.

Certificate extensions can also be contained within the certification request. A certificate extension supplied by the application will override a certificate extension of the same type contained in the certification request. The certificate extension found in the certification request will be copied unmodified to the new certificate with these exceptions:

- The AuthorityInfoAccess, AuthorityKeyIdentifier, BasicConstraints, CrlDistributionPoints, IssuerAltName, NameConstraints, PolicyConstraints, PolicyMappings, and PrivateKeyUsagePeriod extensions will not be copied.
- The keyCertSign and crlSign flags in the KeyUsage extension will be modified based upon the value of the *ca_certificate* parameter.

These signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

No certification path validation is performed by the **gsk_construct_signed_certificate()** routine. An error will be returned if the requested subject name is the same as the subject name in the signing certificate.

gsk_create_certification_request()

Creates a PKCS #10 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.

[CMSERR_FIPS_KEY_PAIR_CONSISTENCY]
FIPS mode key generation failed pair-wise consistency check.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certification requests.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_create_certification_request()** routine creates a request for a new certificate as described in PKCS #10 (Certification Request Syntax Standard). The request is then stored in the request database. The **gsk_export_certification_request()** routine can be called to create an export file containing the request for transmission to the certification authority.

The **gsk_create_certification_request()** routine is similar to the **gsk_create_renewal_request()** routine. Both routines create a PKCS #10 certification request. The difference is the **gsk_create_certification_request()** routine generates a new public/private key pair while the **gsk_create_renewal_request ()** routine uses the public/private key pair provided by the application.

These signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

If not in FIPS mode, an RSA key size must be between 512 and 4096 bits and will be rounded up to a multiple of 16 bits. A DSA key size must be between 512 and 1024 bits and will be rounded up to a multiple of 64 bits.

| In FIPS mode, an RSA key size must be between 1024 and 4096 bits and will be rounded up to a multiple
| of 16 bits. A DSA key size must be 1024 bits.

The record label is used as a friendly name for the database entry. It can be any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string.

The *extensions* parameter can be used to provide certificate extensions for inclusion in the certification request. Whether or not a particular certificate extension will be included in the new certificate is determined by the certification authority.

The database must be open for update in order to add the new request. The database file is updated as part of the **gsk_create_certification_request()** processing. A temporary database file is created using the same name as the database file with ″.new″ appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.

gsk_create_database()

Creates a key or request database.

Format

```
#include <gskcms.h>

gsk_status gsk_create_database (
                            char *                filename,
                            char *                password,
                            gskdb_database_type   db_type,
                            gsk_size              record_length,
                            gsk_time              pwd_expiration,
                            gsk_handle *          db_handle)
```

Parameters

filename
Specifies the database file name in the local code page. The length of the fully-qualified file name cannot exceed 251.

password
Specifies the database password in the local code page. The password must consist of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string. The user will be prompted to enter the password if NULL is specified for this parameter.

db_type
Specifies the database type and must be gskdb_dbtype_key for a key database or gskdb_dbtype_request for a certification request database.

record_length
Specifies the database record length. The default record length will be used if zero is specified for this parameter. All records in the database will have this length. The minimum record length is 2500. The default record length is 5000.

pwd_expiration
Specifies the database password expiration time as the number of seconds since the POSIX epoch. A value of 0 indicates the password does not expire.

db_handle
Returns the database handle. The application should call the **gsk_close_database()** routine when it no longer needs access to the database.

Results

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

[CMSERR_BAD_FILENAME]
The database file name is not valid.

[CMSERR_DB_EXISTS]
The database already exists.

[CMSERR_INCORRECT_DBTYPE]
The database type is not valid.

[CMSERR_IO_CANCELED]
The user canceled the password prompt.

[CMSERR_IO_ERROR]
An input/output request failed.

[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/read when not in FIPS mode
- Cannot be used while executing in FIPS mode

gsk_create_database_renewal_request()

Creates a PKCS #10 certification renewal request.

Format

```
#include <gskcms.h>

gsk_status gsk_create_database_renewal_request (
                                gsk_handle                db_handle,
                                const char *              label,
                                x509_public_key_info *    public_key,
                                pkcs_private_key_info *   private_key,
                                x509_algorithm_type       signature_algorithm,
                                const char *              subject_name,
                                x509_extensions *         extensions)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine or the **gsk_open_database()** routine. This must be a request database and not a key database.

label
Specifies the label for the request database record. The label is specified in the local code page.

public_key
Specifies the public key for the certification request.

private_key
Specifies the private key for the certification request.

signature_algorithm
Specifies the signature algorithm to be used for the request signature.

subject_name
Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

extensions
Specifies certificate extensions to be included in the certification request. Specify NULL for this parameter if no certificate extensions are provided.

Results

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

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not valid.

[CMSERR_BACKUP_EXISTS]
The backup file already exists.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_LABEL]
The record label is not valid.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certification requests.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_KEY_MISMATCH]
The supplied private key cannot be used to sign a certificate or the private key type is not supported for the requested signature algorithm.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_create_database_renewal_request()** routine creates a certification request as described in PKCS #10 (Certification Request Syntax Standard). The request is then stored in the request database. The **gsk_export_certification_request()** routine can be called to create an export file containing the request for transmission to the certification authority.

The **gsk_create_database_renewal_request()** routine is similar to the **gsk_create_certification_request()** routine. Both routines create a PKCS #10 certification request. The difference is the **gsk_create_certification_request()** routine generates a new public/private key pair while the **gsk_create_database_renewal_request()** routine uses the public/private key pair provided by the application.

The renewal request will be signed using the key specified by the *private_key* parameter and the signature algorithm specified by the *signature_algorithm* parameter.

These signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

gsk_create_database_signed_certificate()

Creates a signed certificate as part of a set of certificates.

Format

```
#include <gskcms.h>

gsk_status gsk_create_database_signed_certificate (
                                gsk_handle              db_handle,
                                const char *            ca_label,
                                const char *            record_label,
                                x509_algorithm_type     key_algorithm,
                                int                     key_size,
                                gsk_buffer *            key_parameters,
                                x509_algorithm_type     signature_algorithm,
                                const char *            subject_name,
                                int                     num_days,
                                gsk_boolean             ca_certificate,
                                x509_extensions *       extensions )
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine or the **gsk_open_database()** routine. This must be a key database and not a request database.

ca_label
Specifies the label of the certificate to be used to sign the new certificate. The key usage for the certificate must allow certificate signing. The label is specified in the local code page.

record_label
Specifies the label for the new database record. The label is specified in the local code page.

key_algorithm
Specifies the certificate key algorithm.

key_size
Specifies the certificate key size in bits.

key_parameters
Specifies the key generation parameters. Specify NULL for this parameter if the key algorithm does not require any key parameters.

signature_algorithm
Specifies the signature algorithm used for the certificate signature.

subject_name
Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

num_days
Specifies the number of days for the certificate validity period as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified 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_HANDLE]
The database handle is not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_LABEL]
The record label or CA certificate label is not valid.

[CMSERR_BAD_SUBJECT_NAME]
The subject name is not valid.

[CMSERR_DUPLICATE_EXTENSION]
Supplied extensions contain a duplicate extension.

[CMSERR_EXPIRED]
The signer certificate is expired.

[CMSERR_FIPS_KEY_PAIR_CONSISTENCY]
FIPS mode key generation failed pair-wise consistency check.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_USAGE]
The signer certificate key usage does not allow signing certificates.

[CMSERR_IO_ERROR]
Unable to read or write a database record.

[CMSERR_ISSUER_NOT_CA]
The signer certificate is not for a certification authority.

[CMSERR_KEY_MISMATCH]
The signer certificate key cannot be used to sign a certificate.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
The signer certificate does not have a private key.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_SUBJECT_IS_CA]
The requested subject name is the same as the signer name.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_create_database_signed_certificate()** routine will generate an X.509 certificate as described in RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). The certificate will be signed using an existing certificate as specified by the *ca_label* parameter and the signature algorithm specified by the *signature_algorithm* parameter.

These signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

A certification authority (CA) certificate will have basic constraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have basic constraints and key usage extensions as follows:

- An RSA key can be used for authentication, digital signature, and data encryption.
- A DSS key can be used for authentication and digital signature.
- A Diffie-Hellman key can be used for key agreement.

The new certificate will be stored in the key database using the supplied record label. The **gsk_export_certificate()** routine can be called to create an export file containing the certificate for transmission to another system.

The following key algorithms are supported:

x509_alg_rsaEncryption
RSA encryption - {1.2.840.113549.1.1.1}

x509_alg_idDsa
Didital Signature Standard (DSS) - {1.2.840.10040.4.1}

x509_alg_dhPublicNumber
Diffie-Hellman (DH) - {1.2.840.10046.2.1}

RSA keys

- Can be used for both CA certificates and end user certificates
- Key size when not in FIPS mode is between 512 and 4096 bits rounded up to a multiple of 16

- Key size in FIPS mode is between 1024 and 4096 bits rounded up to a multiple of 16
- No key parameters

DSS keys

- Can be used for both CA certificates and end user certificates
- Key size when not in FIPS mode is between 512 and 1024 bits rounded up to a multiple of 64
- Key size in FIPS mode of 1024 bits
- Key parameters encoded as an ASN.1 sequence consisting of the prime P, the subprime Q and the base G. Refer to FIPS 186-2 (Digital Signature Standard) for more information on the key parameters. The **gsk_generate_key_parameters()** routine can be used to generate the key parameters.

DH keys

- Can be used only for end user certificates
- Key size when not in FIPS mode is between 512 and 2048 bits rounded up to a multiple of 64
- Key size in FIPS mode of 2048 bits
- Key parameters encoded as an ASN.1 sequence consisting of the prime P, the base G, the subprime Q and the subgroup factor J. Refer to RFC 2631 (Diffie-Hellman Key Agreement Method) for more information on the key parameters. The **gsk_generate_key_parameters()** routine can be used to generate the key parameters.

The record label is used as a friendly name for the database entry. It can be any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string.

A CA certificate will have SubjectKeyIdentifier, KeyUsage and BasicConstraints extensions while an end user certificate will have SubjectKeyIdentifier and KeyUsage extensions. An AuthorityKeyIdentifier extension will be created if the signing certificate has a SubjectKeyIdentifier extension. The application can supply additional extensions through the extensions parameter. An AuthorityKeyIdentifier, KeyUsage or BasicConstraints extension provided by the application will replace the default extension constructed for the certificate, however a SubjectKeyIdentifier extension provided by the application will be ignored.

The database must be open for update in order to add the new certificate. The database file is updated as part of the **gsk_create_database_signed_certificate()** processing. A temporary database file is created using the same name as the database file with ″.new″ appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.

gsk_create_renewal_request()

Creates a PKCS #10 certification renewal request.

This function is deprecated. Use **gsk_create_database_renewal_request()** instead.

Format

```
#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_extentions *         extensions)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine or the **gsk_open_database()** routine. This must be a request database and not a key database.

label
Specifies the label for the request database record. The label is specified in the local code page.

public_key
Specifies the public key for the certification request.

private_key
Specifies the private key for the certification request.

subject_name
Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

extensions
Specifies certificate extensions to be included in the certification request. Specify NULL for this parameter if no certificate extensions are provided.

Results

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

[CMSERR_BACKUP_EXISTS]
The backup file already exists.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_LABEL]
The record label is not valid.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certification requests.

[CMSERR_IO_ERROR]
Unable to write record.

[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_renewal_request()** routine creates a certification request as described in PKCS #10 (Certification Request Syntax Standard). The request is then stored in the request database. The **gsk_export_certification_request()** routine can be called to create an export file containing the request for transmission to the certification authority.

The **gsk_create_renewal_request()** routine is similar to the **gsk_create_certification_request()** routine. Both routines create a PKCS #10 certification request. The difference is the **gsk_create_certification_request()** routine generates a new public/private key pair while the **gsk_create_renewal_request()** routine uses the public/private key pair provided by the application.

The record label is used as a friendly name for the database entry. It can be any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string.

The extensions parameter can be used to provide certificate extensions for inclusion in the certification request. Whether or not a particular certificate extension will be included in the new certificate is determined by the certification authority.

gsk_create_self_signed_certificate()

Creates a self-signed certificate.

Format

```
#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_FIPS_KEY_PAIR_CONSISTENCY]
FIPS mode key generation failed pair-wise consistency check.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_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 2459 (X.509 Public Key Infrastructure). A certification authority certificate will have basic constraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have no basic constraints or key usage limitations. The new certificate is then stored in the key database. The **gsk_export_certificate()** routine can be called to create an export file containing the certificate for transmission to another system.

These signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

If not in FIPS mode, an RSA key size must be between 512 and 4096 bits and will be rounded up to a multiple of 16 bits. A DSA key size must be between 512 and 1024 bits and will be rounded up to a multiple of 64 bits.

In FIPS mode, an RSA key size must be between 1024 and 4096 bits and will be rounded up to a multiple of 16 bits. A DSA key size must be 1024 bits.

The record label is used as a friendly name for the database entry. It can be any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string.

A CA certificate will have SubjectKeyIdentifier, KeyUsage and BasicConstraints extensions while an end user certificate will have a SubjectKeyIdentifier and a CA BasicConstraints extension. The application can supply additional extensions through the *extensions* parameter. A KeyUsage or BasicContraints extension provided by the application will replace the default extension created for the certificate. A SubjectKeyIdentifier extension provided by the application will be ignored. The database must be open for update in order to add the new certificate. The database file is updated as part of the **gsk_create_self_signed_certificate()** processing. A temporary database file is created using the same name as the database file with ″.new″ appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.

Note: A self-signed end-entity certificate (server or client certificate) is not recommended for use in production environments and should only be used to facilitate test environments prior to production. Self-signed certificates do not imply any level of security or authenticity of the certificate because, as their name implies, they are signed by the same key that is contained in the certificate. On the other hand, certificates that are signed by a certificate authority indicate that, at least at the time of signature, the certificate authority approved the information contained in the certificate.

gsk_create_signed_certificate_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.

[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_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_USAGE]
The signer certificate key usage does not allow signing certificates

[CMSERR_IO_ERROR]
Unable to write record.

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

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_create_signed_certificate_record()** routine will generate an X.509 certificate as described in RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). The new certificate will be signed using the certificate specified by the *label* parameter and the signature algorithm specified by the *signature_algorithm* parameter.

The following signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

If not in FIPS mode, an RSA key size must be between 512 and 4096 bits and will be rounded up to a multiple of 16 bits. A DSA key size must be between 512 and 1024 bits and will be rounded up to a multiple of 64 bits.

In FIPS mode, an RSA key size must be between 1024 and 4096 bits and will be rounded up to a multiple of 16 bits. A DSA key size must be 1024 bits.

A certification authority certificate will have basic constraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have basic constraints and key usage extensions which allow the certificate to be used for authentication, digital signatures, and data encryption (except for a DSA key which cannot be used for data encryption). The certificate expiration date will be set to the earlier of the requested expiration date and the expiration date of the signing certificate.

The signing certificate must have an associated private key, the BasicConstraints extension must either be omitted or must have the CA indicator set, and the KeyUsage extension must either be omitted or must allow signing certificates.

A CA certificate will have SubjectKeyIdentifier, KeyUsage and BasicConstraints extensions while an end user certificate will have SubjectKeyIdentifier and KeyUsage extensions. A CA certificate will have SubjectKeyIdentifier, KeyUsage and BasicConstraints extensions while an end user certificate will have SubjectKeyIdentifier and KeyUsage extensions. An AuthorityKeyIdentifier extension will be created if the signing certificate has a SubjectKeyIdentifier extension. The application can supply additional extensions through the extensions parameter. An AuthorityKeyIdentifier, KeyUsage or BasicConstraints extension provided by the application will replace the default extension created for the certificate, however a SubjectKeyIdentifier extension provided by the application will be ignored.

Certificate extensions can also be contained within the certification request. A certificate extension supplied by the application will override a certificate extension of the same type contained in the certification request. The certificate extensions found in the certification request will be copied unmodified to the new certificate with these exceptions:

- The AuthorityInfoAccess, AuthorityKeyIdentifier, BasicConstraints, CrlDistributionPoints, IssuerAltName, NameConstraints, PolicyConstraints, PolicyMappings, and PrivateKeyUsagePeriod extensions will not be copied
- The keyCertSign and crlSign flags in the KeyUsage extension will be modified based upon the value of the *ca_certificate* parameter.

No certification path validation is performed by the **gsk_create_signed_certificate_record()** routine. An error will be returned if the requested subject name is the same as the subject name in the signing certificate.

gsk_create_signed_certificate_set()

Creates a signed certificate as part of a set of certificates.

This function is deprecated. Use **gsk_create_database_signed_certificate()** instead.

Format

```
#include <gskcms.h>

gsk_status gsk_create_signed_certificate_set (
                                gsk_handle              db_handle,
                                const char *            ca_label,
                                const char *            record_label,
                                x509_algorithm_type     key_algorithm,
                                int                     key_size,
                                gsk_buffer *            key_parameters,
                                const char *            subject_name,
                                int                     num_days,
                                gsk_boolean             ca_certificate,
                                x509_extensions         extensions )
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine. This must be a key database and not a request database.

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_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_EXPIRED]
The signer certificate is expired.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_USAGE]
The signer certificate key usage does not allow signing certificates.

[CMSERR_IO_ERROR]
Unable to read or write a database record.

[CMSERR_ISSUER_NOT_CA]
The signer certificate is not for a certification authority.

[CMSERR_KEY_MISMATCH]
The signer certificate key cannot be used to sign a certificate.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
The signer certificate does not have a private key.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_SUBJECT_IS_CA]
The requested subject name is the same as the signer name.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_create_signed_certificate_set()** routine will generate an X.509 certificate as described in RFC 2459 (X.509 Public Key Infrastructure). The certificate will be signed using an existing certificate as specified by the ca_label parameter. A certification authority (CA) certificate will have basic constraints and

key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have basic constraints and key usage extensions as follows:

- An RSA key can be used for authentication, digital signature, and data encryption
- A DSS key can be used for authentication and digital signature
- A Diffie-Hellman key can be used for key agreement

The new certificate will be stored in the key database using the supplied record label. The **gsk_export_certificate()** routine can be called to create an export file containing the certificate for transmission to another system.

These key algorithms are supported:

x509_alg_rsaEncryption
RSA encryption - {1.2.840.113549.1.1.1}

x509_alg_idDsa
Didital Signature Standard (DSS) - {1.2.840.10040.4.1}

x509_alg_dhPublicNumber
Diffie-Hellman (DH) - {1.2.840.10046.2.1}

RSA keys

- Can be used for both CA certificates and end user certificates
- Key size when not in FIPS mode is between 512 and 4096 bits rounded up to a multiple of 16
- Key size in FIPS mode is between 1024 and 4096 bits rounded up to a multiple of 16
- No key parameters

DSS keys

- Can be used for both CA certificates and end user certificates
- Key size when not in FIPS mode is between 512 and 1024 bits rounded up to a multple of 64
- Key size in FIPS mode of 1024 bits
- Key parameters encoded as an ASN.1 sequence consisting of the prime P, the subprime Q and the base G. Refer to FIPS 186-2 (Digital Signature Standard) for more information on the key parameters. The **gsk_generate_key_parameters()** routine can be used to generate the key parameters.

DH keys

- Can be used only for end user certificates
- Key size when not in FIPS mode is between 512 and 2048 bits rounded up to a multiple of 64
- Key size in FIPS mode of 2048 bits
- Key parameters encoded as an ASN.1 sequence consisting of the prime P, the base G, the subprime Q and the subgroup factor J. Refer to RFC 2631 (Diffie-Hellman Key Agreement Method) for more information on the key parameters. The **gsk_generate_key_parameters()** routine can be used to generate the key parameters.

The record label is used as a friendly name for the database entry. It can be any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string.

A CA certificate will have SubjectKeyIdentifier, KeyUsage and BasicConstraints extensions while an end user certificate will have SubjectKeyIdentifier and KeyUsage extensions. The application can supply additional extensions through the extensions parameter. A KeyUsage or BasicContraints extension provided by the application will replace the default extension created for the certificate. A SubjectKeyIdentifier extension provided by the application will be ignored. An AuthorityKeyIdentifier extension will be created if the CA certificate has a SubjectKeyIdentifier extension.

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_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_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_EXPIRED]
The signer certificate is expired.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_USAGE]
The signer certificate key usage does not allow signing a CRL.

[CMSERR_ISSUER_NOT_CA]
The signer certificate is not for a certification authority.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
The signer certificate does not have a private key.

[CMSERR_RECORD_NOT_FOUND]
The signer certificate is not found in the key database.

Usage

The **gsk_create_signed_crl_record()** routine will generate an X.509 certificate revocation list (CRL) as described in RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). The new CRL will be signed using the certificate specified by the *label* parameter and the signature algorithm specified by the *signature_algorithm* parameter.

The following signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

The number of days until the next CRL is issued will be set to the earlier of the requested date and the expiration of the signing certificate.

The signing certificate must have an associated private key, the BasicConstraints extension must either be omitted or must have the CA indicator set, and the KeyUsage extension must either be omitted or must allow signing certificate revocation lists.

The CRL will have a CRLNumber extension containing the value specified by the *crl_number* parameter. It will also have an AuthorityKeyIdentifier extension if the signing certificate has a SubjectKeyIdentifier extension. The application can supply additional extensions through the *extensions* parameter. An AuthorityKeyIdentifier or CRLNumber extension provided by the application will replace the default extension created for the CRL.

No certification path validation is performed by the **gsk_create_signed_crl_record()** routine.

gsk_decode_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 certificate's chain are discarded). The root certificate for the chain is the final entry in the array.

The certificate and key must have been encoded according to the Personal Information Exchange Syntax (PKCS #12). The supplied stream can be the binary ASN.1 sequence or the Base64 encoding of the ASN.1 sequence. A Base64 encoded stream is assumed to be in the local code page and must include the encoding header and footer lines.

In FIPS mode, the only supported decryption algorithm for the import file is:

- **x509_alg_pbeWithSha1And3DesCbc** - Triple DES with SHA-1 digest.

gsk_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_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.

issuer_certificates
Specifies the certificate chain for the subject certificate.

format
Specifies the export format. These values may be specified:

gskdb_export_pkcs12v1_binary
Binary PKCS #12 Version 1.

gskdb_export_pkcs12v1_base64
Base64 PKCS# 12 Version 1.

gskdb_export_pkcs12v3_binary
Binary PKCS #12 Version 3.

gskdb_export_pkcs12v3_base64
Base64 PKCS #12 Version 3.

algorithm
Specifies the encryption algorithm for the export file. The strong encryption algorithms may not be available depending upon government export regulations. These values may be specified:

x509_alg_pbeWithSha1And40BitRc2Cbc
40-bit RC2 with SHA-1 digest.

x509_alg_pbeWithSha1And128BitRc2Cbc
128-bit RC2 with SHA-1 digest.

x509_alg_pbeWithSha1And40BitRc4
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.

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_BAD_RNG_OUTPUT]
In FIPS mode, random bytes generation produced duplicate output.

[CMSERR_FMT_NOT_SUPPORTED]
An unsupported export file format is specified.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
The certificate does not have a private key.

[CMSERR_PW_INCORRECT]
The password is not correct.

Usage

The **gsk_encode_export_key()** function encodes an X.509 certificate and its private key into a PKCS #12 data stream. The certificate chain for the subject certificate is supplied via the pkcs_certificates structure *issuer_certificates*, with the root certificate being the final entry in the array.

The export byte stream contains the requested certificate, its private key, and the certification chain. A partial certification chain is exported if the complete chain is not supplied in *issuer_certificates*.

gsk_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 must be stored in the SAF database and not in ICSF.

label
Specifies the label for the database record. The label is specified in the local code page.

format
Specifies the export format. These values may be specified:

gskdb_export_pkcs12v1_binary
Binary PKCS #12 Version 1

gskdb_export_pkcs12v1_base64
Base64 PKCS #12 Version 1

gskdb_export_pkcs12v3_binary
Binary PKCS #12 Version 3

gskdb_export_pkcs12v3_base64
Base64 PKCS #12 Version 3

algorithm
Specifies the encryption algorithm for the export file. The strong encryption algorithms may not be available depending upon government export regulations.

These values may be specified for the PKCS #12 Version 1 format:

x509_alg_pb1WithSha1And40BitRc2Cbc
40-bit RC2 with SHA-1 digest

x509_alg_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, there is only one supported encryption algorithm for the export file.

For PKCS#12 Version 3:

x509_alg_pbeWithSha1And3DesCbc
Triple DES with SHA-1 digest.

password
Specifies the password for the export file. The password is in the local code page and must consist of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string. The user will be prompted to enter the password if NULL is specified for this parameter.

stream
Return the byte stream for the encoded certificate. The application should call the **gsk_free_buffer()** routine to release the storage when it is no longer needed.

Results

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

CMSERR_ALG_NOT_SUPPORTED]
The encryption algorithm is not supported.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_LABEL]
The record label or CA certificate label is not valid.

[CMSERR_BAD_RNG_OUTPUT]
In FIPS mode, random bytes generation produced duplicate output.

[CMSERR_FMT_NOT_SUPPORTED]
An unsupported export file format is specified.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
The signer certificate does not have a private key.

[CMSERR_RECORD_NOT_FOUND]
The requested record is not found.

Usage

The **gsk_export_key()** routine exports an X.509 certificate and the associated private key. The certificate can be exported using either the PKCS #12 Version 1 format or the PKCS #12 Version 3 format. This can be either the binary value or the Base64 encoding of the binary value. A Base64 encoded stream will be in the local code page and will include the encoding header and footer lines.

The PKCS #12 Version 1 format is obsolete. However, it is the only format supported by some SSL implementations and must be used when moving a certificate and key to one of those systems. If not running in FIPS mode, you should use either x509_alg_pb1WithSha1And40BitRc2Cbc or x509_alg_pb1withSha1And3DesCbc for interoperability with these older SSL implementations.

The export file will contain the requested certificate, its private key, and the certification chain. A partial certification chain will be exported if the complete chain is not in the database.

gsk_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 due to the current state.

[CMSERR_FIPS_MODE_EXECUTE_FAILED]
The request to execute in FIPS mode failed because the Cryptographic Services Security Level 3 FMID is not installed so that the required System SSL DLLs could not be loaded.

[CMSERR_FIPS_MODE_SWITCH]
The System SSL FIPS mode state cannot be changed to FIPS mode because it is currently not in FIPS mode.

[CMSERR_KATPW_FAILED]
The power on known answer tests failed. FIPS mode cannot be set.

Usage

The gsk_fips_state_set function sets the enumerated value for the System SSL FIPS mode state.

In order to set FIPS mode, this function must be executed prior to all other System SSL API functions with the exception of gsk_get_cms_vector, gsk_get_ssl_vector and gsk_fips_state_query. It is possible to switch to a non-FIPS mode at a later time. It is not possible to switch from non-FIPS mode to FIPS mode at any time.

The following enumerated value are supported:

GSK_FIPS_STATE_ON
FIPS mode state has been set to FIPS mode.

GSK_FIPS_STATE_OFF
FIPS mode state has been set to non-FIPS mode.

gsk_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 as defined in PKCS #3 (Diffie-Hellman Key Agreement Standard) and RFC 2631 (Diffie-Hellman Key Agreement Method). The required key parameters P and G and the optional key parameters Q and J are supplied as an ASN.1-encoded sequence as defined in either PKCS #3 or RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). The return values will be the binary values for Y and X. The key size is determined by the size of the modulus P and must be between 512 and 2048 bits if not executing in FIPS mode, and must be 2048 bits if executing in FIPS mode. The private value X will be less than Q-1 if Q is present in the key parameters, otherwise the private value X will be less than P-1.

Multiple Diffie-Hellman Key Agreement key pairs can share the same group parameters (P and G). This is useful when generating multiple keys of the same type since it is very time-consuming to compute values for P and G. In addition, the Diffie-Hellman key agreement algorithm requires both parties to use the same group parameters when computing the shared secret value.

gsk_generate_key_pair()

Generates a public/private key pair.

Format

```
#include <gskcms.h>

gsk_status gsk_generate_key_pair (
                          x509_algorithm_type          key_algorithm,
                          int                          key_size,
                          gsk_buffer *                 key_params,
                          x509_public_key_info *       public_key,
                          pkcs_private_key_info *      private_key,
                          gsk_buffer *                 key_identifier)
```

Parameters

key_algorithm
Specifies the key algorithm.

key_size
Specifies the key size in bits.

key_params
Specifies the key parameters as an ASN.1-encoded sequence. Specify NULL for this parameter if the key algorithm does not require any parameters.

public_key
Returns the generated public key. The application should call the **gsk_free_public_key_info()** routine to release the public key when it is no longer needed.

private_key
Returns the generated private key. The application should call the **gsk_free_private_key_info()** routine to release the private key when it is no longer needed.

key_identifier
Returns the key identifier for the generated public key. The application should call the **gsk_free_buffer()** routine to release the key identifier when it is no longer needed. Specify NULL for this parameter if the key identifier is not needed.

Results

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

[CMSERR_ALG_NOT_SUPPORTED]
The key algorithm is not supported.

[CMSERR_BAD_DH_PARAMS]
The Diffie-Hellman group parameters are not valid.

[CMSERR_BAD_DSA_PARAMS]
The DSS parameters are not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_FIPS_KEY_PAIR_CONSISTENCY]
FIPS mode key generation failed pair-wise consistency check.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_generate_key_pair()** routine will generate a public/private key pair. The format of the public and private key values returned by the **gsk_generate_key_pair()** routine is defined in RFC 3280 (Internet 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 must be between 512 and 1024 bits if not executing in FIPS mode, and must be 1024 bits if executing in FIPS mode, and will be rounded up to a multiple of 64 bits if necessary. The key parameters are the prime P, the subprime Q, and the base G. The requested key size must be the same as the size of the prime P. The **gsk_generate_key_parameters()** routine can be used to generate the key parameters.

- **x509_alg_dhPublicNumber - Diffie-Hellman Key Exchange - {1.2.840.10046.2.1}**

  The key size must be between 512 and 2048 bits if not executing in FIPS mode, and must be 2048 bits if executing in FIPS mode, and will be rounded up to a multiple of 64 bits if necessary. The key parameters are the prime P, the base G, the subprime Q, and the subgroup factor J. The requested key size must be the same as the size of the prime P. The **gsk_generate_key_parameters()** routine can be used to generate the key parameters.

  The subprime Q and the subgroup factor J are optional key parameters. This allows the **gsk_generate_key_pair()** routine to accept key parameters generated in accordance with PKCS #3 (Diffie-Hellman Key Agreement Standard) as well as key parameters generated in accordance with RFC 2631 (Diffie-Hellman Key Agreement Method). The private value X will be less than Q-1 if Q is present in the key parameters, otherwise the private value X will be less than P-1.

  Multiple Digital Signature Standard keys or Diffie-Hellman Key Exchange keys can share the same group parameters (P, Q, and G). This is useful when generating multiple keys of the same type since it is very time-consuming to compute values for P, Q, and G. In addition, the Diffie-Hellman key agreement algorithm requires both parties to use the same group parameters when computing the secret value.

gsk_generate_key_parameters()

Generates a key parameters.

Format

```
#include <gskcms.h>

gsk_status gsk_generate_key_parameters (
                                        x509_algorithm_type          key_algorithm,
                                        int                          key_size,
                                        gsk_buffer *                 key_params )
```

Parameters

key_algorithm
Specifies the key algorithm.

key_size
Specifies the key size in bits.

key_params
Specifies the key parameters as an ASN.1-encoded sequence. The application should call the **gsk_free_buffer()** routine to release the key parameters when they are no longer needed.

Results

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

[CMSERR_ALG_NOT_SUPPORTED]
The key algorithm is not supported.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_generate_key_parameters()** routine will generate key parameters that can then be used with the gsk_generate_key_pair() routine to generate one or more public/private key pairs.

These key algorithms are supported:

- **x509_alg_idDsa - Digital Signature Standard - {1.2.840.10040.4.1}**

  The key size must be between 512 and 1024 bits if not executing in FIPS mode, and must be 1024 bits if executing in FIPS mode, and will be rounded up to a multiple of 64 bits if necessary. The generated ASN.1 sequence will consist of the prime P, the subprime Q, and the base G. Refer to FIPS 186-2 (Digital Signature Standard) for more information on the generation of the key parameters.

- **x509_alg_dhPublicNumber - Diffie-Hellman Key Exchange - {1.2.840.10046.2.1}**

  The key size must be between 512 and 2048 bits if not executing in FIPS mode, and must be 2048 bits if executing in FIPS mode, and will be rounded up to a multiple of 64 bits if necessary. The generated ASN.1 sequence will consist of the prime P, the base G, the subprime Q, and the subgroup factor J. Refer to RFC 2631 (Diffie-Hellman Key Agreement Method) for more information on the generation of the key parameters and RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile) for more information on the ASN.1 encoding.

  Multiple Digital Signature Standard keys or Diffie-Hellman Key Exchange keys can share the same group parameters (P, Q, and G). This is useful when generating multiple keys of the same type since it is very time-consuming to compute values for P, Q, and G. In addition, the Diffie-Hellman key

agreement algorithm requires both parties to use the same group parameters when computing the secret value (an SSL client will generate temporary Diffie-Hellman values if the group parameters in the client certificate are not the same as the group parameters in the server certificate).

gsk_generate_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()** buffer() routine to release the secret value when it is no longer needed.

Results

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

[CMSERR_BAD_DH_PARAMS]
The Diffie-Hellman group parameters are not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_generate_secret()** routine will generate the Diffie-Hellman shared secret value as defined in PKCS #3 (Diffie-Hellman Key Agreement Standard) and RFC 2631 (Diffie-Hellman Key Agreement Method). The required key parameters P and G and the optional key parameters Q and J are supplied as an ASN.1-encoded sequence as defined in either PKCS #3 or RFC 3280 (Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile). The return value will be the binary value for Z. The key size is determined by the size of the modulus P and must be between 512 and 2048 bits if not executing in FIPS mode, or it must be 2048 bits if in FIPS mode.

gsk_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_EXPIRED]
The certificate is expired.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_USAGE]
The issuer certificate does not allow signing certificates.

[CMSERR_ISSUER_NOT_CA]
The certificate issuer is not a certification authority.

[CMSERR_ISSUER_NOT_FOUND]
The issuer certificate is not in the key database.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NOT_YET_VALID]
The certificate is not yet valid.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_import_certificate()** routine imports an X.509 certificate and creates a new database record. An error will be returned if the certificate is already in the database. The database must be a key database and must be open for update in order to import certificates.

The supplied stream can represent either the ASN.1 DER encoding for the certificate or the Cryptographic Message Syntax (PKCS #7) encoding for the certificate. This can be either the binary value or the Base64 encoding of the binary value. A Base64 encoded stream must be in the local code page and must include the encoding header and footer lines.

The **gsk_import_certificate()** routine imports a single certificate. If the PKCS #7 message contains multiple certificates, only the first certificate and its certificate chain will be imported. The certificate subject name will be used as the label for certificates added from the certification chain. A chain certificate will not be added to the database if the label is not unique or if the certificate is already in the database.

A unique record identifier is assigned when the record is added to the database. The certificate signature will be verified using the certificate of the issuer. An error will be returned if the issuer certificate is not already in the key database and is not contained in the PKCS #7 message stream. The certificate will be marked as a trusted certificate when it is added to the database.

The record label is used as a friendly name for the database entry. It can be any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string.

An existing certificate can be replaced by specifying the label of the existing certificate. The issuer name, subject name, and subject public key in the new certificate must be the same as the existing certificate. If the existing certificate has a private key, the private key is not changed when the certificate is replaced.

The database file is updated as part of the **gsk_import_certificate()** processing. A temporary database file is created using the same name as the database file with ″.new″ appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.

gsk_import_key()

Imports a certificate and associated private key.

Format

```
#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_EXPIRED]
The certificate is expired.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_USAGE]
The issuer certificate does not allow signing certificates.

[CMSERR_ISSUER_NOT_CA]
The certificate issuer is not a certification authority.

[CMSERR_ISSUER_NOT_FOUND]
The issuer certificate is not in the key database.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NOT_YET_VALID]
The certificate is not yet valid.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_import_key()** routine imports an X.509 certificate and its private key and creates a new database record. An error will be returned if the database already contains the certificate. The database must be open for update in order to import certificates.

The certificate and key must have been encoded according to the Personal Information Exchange Syntax (PKCS #12). If executing in FIPS mode, the only supported encryption is the x509_alg_pbeWithSha1And3DesCbc algorithm. The supplied stream can be the binary ASN.1 sequence or the Base64 encoding of the ASN.1 sequence. A Base64 encoded stream is assumed to be in the local code page and must include the encoding header and footer lines.

The record label is used as a friendly name for the database entry. It can be any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string. An error will be returned if the certificate already exists in the key database or the record label is not unique.

A unique record identifier is assigned when the record is added to the database. The certificate signature will be verified using the certificate of the issuer. The certificate will be marked as a trusted certificate when it is added to the database.

Each certificate in the certification chain will be imported if it is present in the import file. The certificate subject name will be used as the label for certificates added from the certification chain. A chain certificate will not be added to the database if the label is not unique or if the certificate is already in the database.

The database file is updated as part of the **gsk_import_key()** processing. A temporary database file is created using the same name as the database file with ″.new″ appended to the name. The database file is

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_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 (Password-based Encryption) and PKCS #12 (Personal Information Exchange). The selected algorithm determines how the key is derived from the password.

These password-based encryption algorithms are supported. The strong encryption algorithms may not be available depending upon government export regulations.

- **x509_alg_pbeWithMd2AndDesCbc - 56-bit DES encryption with MD2 digest - {1.2.840.113549.1.5.1}**
- **x509_alg_pbeWithMd5AndDesCbc - 56-bit DES encryption with MD5 digest - {1.2.840.113549.1.5.3}**
- **x509_alg_pbeWithSha1AndDesCbc - 56-bit DES encryption with SHA-1 digest - {1.2.840.113549.1.5.10}**
- **x509_alg_pbeWithMd2AndRc2Cbc - 64-bit RC2 encryption with MD2 digest - {1.2.840.113549.1.5.4}**
- **x509_alg_pbeWithMd5AndRc2Cbc - 64-bit RC2 encryption with MD5 digest - {1.2.840.113549.1.5.6}**
- **x509_alg_pbeWithSha1AndRc2Cbc - 64-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.5.11}**
- **x509_alg_pbeWithSha1And40BitRc2Cbc - 40-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.6}**
- **x509_alg_pbeWithSha1And128BitRc2Cbc - 128-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.5}**
- **x509_alg_pbeWithSha1And40BitRc4 - 40-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.2}**
- **x509_alg_pbeWithSha1And128BitRc4 - 128-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.1}**
- **x509_alg_pbeWithSha1And3DesCbc - 168-bit 3DES encryption with SHA-1 digest - {1.2.840.113549.1.12.1.3}**

gsk_make_encrypted_data_msg()

Creates a PKCS #7 EncryptedData message from application data.

Format

```
#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 (Password-based Encryption) and PKCS #12 (Personal Information Exchange). The selected algorithm determines how the key is derived from the password.

These password-based encryption algorithms are supported. The strong encryption algorithms may not be available depending upon government export regulations.

- **x509_alg_pbeWithMd2AndDesCbc - 56-bit DES encryption with MD2 digest - {1.2.840.113549.1.5.1}**
- **x509_alg_pbeWithMd5AndDesCbc - 56-bit DES encryption with MD5 digest - {1.2.840.113549.1.5.3}**
- **x509_alg_pbeWithSha1AndDesCbc - 56-bit DES encryption with SHA-1 digest - {1.2.840.113549.1.5.10}**
- **x509_alg_pbeWithMd2AndRc2Cbc - 64-bit RC2 encryption with MD2 digest - {1.2.840.113549.1.5.4}**
- **x509_alg_pbeWithMd5AndRc2Cbc - 64-bit RC2 encryption with MD5 digest - {1.2.840.113549.1.5.6}**
- **x509_alg_pbeWithSha1AndRc2Cbc - 64-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.5.11}**
- **x509_alg_pbeWithSha1And40BitRc2Cbc - 40-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.6}**
- **x509_alg_pbeWithSha1And128BitRc2Cbc - 128-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.5}**
- **x509_alg_pbeWithSha1And40BitRc4 - 40-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.2}**
- **x509_alg_pbeWithSha1And128BitRc4 - 128-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.1}**
- **x509_alg_pbeWithSha1And3DesCbc - 168-bit 3DES encryption with SHA-1 digest - {1.2.840.113549.1.12.1.3}**

gsk_make_enveloped_data_content()

Create PKCS #7 EnvelopedData content information

Format

```
#include <gskcms.h>

gsk_status gsk_make_enveloped_data_content (
                          int                     version,
                          pkcs_session_key *      session_key,
                          pkcs_certificates *     recipient_certificates,
                          pkcs_content_info *     content_data,
                          pkcs_content_info *     content_info)
```

Parameters

version
Specifies the PKCS #7 EnvelopedData version number. Specify 0 to create EnvelopedData content as described in PKCS #7 Version 1.5. Specify 1 to create EnvelopedData content as described in PKCS #7 Version 1.6.

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_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_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}**

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 keyEnchiperment.

version
Specifies the PKCS #7 EnvelopedData version number. Specify 0 to create EnvelopedData content as described in PKCS #7 Version 1.5. Specify 1 to create EnvelopedData content as described in PKCS #7 Version 1.6.

session key
Specifies the session encryption key as follows:

- The *encryptionType* field specifies the encryption algorithm.
- The *encryptionKey.length* field specifies the encryption key length in bytes.
- The *encryptionKey.data* field specifies the address of the encryption key. A new key will be generated and returned in this parameter if the key address is NULL. If a new key is generated, the application should call the **gsk_free_buffer()** routine to release the key when it is no longer needed. Note that the *encryptionType* and *encryptionKey.length* fields must be set by the application even when a new session key is to be generated.

recipient_certificates
Specifies the certificates for the message recipients. There must be at least one recipient.

content_data
Specifies the EnvelopedData content. This must be one of the content information types defined in PKCS #7.

content_info
Returns the EnvelopedData content information. The application should call the **gsk_free_content_info()** routine to release the content information when it is no longer needed.

Results

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

[CMSERR_ALG_NOT_AVAILABLE]
The encryption algorithm is not available

[CMSERR_ALG_NOT_SUPPORTED]
The encryption algorithm is not supported

[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_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_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 encryption. Currently, only RSA public keys support data encryption. In addition, if *option_flag* specifies that key encipherment is to be enforced, then the certificate key usage must allow key encipherment.

These encryption algorithms are supported. Strong encryption may not be available depending upon government export regulations.

- **x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - Key lengths 5 and 16 - {1.2.840.113549.3.2}**
- **x509_alg_rc4 - 40-bit and 128-bit RC4 - Key lengths 5 and 16 - {1.2.840.113549.3.4}**
- **x509_alg_desCbcPad - 56-bit DES - Key length 8 - {1.3.14.3.2.7}**
- **x509_alg_desEde3CbcPad - 168-bit 3DES - Key length 24 - {1.2.840.113549.3.7}**

When executing in FIPS mode, encryption algorithms x509_alg_rc2CbcPad, x509_alg_rc4 and x509_alg_desCbcPad are not supported.

gsk_make_enveloped_data_msg()

Creates a PKCS #7 EnvelopedData message from application data.

Format

```
#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 version number. Specify 0 to create an EnvelopedData message as described in PKCS #7 Version 1.5. Specify 1 to create an EnvelopedData message as described in PKCS #7 Version 1.6.

session_key
Specifies the session encryption key as follows:

- The *encryptionType* field specifies the encryption algorithm.
- The *encryptionKey.length* field specifies the encryption key length in bytes.
- The *encryptionKey.data* field specifies the address of the encryption key. A new key will be generated and returned in this parameter if the key address is NULL. If a new key is generated, the application should call the **gsk_free_buffer()** routine to release the key when it is no longer needed. Note that the *encryptionType* and *encryptionKey.length* fields must be set by the application even when a new session key is to be generated.

recipient_certificates
Specifies the certificates for the message recipients. There must be at least one recipient.

data
Specifies the application data for the EnvelopedData message.

stream
Returns the ASN.1 DER-encoded stream. The application should call the **gsk_free_buffer()** routine to release the stream when it is no longer needed.

Results

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

[CMSERR_ALG_NOT_AVAILABLE]
The encryption algorithm is not available.

[CMSERR_ALG_NOT_SUPPORTED]
The encryption algorithm is not supported.

[CMSERR_BAD_KEY_SIZE]
The encryption key size is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported.

[CMSERR_INCORRECT_KEY_USAGE]
A recipient certificate does not allow key encipherment.

[CMSERR_KEY_MISMATCH]
A recipient public key does not support data encryption.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECIPIENT_NOT_FOUND]
No recipient certificates provided.

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid.

Usage

The **gsk_make_enveloped_data_msg()** routine creates a PKCS #7 (Cryptographic Message Syntax) EnvelopedData message and returns the ASN.1 DER-encoded ContentInfo sequence. The enveloped data content type will be Data. The **gsk_read_enveloped_data_msg()** routine can be used to extract the application data from the stream. No validity checking is performed on the recipient certificates. It is assumed that the application has already validated the recipient certificates.

Calling the **gsk_make_enveloped_data_msg()** routine is equivalent to calling the **gsk_make_data_content()** routine, the **gsk_make_enveloped_data_content()** routine, and the **gsk_make_content_msg()** routine.

The session key is used to encrypt the message content. A new session key is generated and returned to the application if no key is provided. For each recipient, the session key is encrypted with the recipient's public key and stored in the EnvelopedData message. This means the public key algorithm must support data encryption. Currently, only RSA public keys support data encryption. In addition, the certificate key usage must allow key encipherment.

These encryption algorithms are supported. Strong encryption may not be available depending upon government export regulations.

- **x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - Key lengths 5 and 16 - {1.2.840.113549.3.2}**
- **x509_alg_rc4 - 40-bit and 128-bit RC4 - Key lengths 5 and 16 - {1.2.840.113549.3.4}**
- **x509_alg_desCbcPad - 56-bit DES - Key length 8 - {1.3.14.3.2.7}**
- **x509_alg_desEde3CbcPad - 168-bit 3DES - Key length 24 - {1.2.840.113549.3.7}**

When executing in FIPS mode, encryption algorithms x509_alg_rc2CbcPad, x509_alg_rc4 and x509_alg_desCbcPad are not supported.

gsk_make_enveloped_data_msg_extended()

Creates a PKCS #7 EnvelopedData message from application data.

Format

```
#include <gskcms.h>

gsk_status gsk_make_enveloped_data_msg_extended (
                                    gsk_process_option          option_flag,
                                    int                         version,
                                    pkcs_session_key *          session_key,
                                    pkcs_certificates *         recipient_certificates,
                                    gsk_buffer *                data,
                                    gsk_buffer *                stream)
```

Parameters

option_flag
Specifies process options to customize process behavior:

- Enforce recipient certificate has key 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 version number. Specify 0 to create an EnvelopedData message as described in PKCS #7 Version 1.5. Specify 1 to create an EnvelopedData message as described in PKCS #7 Version 1.6.

session_key
Specifies the session encryption key as follows:

- The *encryptionType* field specifies the encryption algorithm.
- The *encryptionKey.length* field specifies the encryption key length in bytes.
- The *encryptionKey.data* field specifies the address of the encryption key. A new key will be generated and returned in this parameter if the key address is NULL. If a new key is generated, the application should call the **gsk_free_buffer()** routine to release the key when it is no longer needed. Note that the *encryptionType* and *encryptionKey.length* fields must be set by the application even when a new session key is to be generated.

recipient_certificates
Specifies the certificates for the message recipients. There must be at least one recipient.

data
Specifies the application data for the EnvelopedData message.

stream
Returns the ASN.1 DER-encoded stream. The application should call the **gsk_free_buffer()** routine to release the stream when it is no longer needed.

Results

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

[CMSERR_ALG_NOT_AVAILABLE]
The encryption algorithm is not available.

[CMSERR_ALG_NOT_SUPPORTED]
The encryption algorithm is not supported.

[CMSERR_BAD_KEY_SIZE]
The encryption key size is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported.

[CMSERR_INCORRECT_KEY_USAGE]
A recipient certificate does not allow key encipherment.

[CMSERR_KEY_MISMATCH]
A recipient public key does not support data encryption.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECIPIENT_NOT_FOUND]
No recipient certificates provided.

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid.

Usage

The **gsk_make_enveloped_data_msg_extended()** routine creates a PKCS #7 (Cryptographic Message Syntax) EnvelopedData message and returns the ASN.1 DER-encoded ContentInfo sequence. Processing is equivalent to **gsk_make_enveloped_data_msg()**, except that the recipient certificate key usage need not assert key encipherment. The enveloped data content type will be Data. The **gsk_read_enveloped_data_msg()** routine or the **gsk_read_enveloped_data_msg_extended()** routine can be used to extract the application data from the stream. No validity checking is performed on the recipient certificates. It is assumed that the application has already validated the recipient certificates.

Calling the **gsk_make_enveloped_data_msg_extended()** routine is equivalent to calling the **gsk_make_data_content()** routine, the **gsk_make_enveloped_data_content_extended()** routine, and the **gsk_make_content_msg()** routine.

The session key is used to encrypt the message content. A new session key is generated and returned to the application if no key is provided. For each recipient, the session key is encrypted with the recipient's public key and stored in the EnvelopedData message. This means the public key algorithm must support data encryption. Currently, only RSA public keys support data encryption. In addition, if option_flag specifies that key encipherment is to be enforced, then the certificate key usage must allow key encipherment.

These encryption algorithms are supported. Strong encryption may not be available depending upon government export regulations.

- **x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - Key lengths 5 and 16 - {1.2.840.113549.3.2}**
- **x509_alg_rc4 - 40-bit and 128-bit RC4 - Key lengths 5 and 16 - {1.2.840.113549.3.4}**
- **x509_alg_desCbcPad - 56-bit DES - Key length 8 - {1.3.14.3.2.7}**
- **x509_alg_desEde3CbcPad - 168-bit 3DES - Key length 24 - {1.2.840.113549.3.7}**

When executing in FIPS mode, encryption algorithms x509_alg_rc2CbcPad, x509_alg_rc4 and x509_alg_desCbcPad are not supported.

gsk_make_signed_data_content()

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 version number. Specify 0 to create SignedData content information as described in PKCS #7 Version 1.4, specify 1 to create SignedData content information as described in PKCS #7 Version 1.5, or specify 2 to create SignedData content information as described in PKCS #7 Version 1.6.

digest_algorithm
Specifies the digest algorithm.

include_certificates
Specify TRUE if the signer and certification authority certificates are to be included in the SignedData content information. Specify FALSE if the certificates are not to be included.

signer_certificates
Specifies the certificates and associated private keys for the message signers. There must be at least one signer.

ca_certificates
Specifies the certification authority certificates. Zero or more certification authority certificates can be included in the SignedData content information. This parameter is ignored if the include_certificates parameter is set to FALSE. NULL can be specified for this parameter if no CA certificates are to be included in the message.

content_data
Specifies the SignedData content. This must be one of the content information types defined in PKCS #7.

content_info
Returns the SignedData content information. The application should call the **gsk_free_content_info()** routine to release the content information when it is no longer needed.

Results

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

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported.

[CMSERR_DIGEST_KEY_MISMATCH]
The digest algorithm is not supported for the private key type.

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_SIGNER_NOT_FOUND]
No signer certificate provided or the certificate is not valid.

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid

Usage

The **gsk_make_signed_data_content()** routine creates PKCS #7 (Cryptographic Message Syntax) SignedData content information. The data content type must be one of the types defined by PKCS #7. The **gsk_read_signed_data_content()** routine can be used to extract the content data from the SignedData content information. The key usage for the signer certificates must allow digital signature. No validity checking will be performed on the signer certificates. It is assumed that the application has already validated the signer certificates.

A signature is included for each signer provided by the *signer_certificates* parameter. The X.509 certificates used to sign the message will be included in the SignedData content information if the *include_certificates* parameter is set to TRUE. The message receiver will need to provide the signer certificates if the *include_certificates* parameter is set to FALSE.

You can optionally include certification authority certificates in the SignedData content information. These certificate can then be used by the message receiver to validate the signer certificates.

These digest algorithms are supported:

x509_alg_md2Digest
MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

x509_alg_md5Digest
MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

x509_alg_sha1Digest
SHA-1 digest (RSA and DSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest
SHA-224 digest (RSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest
SHA-256 digest (RSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest
SHA-384 digest (RSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest
SHA-512 digest (RSA keys only) - {2.16.840.1.101.3.4.2.3}

| When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not
| supported.

gsk_make_signed_data_content_extended()

Creates PKCS #7 SignedData content information.

Format

```
#include <gskcms.h>

gsk_status gsk_make_signed_data_content_extended (
                                        gsk_process_option         option_flag,
                                        int                        version,
                                        x509_algorithm_type        digest_algorithm,
                                        gsk_boolean                include_certificates,
                                        pkcs_cert_keys *           signer_certificates,
                                        pkcs_certificates *        ca_certificates,
                                        pkcs_content_info *        content_data,
                                        gsk_attributes_signers *   attributes_signers,
                                        pkcs_content_info *        content_info,)
```

Parameters

option_flag
Specifies process options to customize process behavior.

- Enforce signing certificate has digital signing capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate digitalSignature.
- Don't allow zero-length content data

version
Specifies the PKCS #7 SignedData version number. Specify 0 to create SignedData content information as described in PKCS #7 Version 1.4, specify 1 to create SignedData content information as described in PKCS #7 Version 1.5, or specify 2 to create SignedData content information as described in PKCS #7 Version 1.6.

digest_algorithm
Specifies the digest algorithm.

include_certificates
Specify TRUE if the signer and certification authority certificates are to be included in the SignedData content information. Specify FALSE if the certificates are not to be included.

signer_certificates
Specifies the certificates and associated private keys for the message signers. There must be at least one signer.

ca_certificates
Specifies the certification authority certificates. Zero or more certification authority certificates can be included in the SignedData content information. This parameter is ignored if the include_certificates parameter is set to FALSE. NULL can be specified for this parameter if no CA certificates are to be included in the message.

content_data
Specifies the SignedData content. This must be one of the content information types defined in PKCS #7.

attributes_signers
Specifies the authenticated attributes per signer to be added to the message. Specify NULL for this parameter if there are no authenticated attributes to be included in the message. If specified, the set of authenticated attributes must NOT include content-type or message-digest authenticated attributes as these are automatically provided by **gsk_make_signed_data_content_extended()**. If the set of authenticated attributes includes signing-time, then this will override the signing-time attribute

generated by **gsk_make_signed_data_content_extended()**. The *digestAlgorithm* field within each *gsk_attributes_signer* structure is ignored - the digest algorithm is specified by the *digest_algorithm* parameter.

content_info
Returns the SignedData content information. The application should call the **gsk_free_content_info()** routine to release the content information when it is no longer needed.

Results

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

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported.

[CMSERR_DIGEST_KEY_MISMATCH]
The digest algorithm is not supported for the private key type.

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_SIGNER_NOT_FOUND]
No signer certificate provided or the certificate is not valid.

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid

[CMSERR_CONTENTTYPE_NOT_ALLOWED]
The content-type authenticated attribute is not allowed in *attributes_signers*.

[CMSERR_MESSAGEDIGEST_NOT_ALLOWED]
The message-digest authenticated attribute is not allowed in *attributes_signers*

Usage

The **gsk_make_signed_data_content_extended()** routine creates PKCS #7 (Cryptographic Message Syntax) SignedData content information. The data content type must be one of the types defined by PKCS #7. Processing is similar to **gsk_make_signed_data_content()** except for the presence of the *option_flag* and *authenticated_attributes* parameters. The **gsk_read_signed_data_content()** routine or the **gsk_read_signed_data_content_extended()** routine can be used to extract the content data from the SignedData content information. The key usage for the signer certificates can be optionally specified as to whether digital signature must be allowed. No validity checking is performed on the signer certificates. It is assumed that the application has already validated the signer certificates.

A signature is included for each signer provided by the *signer_certificates* parameter. The X.509 certificates used to sign the message will be included in the SignedData content information if the *include_certificates* parameter is set to TRUE. The message receiver will need to provide the signer certificates if the *include_certificates* parameter is set to FALSE.

You can optionally include certification authority certificates in the SignedData content information. These certificates can then be used by the message receiver to validate the signer certificates.

These digest algorithms are supported:

x509_alg_md2Digest
MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

x509_alg_md5Digest
MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

x509_alg_sha1Digest
SHA-1 digest (RSA and DSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest
SHA-224 digest (RSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest
SHA-256 digest (RSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest
SHA-384 digest (RSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest
SHA-512 digest (RSA keys only) - {2.16.840.1.101.3.4.2.3}

If authenticated attributes are provided via the *attributes_signers* parameter, then signing certificates for all signers represented within the gsk_attributes_signers structure must be provided via the *signer_certificates* parameter.

When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not supported.

gsk_make_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_certificates *     ca_certificates,
                                     gsk_buffer *            data,
                                     gsk_buffer *            stream)
```

Parameters

version
Specifies the PKCS #7 SignedData version number. Specify 0 to create a SignedData message as described in PKCS #7 Version 1.4, specify 1 to create a SignedData message as described in PKCS #7 Version 1.5, or specify 2 to create a SignedData message as described in PKCS #7 Version 1.6.

digest_algorithm
Specifies the digest algorithm.

include_certificates
Specify TRUE if the signer and certification authority certificates are to be included in the SignedData message. Specify FALSE if the certificates are not to be included.

signer_certificates
Specifies the certificates and associated private keys for the message signers. There must be at least one signer.

ca_certificates
Specifies the certification authority certificates. Zero or more certification authority certificates can be included in the SignedData message. This parameter is ignored if the include_certificates parameter is set to FALSE. NULL can be specified for this parameter if no CA certificates are to be included in the message.

data
Specifies the application data for the SignedData message.

stream
Returns the ASN.1 DER-encoded stream. The application should call the **gsk_free_buffer()** routine to release the stream when it is no longer needed.

Results

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

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported.

[CMSERR_DIGEST_KEY_MISMATCH]
The digest algorithm is not supported for the private key type.

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_SIGNER_NOT_FOUND]
No signer certificate provided or the certificate is not valid.

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid.

Usage

The **gsk_make_signed_data_msg()** routine creates a PKCS #7 (Cryptographic Message Syntax) SignedData message and returns the ASN.1 DER-encoded ContentInfo sequence. The signed data content type will be Data. The **gsk_read_signed_data_msg()** routine can be used to extract the application data from the stream. The key usage for the signer certificates must allow digital signature. No validity checking will be performed on the signer certificates. It is assumed that the application has already validated the signer certificates.

Calling the **gsk_make_signed_data_msg()** routine is equivalent to calling the **gsk_make_data_content()** routine, the **gsk_make_signed_data_content()** routine, and the **gsk_make_content_msg()** routine.

A signature is included for each signer provided by the *signer_certificates* parameter. The X.509 certificates used to sign the message will be included in the SignedData message if the *include_certificates* parameter is set to TRUE. The message receiver will need to provide the signer certificates if the *include_certificates* parameter is set to FALSE.

You can optionally include certification authority certificates in the SignedData message. These certificates can then be used by the message receiver to validate the signer certificates.

These digest algorithms are supported:

x509_alg_md2Digest
MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

x509_alg_md5Digest
MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

x509_alg_sha1Digest
SHA-1 digest (RSA and DSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest
SHA-224 digest (RSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest
SHA-256 digest (RSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest
SHA-384 digest (RSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest
SHA-512 digest (RSA keys only) - {2.16.840.1.101.3.4.2.3}

When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not supported.

gsk_make_signed_data_msg_extended()

Creates a PKCS #7 SignedData message from application data.

Format

```
#include <gskcms.h>

gsk_status gsk_make_signed_data_msg_extended (
                                    gsk_process_option        option_flag,
                                    int                       version,
                                    x509_algorithm_type       digest_algorithm,
                                    gsk_boolean               include_certificates,
                                    pkcs_cert_keys *          signer_certificates,
                                    pkcs_certificates *       ca_certificates,
                                    gsk_buffer *              data,
                                    gsk_attributes_signers *  attributes_signers,
                                    gsk_buffer *              stream)
```

Parameters

option_flag
Specifies process options to customize process behavior.

- Enforce signing certificate has digital signing capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate digitalSignature.
- Don't allow zero-length content data

version
Specifies the PKCS #7 SignedData version number. Specify 0 to create a SignedData message as described in PKCS #7 Version 1.4, specify 1 to create a SignedData message as described in PKCS #7 Version 1.5, or specify 2 to create a SignedData message as described in PKCS #7 Version 1.6.

digest_algorithm
Specifies the digest algorithm.

include_certificates
Specify TRUE if the signer and certification authority certificates are to be included in the SignedData message. Specify FALSE if the certificates are not to be included.

signer_certificates
Specifies the certificates and associated private keys for the message signers. There must be at least one signer.

ca_certificates
Specifies the certification authority certificates. Zero or more certification authority certificates can be included in the SignedData message. This parameter is ignored if the include_certificates parameter is set to FALSE. NULL can be specified for this parameter if no CA certificates are to be included in the message.

data
Specifies the application data for the SignedData message.

attributes_signers
Specifies the authenticated attributes per signer to be added to the message. Specify NULL for this parameter if there are no authenticated attributes to be included in the message. If specified, then the set of authenticated attributes must NOT include content-type or message-digest authenticated attributes as these are automatically provided by **gsk_make_signed_data_msg_extended()**. If the set of authenticated attributes includes signing-time, then this will override the signing-time attribute generated by **gsk_make_signed_data_msg_extended()**. The *digest_algorithm* field within each *gsk_attributes_signer* structure is ignored - the digest algorithm is specified by the *digest_algorithm* parameter.

stream
Returns the ASN.1 DER-encoded stream. The application should call the **gsk_free_buffer()** routine to release the stream when it is no longer needed.

Results

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

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported.

[CMSERR_DIGEST_KEY_MISMATCH]
The digest algorithm is not supported for the private key type.

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_SIGNER_NOT_FOUND]
No signer certificate provided or the certificate is not valid.

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid.

[CMSERR_CONTENTTYPE_NOT_ALLOWED]
The content-type authenticated attribute is not allowed in *attributes_signers*.

[CMSERR_MESSAGEDIGEST_NOT_ALLOWED]
The message-digest authenticated attribute is not allowed in *attributes_signers*

Usage

The **gsk_make_signed_data_msg_extended()** routine creates a PKCS #7 (Cryptographic Message Syntax) SignedData message and returns the ASN.1 DER-encoded ContentInfo sequence. The signed data content type will be Data. The **gsk_read_signed_data_msg()** or the **gsk_read_signed_data_msg_extended()** routine can be used to extract the application data from the stream. The key usage for the signer certificates can be optionally specified as to whether digital signature must be allowed. No validity checking will be performed on the signer certificates. It is assumed that the application has already validated the signer certificates.

Calling the **gsk_make_signed_data_msg_extended()** routine is equivalent to calling the **gsk_make_data_content()** routine, the **gsk_make_signed_data_content_extended()** routine, and the **gsk_make_content_msg()** routine.

A signature is included for each signer provided by the *signer_certificates* parameter. The X.509 certificates used to sign the message will be included in the SignedData message if the *include_certificates* parameter is set to TRUE. The message receiver will need to provide the signer certificates if the *include_certificates* parameter is set to FALSE.

You can optionally include certification authority certificates in the SignedData message. These certificates can then be used by the message receiver to validate the signer certificates.

These digest algorithms are supported:

x509_alg_md2Digest
MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

x509_alg_md5Digest
MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

x509_alg_sha1Digest
SHA-1 digest (RSA and DSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest
SHA-224 digest (RSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest
SHA-256 digest (RSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest
SHA-384 digest (RSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest
SHA-512 digest (RSA keys only) - {2.16.840.1.101.3.4.2.3}

If authenticated attributes are provided via the *attributes_signers* parameter, then signing certificates for all signers represented within the gsk_attributes_signers structure must be provided via the *signer_certificates* parameter.

When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not supported.

gsk_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 as long as the same database handle is used by all of the threads.

A FIPS database file may only be opened for update while executing in FIPS mode. A FIPS database may be opened read-only while executing in non-FIPS mode. A non-FIPS database file cannot be opened for read or update while executing in FIPS mode.

gsk_open_database_using_stash_file()

Opens a key or request database using a stash file for the database password.

Format

```
#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 as long as the same database handle is used by all of the threads.

A FIPS database file may only be opened for update while executing in FIPS mode. A FIPS database may be opened read-only while executing in non-FIPS mode. A non-FIPS database file cannot be opened for read or update while executing in FIPS mode.

gsk_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 78) prior to calling this function.

Format

```
#include <gskcms.h>

gsk_status gsk_perform_kat ()
```

Results

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

[CMSERR_API_NOT_SUPPORTED]
The API is not supported in non-FIPS mode.

[CMSERR_KATPW_FAILED]
A known answer test has failed. This is a severe error and the application should terminate.

Usage

The **gsk_perform_kat()** routine can be used whenever an application, in order to meet security requirements, needs to check the correctness of cryptographic algorithms that are part of the product. The routine performs Known Answer Tests on the following cryptographic algorithms:

- AES 128-bit and AES 256-bit encryption and decryption
- Triple-DES encryption and decryption
- RSA signature generation/verification and encryption/decryption
- DSA signature generation and verification
- SHA Digest Algorithms: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 and HMAC-SHA-1
- Random number generation

If an error is encountered during testing, the **gsk_perform_kat()** routine will terminate and return the appropriate error code.

The **gsk_perform_kat()** routine will test software or hardware cryptographic algorithms depending on the value of the GSK_HW_CRYPTO environment variable at the time the CMS DLL (GSKCMS31 or GSKCMS64) is loaded.

gsk_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.

The decryption key is derived from the password as described in PKCS #5 (Password-based Encryption) and PKCS #12 (Personal Information Exchange). The selected algorithm determines how the key is derived from the password.

These password-based encryption algorithms are supported. The strong encryption algorithms may not be available depending upon government export regulations.

- **x509_alg_pbeWithMd2AndDesCbc - 56-bit DES encryption with MD2 digest - {1.2.840.113549.1.5.1}**
- **x509_alg_pbeWithMd5AndDesCbc - 56-bit DES encryption with MD5 digest - {1.2.840.113549.1.5.3}**
- **x509_alg_pbeWithSha1AndDesCbc - 56-bit DES encryption with SHA-1 digest - {1.2.840.113549.1.5.10}**
- **x509_alg_pbeWithMd2AndRc2Cbc - 64-bit RC2 encryption with MD2 digest - {1.2.840.113549.1.5.4}**
- **x509_alg_pbeWithMd5AndRc2Cbc - 64-bit RC2 encryption with MD5 digest - {1.2.840.113549.1.5.6}**
- **x509_alg_pbeWithSha1AndRc2Cbc - 64-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.5.11}**
- **x509_alg_pbeWithSha1And40BitRc2Cbc - 40-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.6}**
- **x509_alg_pbeWithSha1And128BitRc2Cbc - 128-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.5}**
- **x509_alg_pbeWithSha1And40BitRc4 - 40-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.2}**
- **x509_alg_pbeWithSha1And128BitRc4 - 128-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.1}**
- **x509_alg_pbeWithSha1And3DesCbc - 168-bit 3DES encryption with SHA-1 digest - {1.2.840.113549.1.12.1.3}**

gsk_read_encrypted_data_msg()

Processes a PKCS #7 EncryptedData message.

Format

```
#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.

Calling the **gsk_read_encrypted_data_msg()** routine is equivalent to calling the **gsk_read_content_msg()** routine, the **gsk_read_encrypted_data_content()** routine, and the **gsk_read_data_content()** routine.

The decryption key is derived from the password as described in PKCS #5 (Password-based Encryption) and PKCS #12 (Personal Information Exchange). The selected algorithm determines how the key is derived from the password.

These password-based encryption algorithms are supported. The strong encryption algorithms may not be available depending upon government export regulations.

- **x509_alg_pbeWithMd2AndDesCbc - 56-bit DES encryption with MD2 digest - {1.2.840.113549.1.5.1}**
- **x509_alg_pbeWithMd5AndDesCbc - 56-bit DES encryption with MD5 digest - {1.2.840.113549.1.5.3}**
- **x509_alg_pbeWithSha1AndDesCbc - 56-bit DES encryption with SHA-1 digest - {1.2.840.113549.1.5.10}**
- **x509_alg_pbeWithMd2AndRc2Cbc - 64-bit RC2 encryption with MD2 digest - {1.2.840.113549.1.5.4}**
- **x509_alg_pbeWithMd5AndRc2Cbc - 64-bit RC2 encryption with MD5 digest - {1.2.840.113549.1.5.6}**
- **x509_alg_pbeWithSha1AndRc2Cbc - 64-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.5.11}**
- **x509_alg_pbeWithSha1And40BitRc2Cbc - 40-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.6}**
- **x509_alg_pbeWithSha1And128BitRc2Cbc - 128-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.5}**
- **x509_alg_pbeWithSha1And40BitRc4 - 40-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.2}**
- **x509_alg_pbeWithSha1And128BitRc4 - 128-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.1}**
- **x509_alg_pbeWithSha1And3DesCbc - 168-bit 3DES encryption with SHA-1 digest - {1.2.840.113549.1.12.1.3}**

gsk_read_enveloped_data_content()

Processes PKCS #7 EnvelopedData content information.

Format

```
#include <gskcms.h>

gsk_status gsk_read_enveloped_data_content (
                                        pkcs_cert_keys *          recipient_keys,
                                        pkcs_content_info *       content_info,
                                        x509_algorithm_type *     encryption_algorithm,
                                        gsk_size *                key_size,
                                        pkcs_content_info *       content_data)
```

Parameters

recipient_keys
Specifies one or more certificates and associated private keys.

content_info
Specifies the content information to be processed.

encryption_algorithm
Returns the encryption algorithm used to encrypt the message content.

key_size
Returns the encryption key size in bytes.

content_data
Returns the EnvelopedData content data. The application should call the **gsk_free_content_info()** routine to release the content information when it is no longer needed.

Results

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

[CMSERR_ALG_NOT_AVAILABLE]
The encryption algorithm is not available.

[CMSERR_ALG_NOT_SUPPORTED]
The encryption algorithm is not supported.

[CMSERR_BAD_KEY_SIZE]
The encryption key size is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The message content type is not EnvelopedData or the content of the EnvelopedData message is not supported.

[CMSERR_INCORRECT_KEY_USAGE]
The recipient certificate does not allow key encipherment.

[CMSERR_KEY_MISMATCH]
A recipient private key does not support data decryption.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECIPIENT_NOT_FOUND]
No matching recipient certificate provided.

Usage

The **gsk_read_enveloped_data_content()** routine processes PKCS #7 (Cryptographic Message Syntax) EnvelopedData content information created by the **gsk_make_enveloped_data_content()** routine.

The *recipient_keys* parameter supplies one or more recipient certificates and associated private keys. The **gsk_read_enveloped_data_content()** routine will search for a certificate matching one of the message recipients. The private key will be used to decrypt the session key and the session key will then be used to decrypt the enveloped data. The certificate key usage must allow key encipherment.

No certificate validation is performed by the **gsk_read_enveloped_data_content()** routine. It is assumed that the application has already validated the recipient certificates.

These encryption algorithms are supported. Strong encryption may not be available depending upon government export regulations.

- **x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - {1.2.840.113549.3.2}**
- **x509_alg_rc4 - 40-bit and 128-bit RC4 - {1.2.840.113549.3.4}**
- **x509_alg_desCbcPad - 56-bit DES - {1.3.14.3.2.7}**
- **x509_alg_desEde3CbcPad - 168-bit 3DES - {1.2.840.113549.3.7}**

When executing in FIPS mode, encryption algorithms x509_alg_rc2CbcPad, x509_alg_rc4 and x509_alg_desCbcPad are not supported.

gsk_read_enveloped_data_content_extended()

Processes PKCS #7 EnvelopedData content information.

Format

```
#include <gskcms.h>

gsk_status gsk_read_enveloped_data_content_extended (
                                        gsk_process_option         option_flag
                                        pkcs_cert_keys *           recipient_keys,
                                        pkcs_content_info *        content_info,
                                        x509_algorithm_type *      encryption_algorithm,
                                        gsk_size *                 key_size,
                                        pkcs_content_info *        content_data)
```

Parameters

option_flag
Specifies process options to customize process behavior.

- Enforce recipient certificate has key encipherment capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate keyEncipherment.
- Enforce key parity when using DES or 3DES session keys.

recipient_keys
Specifies one or more certificates and associated private keys.

content_info
Specifies the content information to be processed.

encryption_algorithm
Returns the encryption algorithm used to encrypt the message content.

key_size
Returns the encryption key size in bytes.

content_data
Returns the EnvelopedData content data. The application should call the **gsk_free_content_info()** routine to release the content information when it is no longer needed.

Results

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

[CMSERR_ALG_NOT_AVAILABLE]
The encryption algorithm is not available.

[CMSERR_ALG_NOT_SUPPORTED]
The encryption algorithm is not supported.

[CMSERR_BAD_KEY_SIZE]
The encryption key size is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The message content type is not EnvelopedData or the content of the EnvelopedData message is not supported.

[CMSERR_INCORRECT_KEY_USAGE]
The recipient certificate does not allow key encipherment.

[CMSERR_KEY_MISMATCH]
A recipient private key does not support data decryption.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECIPIENT_NOT_FOUND]
No matching recipient certificate provided.

Usage

The **gsk_read_enveloped_data_content_extended()** routine processes PKCS #7 (Cryptographic Message Syntax) EnvelopedData content information created by the **gsk_make_enveloped_data_content()** routine or the **gsk_make_enveloped_data_content_extended()** routine. Processing is equivalent to **gsk_read_enveloped_data_content()**, except that the recipient certificate key usage need not assert key encipherment.

The *recipient_keys* parameter supplies one or more recipient certificates and associated private keys. The **gsk_read_enveloped_data_content_extended()** routine will search for a certificate matching one of the message recipients. The private key will be used to decrypt the session key and the session key will then be used to decrypt the enveloped data. In addition, if *option_flag* specifies that key encipherment is to be enforced, then the certificate key usage must allow key encipherment and session keys need not be odd parity.

No certificate validation is performed by the **gsk_read_enveloped_data_content_extended()** routine. It is assumed that the application has already validated the recipient certificates.

These encryption algorithms are supported. Strong encryption may not be available depending upon government export regulations.

- **x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - {1.2.840.113549.3.2}**
- **x509_alg_rc4 - 40-bit and 128-bit RC4 - {1.2.840.113549.3.4}**
- **x509_alg_desCbcPad - 56-bit DES - {1.3.14.3.2.7}**
- **x509_alg_desEde3CbcPad - 168-bit 3DES - {1.2.840.113549.3.7}**

When executing in FIPS mode, encryption algorithms x509_alg_rc2CbcPad, x509_alg_rc4 and x509_alg_desCbcPad are not supported.

gsk_read_enveloped_data_msg()

Processes a PKCS #7 EnvelopedData message.

Format

```
#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_BAD_ENCODING]
The message content type is not EnvelopedData or the message content is not Data.

[CMSERR_BAD_KEY_SIZE]
The encryption key size is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The message content type is not EnvelopedData or the content of the EnvelopedData message is not Data.

[CMSERR_INCORRECT_KEY_USAGE]
The recipient certificate does not allow key encipherment.

[CMSERR_KEY_MISMATCH]
A recipient private key does not support data decryption.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[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 may not be available depending upon government export regulations.

- **x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - {1.2.840.113549.3.2}**
- **x509_alg_rc4 - 40-bit and 128-bit RC4 - {1.2.840.113549.3.4}**
- **x509_alg_desCbcPad - 56-bit DES - {1.3.14.3.2.7}**
- **x509_alg_desEde3CbcPad - 168-bit 3DES - {1.2.840.113549.3.7}**

When executing in FIPS mode, encryption algorithms x509_alg_rc2CbcPad, x509_alg_rc4 and x509_alg_desCbcPad are not supported.

gsk_read_enveloped_data_msg_extended()

Processes a PKCS #7 EnvelopedData message.

Format

```
#include <gskcms.h>

gsk_status gsk_read_enveloped_data_msg_extended (
                                        gsk_process_option          option_flag,
                                        pkcs_cert_keys *            recipient_keys,
                                        gsk_buffer *                stream,
                                        x509_algorithm_type *       encryption_algorithm,
                                        gsk_size *                  key_size,
                                        gsk_buffer *                data)
```

Parameters

option_flag
Specifies process options to customize process behavior.

- Enforce recipient certificate has key encipherment capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate keyEncipherment.
- Enforce key parity when using DES or 3DES session keys.

recipient_keys
Specifies one or more certificates and associated private keys.

stream
Specifies the ASN.1 DER-encoded stream to be processed.

encryption_algorithm
Returns the encryption algorithm used to encrypt the message content.

key_size
Returns the encryption key size in bytes.

data
Returns the content of the EnvelopedData message. The application should call the **gsk_free_buffer()** routine to release the data when it is no longer needed.

Results

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

[CMSERR_BAD_ENCODING]
The message content type is not EnvelopedData or the message content is not Data.

[CMSERR_BAD_KEY_SIZE]
The encryption key size is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The message content type is not EnvelopedData or the content of the EnvelopedData message is not Data.

[CMSERR_INCORRECT_KEY_USAGE]
The recipient certificate does not allow key encipherment.

[CMSERR_KEY_MISMATCH]
A recipient private key does not support data decryption.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECIPIENT_NOT_FOUND]
No matching recipient certificate provided.

Usage

The **gsk_read_enveloped_data_msg_extended()** routine processes a PKCS #7 (Cryptographic Message Syntax) EnvelopedData message created by the **gsk_make_enveloped_data_content()** routine or the **gsk_make_enveloped_data_msg_extended()** routine and returns the message content. Processing is equivalent to **gsk_read_enveloped_data_content()**, except that the recipient certificate key usage need not assert key encipherment and session keys need not be odd parity. The enveloped data content type must be Data.

Calling the **gsk_read_enveloped_data_msg_extended()** routine is equivalent to calling the **gsk_read_content_msg()** routine, the **gsk_read_enveloped_data_content_extended()** routine, and the **gsk_read_data_content()** routine.

The recipient_keys parameter supplies one or more recipient certificates and associated private keys. The **gsk_read_enveloped_data_msg_extended()** routine will search for a certificate matching one of the message recipients. The private key will be used to decrypt the session key and the session key will then be used to decrypt the enveloped data. If *option_flag* specifies that key encipherment is to be enforced, then the certificate key usage must allow key encipherment.

No certificate validation is performed by the **gsk_read_enveloped_data_msg_extended()** routine. It is assumed that the application has already validated the recipient certificates.

These encryption algorithms are supported. Strong encryption may not be available depending upon government export regulations.

- **x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - {1.2.840.113549.3.2}**
- **x509_alg_rc4 - 40-bit and 128-bit RC4 - {1.2.840.113549.3.4}**
- **x509_alg_desCbcPad - 56-bit DES - {1.3.14.3.2.7}**
- **x509_alg_desEde3CbcPad - 168-bit 3DES - {1.2.840.113549.3.7}**

When executing in FIPS mode, encryption algorithms x509_alg_rc2CbcPad, x509_alg_rc4 and x509_alg_desCbcPad are not supported.

gsk_read_signed_data_content()

Processes PKCS #7 SignedData content information.

Format

```
#include <gskcms.h>

gsk_status gsk_read_signed_data_content (
                                        pkcs_certificates *         local_certificates,
                                        pkcs_content_info *         content_info,
                                        gsk_boolean *               used_local,
                                        pkcs_certificates *         msg_certificates,
                                        pkcs_certificates *         signer_certificates,
                                        pkcs_content_info *         content_data)
```

Parameters

local_certificates
Specifies zero or more X.509 certificates to use when verifying the message signatures. NULL can be specified for this parameter if no local certificates are provided.

content_info
Specifies the content information to be processed.

used_local
This parameter will be set to TRUE if the signatures were verified using just the certificates supplied by the local_certificates parameter. This parameter will be set to FALSE if any of the signatures were verified using certificates contained within the message.

msg_certificates
Returns the X.509 certificates contained within the message. The application should call the **gsk_free_certificates()** routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the message certificates are not needed.

signer_certificates
Returns the certificates used to sign the message. The application should call the **gsk_free_certificates()** routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the signer certificates are not needed.

content_data
Returns the SignedData content data. The application should call the **gsk_free_content_info()** routine to release the data when it is no longer needed.

Results

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

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_BAD_SIGNATURE]
Signature is not correct.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not SignedData.

[CMSERR_DIGEST_KEY_MISMATCH]
The digest algorithm is not supported for the private key type.

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[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 will attempt to locate the signer certificate in the SignedData message. An error will be returned if the signer certificate cannot be found or if the certificate key usage does not allow digital signature.

No certificate validation is performed by the **gsk_read_signed_data_content()** routine. It is assumed that the application has already validated the local certificates. The certificates contained in the SignedData message will be returned in the *msg_certificates* parameter and the *used_local* parameter will be set to FALSE if any of these certificates were used to verify the message signatures. It is the responsibility of the application to validate the message certificates (for example, by calling the **gsk_validate_certificate()** routine for each of the signer certificates).

These digest algorithms are supported:

x509_alg_md2Digest
MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

x509_alg_md5Digest
MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

x509_alg_sha1Digest
SHA-1 digest (RSA and DSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest
SHA-224 digest (RSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest
SHA-256 digest (RSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest
SHA-384 digest (RSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest
SHA-512 digest (RSA keys only) - {2.16.840.1.101.3.4.2.3}

When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not supported.

gsk_read_signed_data_content_extended()

Processes PKCS #7 SignedData content information.

Format

```
#include <gskcms.h>

gsk_status gsk_read_signed_data_content_extended (
                                        gsk_process_option          option_flag
                                        pkcs_certificates *         local_certificates,
                                        pkcs_content_info *         content_info,
                                        gsk_boolean *               used_local,
                                        pkcs_certificates *         msg_certificates,
                                        pkcs_certificates *         signer_certificates,
                                        gsk_attributes_signers *    attributes_signers,
                                        pkcs_content_info *         content_data)
```

Parameters

option_flag
Specifies process options to customize process behavior.

- Enforce signing certificate has digital signing capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate digitalSignature.
- Don't allow zero-length content data.

local_certificates
Specifies zero or more X.509 certificates to use when verifying the message signatures. NULL can be specified for this parameter if no local certificates are provided.

content_info
Specifies the content information to be processed.

used_local
This parameter will be set to TRUE if the signatures were verified using just the certificates supplied by the local_certificates parameter. This parameter will be set to FALSE if any of the signatures were verified using certificates contained within the message.

msg_certificates
Returns the X.509 certificates contained within the message. The application should call the **gsk_free_certificates()** routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the message certificates are not needed.

signer_certificates
Returns the certificates used to sign the message. The application should call the **gsk_free_certificates()** routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the signer certificates are not needed.

attributes_signers
Returns the authenticated attributes per signer contained within the message. The application should call the **gsk_free_attributes_signers()** routine to release the gsk_attributes_signers structure when it is no longer needed. Specify NULL for this parameter if the authenticated attributes per signer are not needed. The set of authenticated attributes returned, omits the content-type and message-digest authenticated attributes as these authenticated attributes must always be present, if any authenticated attributes are present, and are automatically verified by **gsk_read_signed_data_content_extended()**. The *digestAlgorithm* field within each *gsk_attributes_signer* structure returns the digest algorithm originally used for the signer.

content_data
Returns the SignedData content data. The application should call the **gsk_free_content_info()** routine to release the data when it is no longer needed.

Results

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

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_BAD_SIGNATURE]
Signature is not correct.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not SignedData.

[CMSERR_DIGEST_KEY_MISMATCH]
The digest algorithm is not supported for the private key type.

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_SIGNER_NOT_FOUND]
Signer certificate not found.

Usage

The **gsk_read_signed_data_content_extended()** routine processes PKCS #7 (Cryptographic Message Syntax) SignedData message created by the **gsk_make_signed_data_content_extended()** routine and returns the content data and authenticated attributes per signed (if present).

Processing is equivalent to **gsk_read_signed_data_content()**, with these differences:

- The signing certificate key usage need not assert digital signing capabilities depending on *option_flag*.
- Zero length content is acceptable depending on *option_flag*.
- Authenticated attributes and the digest algorithm used to create the signed data per signer, if present, are returned.

The *local_certificates* parameter can supply the signer certificates used to verify the message signatures. If a certificate is not found for a message signer, the **gsk_read_signed_data_content_extended()** routine will attempt to locate the signer certificate in the SignedData message. An error will be returned if the signer certificate cannot be found. An error may optionally be returned if the certificate key usage does not allow digital signature.

No certificate validation is performed by the **gsk_read_signed_data_content_extended()** routine. It is assumed that the application has already validated the local certificates. The certificates contained in the SignedData message will be returned in the *msg_certificates* parameter and the *used_local* parameter will be set to FALSE if any of these certificates were used to verify the message signatures. It is the responsibility of the application to validate the message certificates (for example, by calling the **gsk_validate_certificate()** routine for each of the signer certificates).

These digest algorithms are supported:

x509_alg_md2Digest
MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

x509_alg_md5Digest
MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

x509_alg_sha1Digest
SHA-1 digest (RSA and DSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest
SHA-224 digest (RSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest
SHA-256 digest (RSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest
SHA-384 digest (RSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest
SHA-512 digest (RSA keys only) - {2.16.840.1.101.3.4.2.3}

If authenticated attributes are returned via the *attributes_signers* parameter, then it is recommended that signing certificates for all signers represented within the gsk_attributes_signers structure should be requested via the *signer_certificates* parameter.

When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not supported.

gsk_read_signed_data_msg()

Processes a PKCS #7 SignedData message.

Format

```
#include <gskcms.h>

gsk_status gsk_read_signed_data_msg (
                     pkcs_certificates *       local_certificates,
                     gsk_buffer *              stream,
                     gsk_boolean *             used_local,
                     pkcs_certificates *       msg_certificates,
                     pkcs_certificates *       signer_certificates,
                     gsk_buffer *              data)
```

Parameters

local_certificates
Specifies zero or more X.509 certificates to use when verifying the message signatures. NULL can be specified for this parameter if no local certificates are provided.

stream
Specifies the ASN.1 DER-encoded stream to be processed.

used_local
This parameter will be set to TRUE if the signatures were verified using just the certificates supplied by the local_certificates parameter. This parameter will be set to FALSE if any of the signatures were verified using certificates contained within the message.

msg_certificates
Returns the X.509 certificates contained within the message. The application should call the **gsk_free_certificates()** routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the message certificates are not needed.

signer_certificates
Returns the certificates used to sign the message. The application should call the **gsk_free_certificates()** routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the signer certificates are not needed.

data
Returns the content of the SignedData message. The application should call the **gsk_free_buffer()** routine to release the data when it is no longer needed.

Results

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

[ASN_NO_MEMORY]
Insufficient storage is available.

[ASN_SELECTION_OUT_OF_RANGE]
Certificate type or version number is not valid.

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The message content type is not SignedData or the content of the SignedData message is not Data.

[CMSERR_BAD_SIGNATURE]
Signature is not correct.

[CMSERR_DIGEST_KEY_MISMATCH]
The digest algorithm is not supported for the private key type.

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_SIGNER_NOT_FOUND]
Signer certificate not found.

Usage

The **gsk_read_signed_data_msg()** routine processes a PKCS #7 (Cryptographic Message Syntax) SignedData message created by the **gsk_make_signed_data_msg()** routine and returns the message content. The signed data content type must be Data.

Calling the **gsk_read_signed_data_msg()** routine is equivalent to calling the **gsk_read_content_msg()** routine, the **gsk_read_signed_data_content()** routine, and the **gsk_read_data_content()** routine.

The *local_certificates* parameter can supply the signer certificates used to verify the message signatures. If a certificate is not found for a message signer, the **gsk_read_signed_data_msg()** routine will attempt to locate the signer certificate in the SignedData message. An error will be returned if the signer certificate cannot be found or if the certificate key usage does not allow digital signature.

No certificate validation is performed by the **gsk_read_signed_data_msg()** routine. It is assumed that the application has already validated the local certificates. The certificates contained in the SignedData message will be returned in the *msg_certificates* parameter and the *used_local* parameter will be set to FALSE if any of these certificates were used to verify the message signatures. It is the responsibility of the application to validate the message certificates (for example, by calling the **gsk_validate_certificate()** routine for each of the signer certificates).

These digest algorithms are supported:

x509_alg_md2Digest
MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

x509_alg_md5Digest
MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

x509_alg_sha1Digest
SHA-1 digest (RSA and DSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest
SHA-224 digest (RSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest
SHA-256 digest (RSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest
SHA-384 digest (RSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest
SHA-512 digest (RSA keys only) - {2.16.840.1.101.3.4.2.3}

When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not supported.

gsk_read_signed_data_msg_extended()

Processes a PKCS #7 SignedData message.

Format

```
#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.
- Don't allow zero-length content data.

local_certificates
Specifies zero or more X.509 certificates to use when verifying the message signatures. NULL can be specified for this parameter if no local certificates are provided.

stream
Specifies the ASN.1 DER-encoded stream to be processed.

used_local
This parameter will be set to TRUE if the signatures were verified using just the certificates supplied by the local_certificates parameter. This parameter will be set to FALSE if any of the signatures were verified using certificates contained within the message.

msg_certificates
Returns the X.509 certificates contained within the message. The application should call the **gsk_free_certificates()** routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the message certificates are not needed.

signer_certificates
Returns the certificates used to sign the message. The application should call the **gsk_free_certificates()** routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the signer certificates are not needed.

attributes_signers
Returns the authenticated attributes per signer contained within the message. The application should call the **gsk_free_attributes_signers()** routine to release the gsk_attributes_signers structure when it is no longer needed. Specify NULL for this parameter if the authenticated attributes per signer are not needed. The set of authenticated attributes returned, omits the content-type and message-digest authenticated attributes as these authenticated attributes must always be present, if any authenticated attributes are present, and are automatically verified by **gsk_read_signed_data_msg_extended()**. The *digestAlgorithm* field within each *gsk_attributes_signer* structure returns the digest algorithm originally used for the signer.

data
Returns the content of the SignedData message. The application should call the **gsk_free_buffer()** routine to release the data when it is no longer needed.

Results

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

[ASN_NO_MEMORY]
Insufficient storage is available.

[ASN_SELECTION_OUT_OF_RANGE]
Certificate type or version number is not valid.

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The message content type is not SignedData or the content of the SignedData message is not Data.

[CMSERR_BAD_SIGNATURE]
Signature is not correct.

[CMSERR_DIGEST_KEY_MISMATCH]
The digest algorithm is not supported for the private key type.

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_SIGNER_NOT_FOUND]
Signer certificate not found.

Usage

The **gsk_read_signed_data_msg_extended()** routine processes a PKCS #7 (Cryptographic Message Syntax) SignedData message created by the **gsk_make_signed_data_msg_extended()** routine and returns the message content and all authenticated attributes (if present). The signed data content type must be Data.

Processing is equivalent to **gsk_read_signed_data_msg()**, with these differences:

- The signing certificate key usage need not assert digital signing capabilities depending on *option_flag*.
- Zero length content is acceptable depending on *option_flag*.
- Authenticated attributes and the digest algorithm used to create the signed data per signer, if present, are returned.

Calling the **gsk_read_signed_data_msg_extended()** routine is equivalent to calling the **gsk_read_content_msg()** routine, the **gsk_read_signed_data_content_extended()** routine, and the **gsk_read_data_content()** routine.

The *local_certificates* parameter can supply the signer certificates used to verify the message signatures. If a certificate is not found for a message signer, the **gsk_read_signed_data_msg_extended()** routine will attempt to locate the signer certificate in the SignedData message. An error will be returned if the signer certificate cannot be found. An error may optionally be returned if the certificate key usage does not allow digital signature.

No certificate validation is performed by the **gsk_read_signed_data_msg_extended()** routine. It is assumed that the application has already validated the local certificates. The certificates contained in the

SignedData message will be returned in the *msg_certificates* parameter and the *used_local* parameter will be set to FALSE if any of these certificates were used to verify the message signatures. It is the responsibility of the application to validate the message certificates (for example, by calling the **gsk_validate_certificate()** routine for each of the signer certificates).

These digest algorithms are supported:

x509_alg_md2Digest
MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

x509_alg_md5Digest
MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

x509_alg_sha1Digest
SHA-1 digest (RSA and DSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest
SHA-224 digest (RSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest
SHA-256 digest (RSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest
SHA-384 digest (RSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest
SHA-512 digest (RSA keys only) - {2.16.840.1.101.3.4.2.3}

If authenticated attributes are returned via the *attributes_signers* parameter, then it is recommended that signing certificates for all signers represented within the gsk_attributes_signers structure should be requested via the *signer_certificates* parameter.

When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not supported.

gsk_replace_record()

Replaces a record in a key or request database.

Format

```
#include <gskcms.h>

gsk_status gsk_replace_record (
                              gsk_handle              db_handle,
                              gskdb_record *          record)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine or the **gsk_open_database()** routine.

record
Specifies the database record.

Results

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

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not supported.

[CMSERR_BACKUP_EXISTS]
The backup file already exists.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_LABEL]
The record label is not valid.

[CMSERR_BAD_RNG_OUTPUT]
In FIPS mode, random bytes generation produced duplicate output.

[CMSERR_DEFAULT_KEY_CHANGED]
The default key cannot be changed.

[CMSERR_INCORRECT_DBTYPE]
The record type is not supported for the database type.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
No private key is provided for a record type that requires a private key.

[CMSERR_PUBLIC_KEY_CHANGED]
The subject public key cannot be changed.

[CMSERR_RECORD_NOT_FOUND]
Record is not found.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_RECTYPE_NOT_VALID]
The record type is not valid.

[CMSERR_SUBJECT_CHANGED]
The subject name cannot be changed.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_replace_record()** routine replaces a record in a key or request database. The database must be open for update in order to replace records. The unique record identifier identifies the record to be replaced. Unused and reserved fields in the gskdb_record structure must be initialized to zero. If the record has a private key, the encrypted private key will be generated from the private key supplied in the database record.

The recordType field identifies the database record type as follows:

gskdb_rectype_certificate
The record contains an X.509 certificate.

gskdb_rectype_certKey
The record contains an X.509 certificate and private key.

gskdb_rectype_keyPairTerm
The record contains a PKCS #10 certification request and private key.

The recordFlags field is a bit field with these values:

GSKDB_RECFLAG_TRUSTED
The certificate is trusted.

GSKDB_RECFLAG_DEFAULT
This is the default key

The record label is used as a friendly name for the database entry and is in the local code page. It can be set to any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be set to an empty string.

If the record contains a certificate, the certificate will be validated and the record will not be replaced in the database if the validation check fails. If executing in FIPS mode, only FIPS-approved algorithms and key sizes are supported.

With the exception of the record label, all character strings are specified using UTF-8.

The record type, subject name, and subject public key cannot be changed when replacing a record. In addition, the GSKDB_RECFLAG_DEFAULT flag cannot be changed when replacing a record (call the **gsk_set_default_key()** routine to change the default record for the database).

The database file is updated as part of the **gsk_replace_record()** processing. A temporary database file is created using the same name as the database file with ″.new″ appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.

gsk_set_default_key()

Sets the default key.

Format

```
#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.

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_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_KEY_SIZE]
The key size is not valid.

[CMSERR_KEY_MISMATCH]
The supplied key does not match the signature algorithm.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_sign_certificate()** routine will sign an X.509 certificate using the supplied private key. The private key can be an RSA key or a DSA key. If executing in FIPS mode, the minimum key size is 1024 bits. The private key can be an ASN.1-encoded value contained in the privateKey field or an ICSF key label contained in the keyToken field. In either case, the key type must be specified by the privateKeyAlgorithm field.

The signature algorithm is obtained from the signature field of the x509_tbs_certificate structure contained within the x509_certificate structure. The generated signature will be placed in the signatureAlgorithm and signatureValue fields of the x509_certificate structure.

The following signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

gsk_sign_crl()

Signs an X.509 certificate revocation list.

Format

```
#include <gskcms.h>

gsk_status gsk_sign_crl (
                         x509_crl *                      crl,
                         pkcs_private_key_info *         private_key)
```

Parameters

crl Specifies the X.509 certificate revocation list.

private_key
Specifies the private key.

Results

The return status will be zero if the signature is successfully generated. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not supported.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_KEY_MISMATCH]
The supplied key does not match the signature algorithm.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_sign_crl()** routine will sign an X.509 certificate revocation list using the supplied private key. The private key can be an RSA key or a DSA key. If executing in FIPS mode, the minimum key size is 1024 bits. The private key can be an ASN.1-encoded value contained in the privateKey field or an ICSF key label contained in the keyToken field. In either case, the key type must be specified by the privateKeyAlgorithm field.

The signature algorithm is obtained from the signature field of the x509_tbs_crl structure contained within the x509_crl structure. The generated signature will be placed in the signatureAlgorithm and signatureValue fields of the x509_crl structure.

The following signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

| When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and
| x509_alg_md5WithRsaEncryption are not supported.

gsk_sign_data()

Signs a data stream.

Format

```
#include <gskcms.h>

gsk_status gsk_sign_data (
                          x509_algorithm_type              sign_algorithm,
                          pkcs_private_key_info *          private_key,
                          gsk_boolean                      is_digest,
                          gsk_buffer *                     data,
                          gsk_buffer *                     signature)
```

Parameters

sign_algorithm
Specifies the signature algorithm.

private_key
Specifies the private key.

is_digest
Specify TRUE if the data stream digest has been computed or FALSE if the data stream digest needs to be computed.

data
Specifies either the data stream digest (is_digest is TRUE) or the data stream (is_digest is FALSE).

signature
Returns the generated signature. The caller should release the signature buffer when it is no longer needed by calling the **gsk_free_buffer()** routine.

Results

The return status will be zero if the signature is successfully generated. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not supported.

[CMSERR_BAD_DIGEST_SIZE]
The digest size is not correct.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_RNG_OUTPUT]
In FIPS mode, random bytes generation produced duplicate output.

[CMSERR_KEY_MISMATCH]
The supplied key does not match the signature algorithm.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_sign_data()** routine will generate the signature for a data stream using the supplied private key. The private key can be an RSA key or a DSA key. If executing in FIPS mode, the minimum key size is 1024 bits. The private key can be an ASN.1-encoded value contained in the privateKey field or an ICSF key label contained in the keyToken field. In either case, the key type must be specified by the privateKeyAlgorithm field.

The application can either provide the message digest or have the **gsk_sign_data()** routine compute the message digest.

When the application provides the message digest, the digest length must be correct for the specified signature algorithm. Digest lengths: MD2 and MD5 are 16 bytes; SHA-1 is 20 bytes; SHA-224 is 28 bytes; SHA-256 is 32 bytes; SHA-384 is 48 bytes and SHA-512 is 64 bytes. The supplied digest will be used as-is without any further processing (specifically, for an RSA encryption key, the digest will not be encoded as an ASN.1 DigestInfo sequence before generating the signature)

The following signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_md5WithRsaEncryption
RSA encryption with combined MD5 and SHA-1 digests

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

gsk_validate_certificate()

Validates an X.509 certificate.

Format

```
#include <gskcms.h>

gsk_status gsk_validate_certificate (
                                    gskdb_data_sources *          data_sources,
                                    x509_certificate *            subject_certificate,
                                    gsk_boolean                   accept_root,
                                    gsk_int32 *                   issuer_record_id)
```

Parameters

data_sources
Specifies the data sources for CA certificates and revocation lists. The data sources are searched in the order they occur in the data source array, so trusted sources should be included before untrusted sources and local sources should be included before remote sources.

subject_certificate
Specifies the certificate to be validated.

accept_root
Specify TRUE if a self-signed root certificate is to be accepted without checking the data sources. Specify FALSE if a self-signed root certificate must be found in one of the trusted data sources in order to be accepted.

issuer_record_id
Returns the record identifier for the issuer certificate used to validate the certificate. The record identifier will be 0 if the issuer certificate is found in a non-database source. Specify NULL for this parameter if the issuer record identifier is not needed.

Results

The return status will be zero if the validation is successful. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not supported.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_ISSUER_NAME]
The certificate issuer name is not valid.

[CMSERR_BAD_SIGNATURE]
The signature is not correct.

[CMSERR_CERT_CHAIN_NOT_TRUST]
The certification chain is not trusted

[CMSERR_CERTIFICATE_REVOKED]
The certificate is revoked.

[CMSERR_EXPIRED]
The certificate is expired.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_USAGE]
The issuer certificate does not allow signing certificates

[CMSERR_ISSUER_NOT_CA]
The certificate issuer is not a certification authority.

[CMSERR_ISSUER_NOT_FOUND]
The issuer certificate is not found in one of the data sources.

[CMSERR_NAME_CONSTRAINTS_VIOLATED]
The certificate name is not consistent with the name constraints.

[CMSERR_NAME_NOT_SUPPORTED]
The AuthorityKeyIdentifier extension name is not a directory name.

[CMSERR_NOT_YET_VALID]
The certificate is not yet valid.

[CMSERR_PATH_TOO_LONG]
The certification chain exceeds the maximum allowed by the CA.

[CMSERR_SELF_SIGNED_NOT_FOUND]
A self-signed certificate is not found in a trusted data source

Usage

The **gsk_validate_certificate()** routine validates an X.509 certificate by performing these checks on the subject certificate:

- The certificate subject name must be either a non-empty distinguished name or an empty distinguished name with a SubjectAltName certificate extension
- An empty subject name is not allowed for a CA certificate
- The certificate issuer name must not be an empty distinguished name
- The CertificatePolicy extension, if present, must not be a critical extension
- The current time must not be earlier than the start of the certificate validity period
- The current time must not be later than the end of the certificate validity period
- The issuer certificate must be a valid CA certificate
- The certificate signature must be correct
- The certificate must not be revoked
- The certification chain must lead to a certificate obtained from a trusted data source
- No certificate in the certification chain can be revoked or expired.

| If executing in FIPS mode, only FIPS-approved algorithms and key sizes are supported (see Chapter 2,
| "System SSL and FIPS 140-2," on page 3 for more details).

The **gsk_validate_certificate()** routine will obtain any necessary CA certificates from the supplied data sources. The CA certificate will be validated as described if it is obtained from an untrusted data source. In addition, these checks will be performed on CA certificates when validating the certification chain:

- The BasicConstraints extension, if present, must have the CA indicator set and the path length constraint must not be violated by subordinate certificates in the certification chain
- The NameConstraints extension, if present, must not be violated by the subject certificate

A root certificate is a self-signed certificate and its signature is verified using the public key in the certificate. If accept_root is FALSE, the root certificate must be found in a trusted data source in order to be accepted. If accept_root is TRUE, the self-signed certificate is accepted as long as the signature is correct.

An intermediate certificate or an end-entity certificate is a certificate signed by another entity. Its signature is verified using the public key in the issuer's certificate. The issuer certificate must be found in one of the supplied data sources. When intermediate CA certificates are used, the certificate chain is validated until an issuer is reached whose certificate is in one of the trusted data sources.

The data sources must contain at least one LDAP directory source or CRL source in order to check for revoked certificates. The CRL distribution point name (or the certificate issuer name if the certificate does not have a CrlDistributionPoints extension) is used as the distinguished name of the LDAP directory entry containing the certificate revocation list (CRL). The CRL distribution point name and CRL issuer name must be X.500 directory names. The BasicConstraints certificate extension determines whether the CA revocation list or the user revocation list is used. An error will be returned if a CRL obtained from an untrusted source cannot be validated.

Security levels for connecting to LDAP directories are based on the GSKCMS_CRL_SECURITY_LEVEL setting. When using the CMS APIs, the GSKCMS_CRL_SECURITY_LEVEL setting can be specified using the gsk_set_directory_enum() routine. Security levels can be set to LOW, MEDIUM or HIGH. See "gsk_attribute_set_enum()" on page 12 and "Environment Variables," on page 211 for further information on CRL security level settings.

These data sources are supported:

- gskdb_source_key_database - The source is a key database. The handle must be a database handle returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine, or the **gsk_open_keyring()** routine. This is a trusted data source.
- gskdb_source_directory - The source is an LDAP directory. The handle must be the directory handle returned by the **gsk_open_directory()** routine. This is an untrusted data source. Any certificate or revocation list obtained from this source will be validated before being accepted. Refer to the **gsk_get_directory_certificates()** and **gsk_get_directory_crls()** routines for more information concerning the use of LDAP directory entries.
- gskdb_source_trusted_certs - The source is an array of certificates. This is a trusted data source.
- gskdb_source_untrusted_certs - The source is an array of certificates. This is an untrusted data source. Any certificate used from this list will be validated before being accepted.
- gskdb_source_trusted_crls - The source is an array of certificate revocation lists. This is a trusted data source.
- gskdb_source_untrusted_crls - The source is an array of certificate revocation lists. This is an untrusted data source. Any CRL used from this list will be validated before being accepted.
- gskdb_source_cert_callback - The source is the address of a callback routine which will receive control when an issuer certificate is needed. This is a trusted data source. The subject name is passed as an input parameter and the certCallback routine returns an array of one or more certificates with that subject name. The **gsk_validate_certificate()** routine will call the freeCallback routine to release the certificates. The return status should be 0 if no errors are detected. Otherwise it should be one of the error code listed in the **gskcms.h** include file. The return status should be 0 and the certificate count should be 0 if there are no certificates matching the supplied subject name.
- gskdb_source_crl_callback - The source is the address of a callback routine which will receive control when a certificate needs to be checked to see if it has been revoked. The return value should be 0 if the certificate is not revoked. Otherwise it should be one of the error codes defined in the **gskcms.h** include file.

gsk_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_SIGNATURE]
The signature is not correct.

[CMSERR_KEY_MISMATCH]
The supplied key does not match the signature algorithm.

Usage

The **gsk_verify_certificate_signature()** routine validates an X.509 certificate by computing its signature and then comparing the result to the signature contained in the certificate.

The following signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_md5Sha1WithRsaEncryption
RSA encryption with combined MD5 and SHA-1 digests

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

gsk_verify_crl_signature()

Verifies the signature for an X.509 certificate revocation list.

Format

```
#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_SIGNATURE]
The signature is not correct.

[CMSERR_KEY_MISMATCH]
The supplied key does not match the signature algorithm.

Usage

The **gsk_verify_crl_signature()** routine validates an X.509 certificate revocation list (CRL) by computing its signature and then comparing the result to the signature contained in the CRL.

The following signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_md5Sha1WithRsaEncryption
RSA encryption with combined MD5 and SHA-1 digests

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

gsk_verify_data_signature()

Verifies the signature for a data stream.

Format

```
#include <gskcms.h>

gsk_status gsk_verify_data_signature (
                                      x509_algorithm_type           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_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_SIGNATURE]
The signature is not correct.

[CMSERR_KEY_MISMATCH]
The supplied key does not match the signature algorithm.

Usage

The **gsk_verify_data_signature()** routine validates the signature for a data stream. The public key can be an RSA key or a DSA key.

The application can either provide the message digest or have the **gsk_verify_signed_data()** routine compute the message digest.

When the application provides the message digest, the digest length must be correct for the specified signature algorithm. Digest lengths: MD2 and MD5 are 16 bytes; SHA-1 is 20 bytes; SHA-224 is 28 bytes;

SHA-256 is 32 bytes; SHA-384 is 48 bytes and SHA-512 is 64 bytes. The supplied digest will be used as-is without any further processing (specifically, for an RSA encryption key, the digest will not be encoded as an ASN.1 DigestInfo sequence before comparing it with the digest in the signature)

The following signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_md5Sha1WithRsaEncryption
RSA encryption with combined MD5 and SHA-1 digests

The x509_alg_md5Sha1WithRsaEncryption algorithm is a special algorithm used by the SSL protocol. The data signature consists of the MD5 digest over the data followed by the SHA-1 digest over the data for a total digest length of 36 bytes. The digest is encrypted as-is without any further processing.

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

Chapter 5. Deprecated Secure Sockets Layer APIs

These set of application program interfaces, or APIs, have been superseded by the APIs defined in Chapter 3, “API Reference,” on page 9. These were affected by APAR OA26457.

- **gsk_get_cipher_info()** (see page 160)
- **gsk_initialize()** (see page 161)
- **gsk_secure_soc_init()** (see page 165)
- **gsk_srb_initialize()** (see page 172)

It is **strongly recommended** that existing applications be modified to make use of the set of APIs defined in Chapter 3, “API Reference.” Those modified applications should **only** use the new APIs, and **not** a mix of the new APIs and these deprecated APIs.

IBM may remove support of APIs contained within this topic in a future release.

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 domestic cipher specifications as well as the export cipher specifications.

sec_level
Returns the cipher specifications.

rsvd
Reserved for future use. Specify NULL for this parameter.

Results

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

[GSK_BAD_PARAMETER]
The level value is not valid or a NULL address is specified for sec_level.

Usage

The **gsk_get_cipher_info()** routine returns the available cipher specifications. Both domestic and export ciphers will be included if GSK_HIGH_SECURITY is specified while only export ciphers will be included if GSK_LOW_SECURITY is specified. The **gsk_get_cipher_info()** routine can be called at any time and does not require the **gsk_initialize()** routine to be called first.

The SSL V2 cipher specifications returned for GSK_HIGH_SECURITY are ″713642″ while the SSL V3 cipher specifications are ″050435363738392F303132330A1613100D0915120F0C0306020100″ if not in FIPS mode, and ″35363738392F30313233FF0A1613100D″ in FIPS mode. If the Security Level 3 FMID is not installed, the SSL V2 cipher specifications are ″642″, the SSL V3 cipher specifications are ″0915120F0C0306020100″ and FIPS mode is not supported.

The SSL V2 cipher specifications returned for GSK_LOW SECURITY are ″642″ while the SSL V3 cipher specifications are ″0915120F0C0306020100″ in non-FIPS mode and ″″ in FIPS mode.

Related Topics

gsk_secure_soc_init()

gsk_initialize()

gsk_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_LDAP]
Unable to initialize the LDAP client.

[GSK_ERROR_MULTIPLE_LABEL]
Multiple certificates exist for label.

[GSK_ERROR_MULTIPLE_DEFAULT]
Multiple keys are marked as the default.

[GSK_ERROR_PERMISSION_DENIED]
Not authorized to access the key database, key ring or token.

[GSK_INIT_SEC_TYPE_NOT_VALID]
The security type is not valid.

[GSK_INIT_V2_TIMEOUT_NOT_VALID]
The SSL V2 timeout is not valid.

[GSK_INIT_V3_TIMEOUT_NOT_VALID]
The SSL V3 timeout is not valid.

[GSK_KEYFILE_BAD_FORMAT]
Key database or key ring format is not valid.

[GSK_KEYFILE_BAD_PASSWORD]
Key database password is not correct.

[GSK_KEYFILE_IO_ERROR]
Unable to read the key database, key ring or token.

[GSK_KEYFILE_NO_CERTIFICATES]
The key database, key ring or token does not contain any certificates.

[GSK_KEYFILE_OPEN_FAILED]
Unable to open the key database, key ring or token.

[GSK_KEYFILE_PW_EXPIRED]
Key database password is expired.

Usage

The **gsk_initialize()** routine initializes the System SSL runtime environment for the current process. The **gsk_uninitialize()** routine should be called to release the SSL environment when it is no longer needed. Multiple calls to **gsk_initialize()** will cause the existing environment to be released before creating the new environment.

Environment variables are processed along with the **gsk_initialize** data structures. Information passed in the key database, key ring or token is read as part of the environment initialization. Upon successful completion of **gsk_initialize()**, the application is ready to begin creating and using secure socket connections.

The gsk_init_data structure contains these fields:

sec_types
Specifies one of these null-terminated character strings:

- ″SSLV2″ or ″SSL20″ to use the SSL V2 protocol
- ″SSLV3″ or ″SSL30″ to use the SSL V3 protocol
- ″TLSV1″ or ″TLS10″ to use the TLS V1.0 protocol
- "SSLV2_OFF" to allow either TLS V1.0 or SSL V3 to be used
- ″ALL″ to use any supported protocol

When "SSLV2_OFF" is specified the SSL client/server will attempt first to use the TLS V1.0 protocol, before falling back to the most secure protocol supported by its SSL partner, excluding the SSL V2 protocol.

When ″ALL" is specified for an SSL client, the client will attempt first to use the TLS V1.0 protocol and will fall-back to the most secure protocol that the server will support, excluding the SSL V2 protocol (the client must explicitly request the SSL V2 protocol if it wants to use this protocol).

When ″ALL" is specified for an SSL server, the server will accept any of the supported protocols.

When running in FIPS mode, the minimum requirement is TLS V1 protocol. If only the SSL V2 or the SSL V3 protocol is enabled, then a FIPS mode SSL connection is not possible.

keyring
Specifies the name of the key database, SAF key ring or z/OS PKCS #11 token as a null-terminated character string. When both the password and stash file name are NULL, a SAF key ring or PKCS #11 token is used.

The SAF key ring name is specified as ″userid/keyring″. The current userid is used if the userid is omitted. The user must have READ access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by the user. The user must have UPDATE access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by another user. Note that certificate private keys are not available when using a SAF key ring owned by another user.

The z/OS PKCS #11 token name is specified as *TOKEN*/token-name. *TOKEN* indicates that the specified key ring is actually a token name. The application userid must have READ access to resource USER.token-name in the CRYPTOZ class in order for the certificate and their private keys, if present, to be read.

keyring_pw
Specifies the password for the key database as a null-terminated character string. Specify NULL to indicate no password is provided.

keyring_stash
Specifies the name of the password stash file as a null-terminated character string. Specify NULL to indicate no stash file is provided. The password stash file is used if the keyring_pw value is NULL.

V2_session_timeout
Specifies the SSL V2 session cache timeout value in seconds. The valid range is 0 to 100. A short SSL handshake is performed when a cached session exists since the session parameters have already been negotiated between the client and the server.

V3_session_timeout
Specifies the SSL V3 session cache timeout value in seconds. The valid range is 0 to 86400. A short SSL handshake is performed when a cached session exists since the session parameters have already been negotiated between the client and the server.

LDAP_server
Specifies one or more blank-separated LDAP server host names as a null-terminated character string. Each host name can contain an optional port number separated from the host name by a colon. The LDAP server is used for certificate validation . The LDAP server is used only when LDAP_CA_roots is set to GSK_CA_ROOTS_LOCAL_AND_X500 and *auth_type* is not set to GSK_CLIENT_AUTH_LOCAL or GSK_CLIENT_AUTH_PASSTHRU.

LDAP_port
Specifies the LDAP server port. The default LDAP port will be used if 0 is specified.

LDAP_user
Specifies the distinguished name to use when connecting to the LDAP server and is a null-terminated character string. An anonymous bind will be done if NULL is specified for this field.

LDAP_password
Specifies the password to use when connecting to the LDAP server and is a null-terminated character string. This field is ignored if NULL is specified for LDAP_user.

LDAP_CA_roots
Specifies the location of CA certificates and certificate revocation lists used to validate certificates. When GSK_CA_ROOTS_LOCAL_ONLY is specified, the CA certificates and certificate revocation lists are obtained from the local database. When GSK_CA_ROOTS_LOCAL_AND_X500 is specified, the CA certificates and certificate revocation lists are obtained from the LDAP server if they are not found in the local database. Even when an LDAP server is used, root CA certificates must be found in the local database since the LDAP server is not a trusted data source.

auth_type
Specifies the client authentication type. This field is ignored unless LDAP_CA_roots is set to GSK_CA_ROOTS_LOCAL_AND_X500. The client certificate is not validated when GSK_CLIENT_AUTH_PASSTHRU is specified. The client certificate is validated using just the local database when GSK_CLIENT_AUTH_LOCAL is specified. CA certificates and certificate revocation lists not found in the local database will be obtained from the LDAP server when GSK_CLIENT_AUTH_STRONG or GSK_CLIENT_AUTH_STRONG_OVER_SSL is specified (the local database must still contain the root CA certificates). There is no difference between GSK_CLIENT_AUTH_STRONG and GSK_CLIENT_AUTH_STRONG_OVER_SSL.

Environment variables

Environment variables are processed along with the information passed in the gsk_init_data structure during environment initialization. Also during environment initialization, the key database, key ring or token is read.

The **gsk_initialize()** routine supports these environment variables:

GSKV2CACHESIZE
Specifies the number of entries in the SSL V2 session cache with a range of 0 to 32000. The value specified by the GSK_V2_SIDCACHE_SIZE environment variable will be used if the GSKV2CACHESIZE variable is not defined. The default value is 256 if neither environment variable is defined.

GSKV3CACHESIZE
Specifies the number of entries in the SSL V3 session cache with a range of 0 to 64000. The value specified by the GSK_V3_SIDCACHE_SIZE environment variable will be used if the GSKV3CACHESIZE variable is not defined. The default value is 512 if neither environment variable is defined. The SSL V3 session cache is used for both the SSL V3 and TLS V1.0 protocols.

The environment variables that will be overriden with information passed in the gsk_init_data structure are:

- GSK_KEYRING_FILE
- GSK_KEYRING_PW
- GSK_KEYRING_STASH
- GSK_LDAP_SERVER
- GSK_LDAP_PASSWORD
- GSK_LDAP_PORT
- GSK_LDAP_USER
- GSK_PROTOCOL_SSLV2
- GSK_PROTOCOL_SSLV3
- GSK_PROTOCOL_TLSV1
- GSK_V2_SESSION_TIMEOUT
- GSK_V3_SESSION_TIMEOUT

Related Topics

gsk_secure_soc_init()

gsk_secure_soc_read()

gsk_secure_soc_write()

gsk_secure_soc_close()

gsk_uninitialize()

gsk_secure_soc_init()

Initializes a secure socket connection.

Format

```
#include <gskssl.h>

gsk_soc_data * gsk_secure_soc_init(
                                   gsk_soc_init_data *     init_data)
```

Parameters

init_data
Specifies the socket connection initialization data.

Results

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

[GSK_ERR_INIT_PARM_NOT_VALID]
A connection initialization parameter is not valid.

[GSK_ERROR_BAD_CERT]
A certificate is not valid.

[GSK_ERROR_BAD_DATE]
A certificate is not valid yet or is expired.

[GSK_ERROR_BAD_MAC]
Message verification failed.

[GSK_ERROR_BAD_MALLOC]
Insufficient storage is available.

[GSK_ERROR_BAD_MESSAGE]
Incorrectly-formatted message received from peer application.

[GSK_ERROR_BAD_PEER]
Peer application has violated the SSL protocol.

[GSK_ERROR_BAD_STATE]
The SSL environment has not been initialized.

[GSK_ERROR_CRYPTO]
Cryptographic error detected.

[GSK_ERROR_INCOMPATIBLE_KEY]
The certificate key is not compatible with the negotiated cipher suite.

[GSK_ERROR_IO]
I/O error communicating with peer application.

[GSK_ERROR_LDAP]
An LDAP error is detected.

[GSK_ERROR_LDAP_NOT_AVAILABLE]
The LDAP server is not available.

[GSK_ERROR_NO_CIPHERS]
No cipher specifications.

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

[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_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_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.

[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.

Usage

The **gsk_secure_soc_init()** routine initializes a secure socket connection. The **gsk_initialize()** routine must be called before any secure socket connections can be initialized. After the connection has been initialized, it can be used for secure data transmission using the **gsk_secure_soc_read()** and **gsk_secure_soc_write()** routines. The **gsk_secure_soc_close()** routine should be called to close the connection when it is no longer needed. The **gsk_secure_soc_close()** routine should not be called if an error is returned by the **gsk_secure_soc_init()** routine.

Before calling the **gsk_secure_soc_init()** routine, the application must create a connected socket. For a client, this means calling the **socket()** and **connect()** routines. For a server, this means calling the **socket()**, **listen()**, and **accept()** routines. However, SSL does not require the use of TCP/IP for the communications layer. The socket descriptor can be any integer value that is meaningful to the application. The application must provide its own socket routines if it is not using TCP/IP.

An SSL handshake is performed as part of the processing of the **gsk_secure_soc_init()** routine. This establishes the server identity and optionally the client identity. It also negotiates the cryptographic parameters to be used for the connection.

| The server certificate can use either RSA or DSA as the public/private key algorithm. In FIPS mode, the
| RSA or DSA key size must be at least 1024 bits. An RSA certificate can be used with an RSA, fixed Diffie-Hellman or ephemeral Diffie-Hellman key exchange. A DSA certificate can be used with either a fixed

or ephemeral Diffie-Hellman key exchange. In FIPS mode, the Diffie-Hellman key size must be at least 2048 bits. If the server's certificate contains a key usage extension during the SSL handshake, it must allow key usage as follows:

- RSA certificates using export restricted ciphers (40-bit RC4 encryption and 40-bit RC2 encryption) with a public key size greater than 512 bits must allow digital signature. If operating in FIPS mode, export restricted ciphers cannot be selected.
- RSA or DSA certificates using fixed Diffie-Hellman key exchange must allow key agreement.
- Other RSA certificates must allow key encipherment.
- DSA certificates using ephemeral Diffie-Hellman key exchange must allow digital signature.

The client certificate must support digital signatures. This means the certificate key usage extension (if any) must allow digital signature. The key algorithm can be either the RSA encryption algorithm or the Digital Signature Standard algorithm (DSA).

The SSL server always provides its certificate to the SSL client as part of the handshake. Depending upon the server handshake type, the server may ask the client to provide its certificate. The key label stored in the connection is used to retrieve the certificate from the key database, key ring or token. The default key will be used if no label is set. The key record must contain both an X.509 certificate and a private key.

These SSL V2 cipher specifications are supported in non-FIPS mode only :

- ″1″ = 128-bit RC4 encryption with MD5 message authentication (128-bit secret key)
- ″2″ = 128-bit RC4 export encryption with MD5 message authentication (40-bit secret key)
- ″3″ = 128-bit RC2 encryption with MD5 message authentication (128-bit secret key)
- ″4″ = 128-bit RC2 export encryption with MD5 message authentication (40-bit secret key)
- ″6″ = 56-bit DES encryption with MD5 message authentication (56-bit secret key)
- ″7″ = 168-bit Triple DES encryption with MD5 message authentication (168-bit secret key)

These SSL V3 cipher specifications are supported in non-FIPS mode only:

- ″00″ = No encryption or message authentication and RSA key exchange
- ″01″ = No encryption with MD5 message authentication and RSA key exchange
- ″02″ = No encryption with SHA-1 message authentication and RSA key exchange
- ″03″ = 40-bit RC4 encryption with MD5 message authentication and RSA key exchange
- ″04″ = 128-bit RC4 encryption with MD5 message authentication and RSA key exchange
- ″05″ = 128-bit RC4 encryption with SHA-1 message authentication and RSA key exchange
- ″06″ = 40-bit RC2 encryption with MD5 message authentication and RSA key exchange
- ″09″ = 56-bit DES encryption with SHA-1 message authentication and RSA key exchange
- ″0C″ = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate
- ″0F″ = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- ″12″ = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate
- ″15″ = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate

These SSL V3 cipher specifications are supported in FIPS mode and non-FIPS mode:

- ″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 server's cipher list for a match in the client's list. The first matching cipher is used.

Environment variables

Environment variables are processed along with the information passed in the gsk_init_data structure during environment initialization. Also during environment initialization, the key database, key ring or token is read.

The environment variables that will be overridden by non-NULL values in the gsk_soc_init_data structure are:

- GSK_KEY_LABEL
- GSK_V2_CIPHER_SPECS
- GSK_V3_CIPHER_SPECS

The gsk_soc_init_data structure contains these fields:

fd Specifies the socket descriptor for the secure connection. The socket must remain open until after the **gsk_secure_soc_close()** routine has been called to close the secure connection.

hs_type
Specifies the desired handshake type as follows:

GSK_AS_CLIENT
Performs a client SSL handshake

GSK_AS_CLIENT_NO_AUTH
Performs a client SSL handshake but do not provide a client certificate to the SSL server

GSK_AS_SERVER
Performs a server SSL handshake

GSK_AS_SERVER_WITH_CLIENT_AUTH
Performs a server SSL handshake with client authentication

DName
Specifies either the distinguished name or the key label of the local certificate. Specify NULL to use the default key for the key database, key ring or token.

sec_type
Returns the selected security protocol as ″SSLV2″, ″SSLV3″, or ″TLSV1″. This is a static string and must not be modified or freed by the application.

cipher_specs
Specifies the SSL V2 cipher specifications as a null-terminated string consisting of 1 or more 1-character values. Specify NULL to use the default cipher specifications (″713642″ if Security Level 3 FMID encryption is enabled and ″642″ otherwise). Valid cipher specifications that are not supported due to the installed cryptographic level will be skipped when the connection is initialized. The SSL V2 protocol can only be used when executing in non-FIPS mode.

v3cipher_specs
Specifies the SSL V3 cipher specifications as a null-terminated string consisting of 1 or more 2-character values. Specify NULL to use the default cipher specifications (″050435363738392F303132330A1613100D0915120F0C0306020100″ if Security Level 3 FMID is installed and in non-FIPS mode, ″35363738392F303132330A1613100D″ if Security Level 3 FMID is installed and in FIPS mode, and ″0915120F0C0306020100″ otherwise). The SSL V3 cipher specifications are used for both the SSL V3 and TLS V1.0 protocols. Valid cipher specifications that are not supported due to the installed cryptographic level will be skipped when the connection is initialized. The SSL V3 protocol can only be used when executing in non-FIPS mode.

skread Specifies the address of the read routine used during the SSL handshake.

skwrite
Specifies the address of the write routine used during the SSL handshake.

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 exchabge
- ″03″ = 40-bit RC4 encryption with MD5 message authentication and RSA key exchange
- ″04″ = 128-bit RC4 encryption with MD5 message authentication and RSA key exchange
- ″05″ = 128-bit RC4 encryption with SHA-1 message authentication and RSA key exchange
- ″06″ = 40-bit RC2 encryption with MD5 message authentication and RSA key exchange

- ″09″ = 56-bit DES encryption with SHA-1 message authentication and RSA key exchange
- ″0A″ = 168-bit Triple DES encryption with SHA-1 message authentication and RSA key exchange
- ″0C″ = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate
- ″0D″ = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate
- ″0F″ = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- ″10″ = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- ″12″ = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate
- ″13″ = 168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate
- ″15″ = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
- ″16″ = 168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
- ″2F″ = 128-bit AES encryption with SHA-1 message authentication and RSA key exchange
- ″30″ = 128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate
- ″31″ = 128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- ″32″ = 128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate
- ″33″ = 128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
- ″35″ = 256-bit AES encryption with SHA-1 message authentication and RSA key exchange
- ″36″ = 256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate
- ″37″ = 256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- ″38″ = 256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate
- ″39″ = 256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate

failureReasonCode
Returns the **gsk_secure_soc_init()** error code.

cert_info
Returns peer certificate information. The application must not modify or free this information.

gsk_data
This field is ignored. The key database information is set when **gsk_initialize()** is called.

Related Topics

gsk_initialize()

gsk_secure_soc_write()

gsk_secure_soc_read()

gsk_secure_soc_close()

gsk_get_dn_by_label()

gsk_get_cipher_info()

gsk_secure_soc_reset()

gsk_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.

Refer to the z/OS Authorized Assembler Services Guide for more information about service request blocks.

Chapter 6. Certificate/Key Management

This topic discusses the use of the z/OS shell-based **gskkyman** utility. **gskkyman** is a certificate management utility provided by System SSL to create and manage certificates, certificate requests and PKI private keys within either a key database file or PKCS #11 token.

Setting Up the Environment to Run gskkyman

gskkyman uses the DLLs that are installed with System SSL and must have access to these at run-time. **gskkyman** must also have access to the message catalogs. The `/bin` directory includes a symbolic link to **gskkyman**, therefore, if your `PATH` environment variable contains this directory, you will find **gskkyman**. If your `PATH` environment variable does not contain this directory, add `/usr/lpp/gskssl/bin` to your `PATH` using:

```
PATH=$PATH:/usr/lpp/gskssl/bin
```

`/usr/lib/nls/msg/En_US.IBM-1047` (as well as `/usr/lib/nls/msg/Ja_JP.IBM-939` for JCPT3AJ 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 JCPT3AJ 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 JCPT3AJ 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 dataset (PDSE) in HLQ.SIEALNKE. These DLLs are **not** installed in SYS1.LPALIB by default. If System SSL is to execute in FIPS mode, the DLLs in the HLQ.SIEALNKE dataset cannot be put into the LPA.

If the System SSL DLLs have not been put into either the dynamic LPA or system link list, you must set the STEPLIB environment variable to find the DLLs. For example:

```
export STEPLIB=$STEPLIB:<HLQ>.SIEALNKE
```

Key Database Files

Key database files are password protected because they contain the private keys that are associated with some of the certificates that are contained in the key database. Private keys, as their name implies, should be protected because their value is used in verifying the authenticity of requests made during PKI operations.

It is recommended that key database files be set with these string file permissions:

```
-rw------- (600)   (read-write for only the owner of the key database)
```

The owner of the key database should be the user who will be managing the key database. The program using System SSL (and the key database) must have at least read permission to the key database file at run-time. If the program is a server program that runs under a different user ID than the administrator of the key database file, it is recommended that a group be setup to control access to the key database file. In this case, it is recommended that you set the permissions on the key database file to:

```
-rw-r---- (640)   (read-write for owner and read-only for group)
```

The owner of the key database file is set to the administrator user ID and the group owner of the key database file is set to the group that contains the server that will be using the key database file.

A key database that is created as a FIPS mode database, can only be updated by **gskkyman** or by using the CMS APIs executing in FIPS mode. Such a database, however, may be opened as read-only when executing in non-FIPS mode. Key databases created while in non-FIPS mode cannot be opened when executing in FIPS mode.

gskkyman Interactive Mode Descriptions

Interactive mode is entered when the **gskkyman** command is entered without any parameters. A series of menus will be presented to allow you to select the database functions to be performed. Leading and trailing blanks will be removed from data entries but imbedded blanks will be retained. Blanks will not be removed from passwords.

Database Menu

This is the top-level menu and is displayed when the **gskkyman** command starts:

```
       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 1. Database Menu

Create new database

This option will create a new key database and the associated request database. You will be prompted to enter the key database name, the database password, the password expiration interval, and the database record length and choose either a FIPS or non-FIPS database (see "Key Database Files" on page 173 for a discussion of FIPS mode databases).

The fully-qualified key database name must be between 2 and 251 characters and should either have no extension or an extension of '.kdb' (the maximum database name is 247 characters if the name does not end with an extension of 1-3 characters to allow for the addition of an extension when creating the request database or the password stash file). The key database name may not end with '.rdb' or '.sth' as these extensions are reserved for the request database and the password stash file.

The database password must be between 1 and 128 characters. A password exceeding 128 characters will be truncated to 128 characters.

The password expiration interval must be between 0 and 9999 days (a value of 0 indicates the password does not expire).

The record length must be large enough to contain the largest certificate to be stored in the database and must be between 2500 and 65536.

Two files will be created: the key database and the request database. The request database has an extension of '.rdb'. The file access permissions will be set so only the owner has access to the files.

Open database

This option will open an existing database. You will be prompted to enter the key database name and the database password.

The fully-qualified key database name must be between 2 and 251 characters and should either have no extension or an extension of '.kdb' (the maximum database name is 247 characters if the name does not end with an extension of 1-3 characters to allow for the addition of an extension when accessing the request database or the password stash file). The key database name may not end with '.rdb' or '.sth' as these extensions are reserved for the request database and the password stash file.

Change database password

This option will change the database password. You can change the password at any time but you must change it once it has expired in order to access the database once more. You will be prompted to enter the key database name, the current database password, the new database password, and the new password expiration interval.

The fully-qualified key database name must be between 2 and 251 characters and should either have no extension or an extension of '.kdb' (the maximum database name is 247 characters if the name does not end with an extension of 1-3 characters to allow for the addition of an extension when accessing the request database or the password stash file). The key database name may not end with '.rdb' or '.sth' as these extensions are reserved for the request database and the password stash file.

The new database password must be between 1 and 128 characters.

The password expiration interval must be between 0 and 9999 days (a value of 0 indicates the password does not expire).

Change database record length

This option will change the database record length. All database records have the same length and database entries cannot span records. You can increase the record length if you find it is too small to store a new certificate. You can decrease the record length to reduce the database size if the original record length is too large. You cannot reduce the record length to a value smaller than the largest certificate currently in the database. You will be prompted to enter the key database name, the database password, and the new record length.

The fully-qualified key database name must be between 2 and 251 characters and should either have no extension or an extension of '.kdb' (the maximum database name is 247 characters if the name does not end with an extension of 1-3 characters to allow for the addition of an extension when accessing the request database or the password stash file). The key database name may not end with '.rdb' or '.sth' as these extensions are reserved for the request database and the password stash file.

The new record length must be between 2500 and 65536.

Delete database

This option will delete the key database, the associated request database, and the database password stash file. You will be prompted to enter the key database name.

The fully-qualified key database name must be between 2 and 251 characters and should either have no extension or an extension of '.kdb' (the maximum database name is 247 characters if the name does not end with an extension of 1-3 characters to allow for the addition of an extension when accessing the request database or the password stash file). The key database name may not end with '.rdb' or '.sth' as these extensions are reserved for the request database and the password stash file.

Create key parameter file

This option will create a file containing a set of key generation parameters. Key generation parameters are used when generating Digital Signature Standard (DSS) and Diffie-Hellman (DH) keys. The parameters will be stored in the specified file as an ASN.1-encoded sequence in Base64 format. This file can then be used when creating a signed certificate. The same key generation parameters can be used to generate multiple public/private key pairs. Using the same key generation parameters significantly reduces the time required to generate a public/private key pair. In addition, the Diffie-Hellman key agreement method requires both sides to use the same group parameters in order to compute the key exchange value. Refer to FIPS 186-2 (Digital Signature Standard) and RFC 2631 (Diffie-Hellman Key Agreement Method) for more information on the key generation parameters. The key parameter generation process can take from 1 to 10 minutes depending upon key size, processor speed and system load.

Display certificate file (Binary or Base64 ASN.1 DER)

This option displays information about an X.509 certificate file. You will be prompted to enter the certificate filename. The fully-qualified certificate filename must be between 2 and 251 characters. The specified file must contain either a binary ASN.1 DER-encoded certificate or the Base64-encoding of a binary ASN.1 stream. A Base64-encoded certificate must be in the local code page.

Note: Information retrieved for z/OS PKCS #11 tokens is not cached. Each time a menu is displayed, the information is retrieved from the ICSF TKDS (token key dataspace). This is also true when displaying the list of available z/OS PKCS #11 tokens. On return from displaying a subordinate menu, the current list of tokens is retrieved and the menu refreshed.

Create new token

This option will create a new token. You will be prompted to enter the token name.

The name must be a unique non-empty string and consist of characters that are alphanumeric, national (@ -x5B, # -x7B, $ -x7C) and period (x4B).

The name is specified in the local code page.

The first character must be alphabetic or national. Lowercase letters are permitted but will be folded to uppercase.

Once the token is created the Database Menu is displayed.

Delete token

This option will delete the key token. You will be prompted to enter the token name. If the token exists, the user is prompted again to re-enter the full token name as confirmation prior to deletion of the specified token.

Note: If name consists of lowercase characters it will be uppercased when processed.

Manage token

This option manages the token. You will be prompted to enter the token name. The token that matches the entered name is then used in the Token Management Menu that is subsequently displayed.

Note: If name consists of lowercase characters it will be uppercased when processed.

Manage token from list of tokens

This option displays a list of existing tokens by name from which an entry can be chosen for use in the Token Management Menu that is subsequently displayed.

Note: If name consists of lowercase characters it will be uppercased when processed.

Key/Token Management

The Key/Token Management menus allow for the creation/deletion/management of certificates within a key database file or z/OS PKCS #11 token. Once the key database or token is created, the management of the certificates within the repository is very similar. This is illustrated throughout this topic by the key database menu, which is always on the left, and token menu, which is always on the right, being displayed side by side in the figures.

Key Management Menu/Token Management Menu

The **Key Management Menu** is displayed once the key database has been created or opened. The key database and the associated request database are opened for update and remain open until you return to the **Database Menu**.

The Token Management Menu is displayed once a z/OS PKCS #11 token has been opened.

```
       Key Management Menu

       Database: Database_name
       Expiration Date: Expiration Date

   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):
 ===>
```

```
       Token Management Menu

       Token: Token_name

       Manufacturer:  z/OS PKCS11 API
       Model:  HCR7750
       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 2. Key Management Menu/Token Management Menu

Manage Keys and Certificates

This option manages certificates with private keys. A list of key labels is displayed. Pressing the ENTER key without making a selection will display the next set of labels. Selecting one of the label numbers will display this menu:

```
      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
   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):
 ===>
```

```
      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 3. Key and Certificate Menus

Show certificate information
This option displays information about the X.509 certificate associated with the private key.

Show key information
This option displays information about the private key.

Set key as default
This option makes the current key the default key for the database.

Set certificate trust status
This option sets or resets the trusted status for the X.509 certificate. A certificate cannot be used for authentication unless it is trusted.

Note: All z/OS PKCS #11 token certificates are automatically created with the status set to trusted. Changing of the trust status is not supported for z/OS PKCS #11 token certificates.

Copy certificate and key to another database/token
This option copies the certificate and key to another token or a database. An error is returned if the certificate is already in the token/database or if the label is not unique. A certificate and key may only be copied from a FIPS mode database to another FIPS mode database. A certificate and key may not be copied from a non-FIPS mode database or a PKCS #11 token to a FIPS mode database.

Export certificate to a file
This option exports just the X.509 certificate to a file. The supported export formats are ASN.1 Distinguished Encoding Rules (DER) and PKCS #7 (Cryptographic Message Syntax)

Export certificate and key to a file
This option exports the X.509 certificate and its private key to a file. The private key is encrypted when it is written to the file. The password you select will be needed when you import the file. The supported export formats for a key database file are PKCS #12 Version 1 (obsoleted) and PKCS #12 Version 3. For z/OS PKCS #11 tokens and FIPS mode databases, the export format supported is PKCS #12 Version 3. The strong encryption option uses Triple DES to encrypt the private key while the export encryption option uses 40-bit RC2. Strong encryption is the only supported option when exporting from a FIPS database. The export file will contain the requested certificate and its certification chain.

Delete certificate and key
The certificate and its associated private key are deleted.

Change label
This option will change the label for the database record.

Create a signed certificate and key
This option will create a new certificate and associated public/private key pair. The new certificate will be signed using the certificate in the current record and then stored in either the key database file or z/OS PKCS #11 token.

DSS and DH based certificates are only supported in key database files. The key generation parameters must be compatible with the requested key type and key size.

Keys are in the same domain if they have the same set of key generation parameters. Refer to FIPS 186-2 (Digital Signature Standard) and RFC 2631 (Diffie-Hellman Key Agreement Method) for more information on the key generation parameters. The subject name and one or more subject alternate names can be specified for the new certificate.

The subject name is always an X.500 directory name while a subject alternate name can be an X.500 directory name, a domain name, an e-mail address, an IP address, or a uniform resource identifier. An X.500 directory name consists of common name, organization, and country attributes with optional organizational unit, city/locality, and state/province attributes. A domain name is one or more tokens separated by periods. An e-mail address consists of a user name and a domain name separated by '@'. An IP address is an IPv4 address (nnn.nnn.nnn.nnn) or an IPv6 address (nnnn:nnnn:nnnn:nnnn:nnnn:nnnn:nnnn:nnnn). A uniform resource identifier consists of a scheme name, a domain name, and a scheme-specific portion.

The signature algorithm used when signing the certificate is derived from the key algorithm of the signing certificate. The digest type used when signing the certificate will match the digest type used in the signature algorithm of the signing certificate. If the signing certificate's key algorithm is DSA, or the digest type used by the signing certificate is not a SHA-based digest, then the digest type used will be SHA-1. Possible signature algorithms are:

- x509_alg_sha1WithRsaEncryption
- x509_alg_sha224WithRsaEncryption
- x509_alg_sha256WithRsaEncryption
- x509_alg_sha384WithRsaEncryption
- x509_alg_sha512WithRsaEncryption
- x509_alg_dsaWithSha1

Create a certificate renewal request
This option will create a certification request using the subject name and public/private key pair from an existing certificate. The certificate request will be exported to a file in Base64 format. This file can then be sent to a certification authority for processing. The certificate returned by the certification authority can then be processed using option 5 (Receive requested certificate or a renewal certificate) on the **Key Management Menu** or **Token Management Menu**. The new certificate will replace the existing certificate.

Manage Certificates

This option manages certificates without private keys. A list of key labels is displayed. Pressing the ENTER key without making a selection will display the next set of labels. Selecting one of the label numbers will display this menu:

```
      Certificate Menu

      Label: Certificate_label_name

   1 - Show certificate information
   2 - Set certificate trust status
   3 - Copy certificate to another database
   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):
```

```
      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 4. Certificate Menus

Show certificate information
This option displays information about the X.509 certificate.

Set certificate trust status
This option sets or resets the trusted status for the X.509 certificate. A certificate cannot be used for authentication unless it is trusted.

Note: All z/OS PKCS #11 token certificates are automatically created with the status set to trusted. Changing of the trust status is not supported for z/OS PKCS #11 token certificates.

Copy certificate to another database/token
This option copies the certificate to another token or a key database. An error is returned if the certificate is already in the token/database or if the label is not unique. A certificate and key may only be copied from a FIPS mode database to another FIPS mode database. A certificate and key may not be copied from a non-FIPS mode database or a PKCS #11 token to a FIPS mode database.

Export certificate to a file
This option exports the X.509 certificate to a file. The supported export formats are ASN.1 DER (Distinguished Encoding Rules) and PKCS #7 (Cryptographic Message Syntax). The export file will contain just the requested certificate when the DER format is selected. The export file will contain the requested certificate and its certification chain when the PKCS #7 format is selected.

Delete certificate
The certificate is deleted.

Change label
This option will change the label for the certificate.

Manage Certificate Requests

This option manages certificate requests. A list of request labels is displayed. Pressing the ENTER key without making a selection will display the next set of labels. Selecting one of the label numbers will display this menu:

```
     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):
```

```
      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 5. Request Menus

Show key information
This option displays information about the private key associated with the certificate request.

Export certificate request to a file
This option exports the certificate request to a file in Base64 format. This file can then be sent to a certification authority for processing.

Delete certificate request and key
The certificate request and its associated private key are deleted.

Change label
This option will change the label for the certificate request.

Create New Certificate Request

This option will create a certificate request using either RSA encryption or DSA for the public and private keys (DSA only supported for key databases). The certificate request will be exported to a file in Base64 format. This file can then be sent to a certification authority for processing.

For key databases:
The label has a maximum length of 127 characters and is used to reference the certificate in the request database. The label will also be used when the certificate is received, so it must be unique in both the request and key databases. It must consist of characters which can be represented as 7-bit ASCII characters (letters, numbers, and punctuation) in the ISO8859-1 code page.

For tokens:
The label has a maximum length of 32 characters and is used to reference the certificate request. The label will also be used when the certificate is received, so it must be unique in the token. It must consist of characters which can be represented in the IBM1047 code page.

The subject name and one or more subject alternate names can be specified for the new certificate. The subject name is always an X.500 directory name while a subject alternate name can be an X.500 directory name, a domain name, an e-mail address, an IP address, or a uniform resource identifier. An X.500 directory name consists of common name, organization, and country attributes with optional organizational unit, city/locality, and state/province attributes. A domain name is one or more tokens separated by periods. An e-mail address consists of a user name and a domain name separated by '@'. An IP address is an IPv4 address (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 will receive the signed certificate returned by the certification authority. The certificate can be either a new or renewal certificate issued in response to a certificate request or a renewal of an existing certificate without a corresponding certificate request. If the certificate was issued in response to a certificate request, the certificate request must still be in the request database or token. If this is a renewal certificate without a certificate request, the old certificate must still be in the key database or token and must have the same issuer name and public key. If the key database or token does not contain the private key of the old certificate or contains certificates signed by the old certificate, then the subject name must also be the same when renewing the certificate.

The certificate file must contain either an ASN.1 DER-encoded sequence as defined in RFC 2459 (X.509 Public Key Infrastructure) 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.

When receiving the signed (renewal) certificates into a FIPS key database file only certificates signed with FIPS signature algorithms using FIPS-approved key sizes may be imported. It is the responsibility of the receiver to ensure that the certificate has been signed by a source meeting FIPS 140-2 criteria in order to maintain adherence to the FIPS criteria.

Create a Self-Signed Certificate

This option will create a self-signed certificate using either RSA or DSA encryption for the public and private keys and a certificate signature based on a SHA digest algorithm of the user's choice (if an RSA certificate is requested). Possible signature algorithms are:

- x509_alg_sha1WithRsaEncryption
- x509_alg_sha224WithRsaEncryption
- x509_alg_sha256WithRsaEncryption
- x509_alg_sha384WithRsaEncryption
- x509_alg_sha512WithRsaEncryption
- x509_alg_dsaWithSha1

The certificate can be created for use by a certification authority or an end user. A CA certificate can be used to sign other certificates and certificate revocation lists while an end user certificate can be used for authentication, digital signatures, and data encryption.

Note: DSA certificates are only supported in key database files.

For key databases:

The label has a maximum length of 127 characters and is used to reference the certificate in the request database. The label will also be used when the certificate is received, so it must be unique in both the request and key databases. It must consist of characters which can be represented as 7-bit ASCII characters (letters, numbers, and punctuation) in the ISO8859-1 code page.

For tokens:

The label has a maximum length of 32 characters and is used to reference the certificate request. The label will also be used when the certificate is received, so it must be unique in the token. It must consist of characters which can be represented in the IBM1047 code page.

The number of days until the certificate expires must be between 1 and 9999.

The subject name and one or more subject alternate names can be specified for the new certificate. The subject name is always an X.500 directory name while a subject alternate name can be an X.500 directory name, a domain name, an e-mail address, an IP address, or a uniform resource identifier. An X.500 directory name consists of common name, organization, and country attributes with optional organizational unit, city/locality, and state/province attributes. A domain name is one or more tokens separated by periods. An e-mail address consists of a user name and a domain name separated by '@'. An IP address is an IPv4 address (nnn.nnn.nnn.nnn) or an IPv6 address (nnnn:nnnn:nnnn:nnnn:nnnn:nnnn:nnnn:nnnn). A uniform resource identifier consists of a scheme name, a domain name, and a scheme-specific portion (for example, http://www.endicott.ibm.com/main.html).

Note: A self-signed end-entity certificate (server or client certificate) is not recommended for use in production environments and should only be used to facilitate test environments prior to production. Self-signed certificates do not imply any level of security or authenticity of the certificate because, as their name implies, they are signed by the same key that is contained in the certificate. On the other hand, certificates that are signed by a certificate authority indicate that, at least at the time of signature, the certificate authority approved the information contained in the certificate.

Import a Certificate

| This option will add the contents of the import file to a key database file or z/OS PKCS #11 token. The
| import file may contain one or more certificates without private keys. When each certificate is added to the
| key database, it is marked as trusted. The expiration date associated with each certificate cannot exceed
| February 6, 2016.

| When adding certificates from the import file to a FIPS key database file only certificates signed with FIPS
| signature algorithms using FIPS-approved key sizes may be imported. When processing a chain of
| certificates, processing of the chain will terminate if a non-FIPS certificate is encountered. Certificates
| processed prior to the failing certificate will be added to the key database file. It is the responsibility of the
| importer to ensure that the file came from a FIPS source in order to maintain meeting FIPS 140- 2 criteria.

The import file must contain either an ASN.1 DER-encoded sequence as defined in RFC 2459 (X.509 Public Key Infrastructure) or a signed data message as defined in PKCS #7 (Cryptographic Message Syntax). The data can either be the binary value or the Base64 encoding of the binary value.

If the import file is in PKCS #7 format, only the first certificate and its certification chain will be imported. The certificate subject name will be used as the label for certificates added from the certification chain. A certification chain certificate will not be added to the database or z/OS PKCS #11 token if the label is not unique or if the certificate is already in the database or z/OS PKCS #11 token.

Base64 data is in the local code page. A DER-encoded sequence must start with the encoding header '-----BEGIN CERTIFICATE-----' and end with the encoding footer '-----END CERTIFICATE-----'. A PKCS #7 signed data message must start with the encoding header '-----BEGIN CERTIFICATE-----' and end with the encoding footer '-----END CERTIFICATE-----' or start with the encoding header '-----BEGIN PKCS #7 SIGNED DATA-----' and end with the encoding footer '-----END PKCS #7 SIGNED DATA-----'.

A root certificate is a self-signed certificate and will be imported as long as the certificate is not already in the key database or z/OS PKCS #11 token.

An intermediate CA or end-entity certificate is a certificate signed by another entity. The key database or z/OS PKCS #11 token must already contain a certificate for the issuer. The certificate will not be imported if the certificate authenticity cannot be validated or if the database already contains the certificate.

An existing certificate can be replaced by specifying the label of the existing certificate. The issuer name, subject name, and subject public key in the new certificate must be the same as the existing certificate. If the existing certificate has a private key, the private key is not changed when the certificate is replaced.

Import a Certificate and a Private Key

This option imports a certificate and the associated private key and adds it to the key database or z/OS PKCS #11 token. The certificate will be marked as trusted when it is added. When importing a certificate, the expiration date cannot exceed February 6, 2106.

The import file must contain an ASN.1 DER-encoded sequence as defined in PKCS #12 (Personal Information Exchange Syntax). The data can be either the binary value or the Base64 encoding of the binary value. Base64 data is in the local code page and must start with the encoding header '-----BEGIN CERTIFICATE-----' and end with the encoding footer '-----END CERTIFICATE-----'.

A root certificate is a self-signed certificate and will be imported as long as the certificate is not already in the key database or z/OS PKCS #11 token.

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.

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: 1 <enter>

Key database /home/sufwl1/ssl_cmd/mykey.kdb created.

Press ENTER to continue.
 ===>
```

Figure 6. Creating a New Key Database

Figure 6 shows the input prompts that **gskkyman** produces when you choose **1** to create a new key database. As you can see, default choices are listed in parentheses. In the example, by pressing the Enter key at the **Enter database record length** prompt, the default of 5000 was chosen.

Note: When dealing with certificates which may be large in size or have large key sizes, for example 2048 or 4096, an initial key record length of 5000 may be required.

Note: The maximum length of the password specified for a key database file is 128 characters.

After entering the database record length, a message displays confirming that your database was created (see Figure 6). 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/sufwl1/ssl_cmd/mykey.kdb
      Expiration Date: 2008/12/02  10:11:12

   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 7. Key Management Menu for gskkyman

Figure 7 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 8 on page 187). 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 8. 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 9).

```
       Key Management Menu

       Database: /home/sufwl1/ssl_cmd/mykey.kdb
       Expiration Date: 2008/12/02  10:11:12

   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 9. Key Management Menu

To delete an existing database, from the **Database Menu**, select option **5** (see Figure 10):

```
      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 10. Deleting an Existing Key Database

You are prompted to enter the key database name that you wish to delete. Then you must enter **1** to confirm the delete, or **0** to cancel the delete. If you choose **1**, a message displays to confirm the file has been deleted.

Note: If you delete an existing key database, the associated request database and database password stash file (if existent) **will also be deleted**. It's important to note that anyone with write access to a key database can delete that database either by removing it with the **rm** command or by using **gskkyman** subcommand.

Creating a certificate to be used with a fixed Diffie-Hellman key exchange (Key Database File Only)

Create a server certificate to be used during an SSL handshake using a fixed Diffie-Hellman key exchange. Fixed Diffie-Hellman requires both sides of the exchange to be based off of the same generation parameters. In order for each side to use the same generation parameters, a key parameter file must be created to be used as input to the certificate being signed.

To create a key parameter file, from the **Database Menu**, enter **6**. First you will be asked to select the key type. Only the key types of 1024-bit DSA key or 2048-bit fixed Diffie-Hellman key are valid for use in a FIPS database. Once the key type is determined, you will be prompted to enter a key parameter file name. The file name is interpreted relative to the current directory when gskkyman is invoked. You may also specify a fully qualified file name.

```
       Database Menu

       Database: /home/sufwl1/ssl_cmd/mykey.kdb

   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 - 1024-bit DSA key
   2 - 1024-bit Diffie-Hellman key
   3 - 2048-bit Diffie-Hellman key

Select key type (press ENTER to return to menu):   2 <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 11. Creating a key parameter file to be used with Diffie-Hellman

Once the key parameter file has been created, the next step is to create the signed certificate using an existing certificate in the key database file to sign the server certificate. From the **Key Management Menu**, select **1 - Manage keys and certificates** to display the **Key and Certificate Menu**. In the **Key and Certificate Menu**, choose option **10** to create a signed certificate and key. This requires the displayed certificate to have signing capability.

Select the certificate type by choosing option **9**, user or server certificate with 1024 Diffie-Hellman key from the **Certificate Type Menu**.

Once the certificate type is determined, you will be prompted to enter:

- key parameter file created previously
- a label to uniquely identify the key and certificate within the key database
- the individual fields within the subject name
- certificate expiration (Valid expiration range is 1 to 9999 days. Default value is 365 days)
- the subject alternate names (optional).

```
           Certificate Type Menu

   1 - CA certificate with 1024-bit RSA key
   2 - CA certificate with 2048-bit RSA key
   3 - CA certificate with 4096-bit RSA key
   4 - CA certificate with 1024-bit DSA key
   5 - User or server certificate with 1024-bit RSA key
   6 - User or server certificate with 2048-bit RSA key
   7 - User or server certificate with 4096-bit RSA key
   8 - User or server certificate with 1024-bit DSA key
   9 - User or server certificate with 1024-bit Diffie-Hellman key
  10 - User or server certificate with 2048-bit Diffie-Hellman key

Select certificate type (press ENTER to return to menu):   9 <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 12. Creating a certificate to be used with Diffie_Hellman

Once the certificate is created, the next step is to determine whether the certificate should be marked as the database's default certificate. Setting the certificate as the default certificate allows the certificate to be used by the SSL APIs without having to specify its label.

Sending the Certificate Request

The certificate request file can either be transferred to another system (for example, ftp as an ASCII text file) and then transferred to the certificate authority or placed directly into a mail message sent to a certificate authority using cut-and-paste methods.

In addition to the certificate request file that is generated, a request database (.rdb) file is also created or altered. The request database file will be named the same as the key database file, except it will have an extension of .rdb. For example, a key database file of key.kdb will cause a request database file of key.rdb to be created. This request database file must be saved along with the key database in order for the response for the certificate request to be successfully processed.

Receiving the Signed Certificate or Renewal Certificate

Once a certificate is signed by the certificate authority in response to the certificate request, you must receive it into the key database or z/OS PKCS #11 token. This is for new certificates and renewal certificates.

To receive the certificate, you must store the Base64-encoded certificate in a file on the z/OS system to be read in by the **gskkyman** command. This file should be in the current working directory when **gskkyman** is started. If this file is on another working directory you will have to specify the fully qualified name.

Note: In order to receive the certificate the CA certificate must also exist in the key database or z/OS PKCS #11 token. To store a CA certificate, refer to “Importing a Certificate from a File as a Trusted CA Certificate” on page 196.

To receive a certificate issued on your behalf, from the **Key Management Menu**, see Figure 9 on page 187 and enter option **5**.

```
       Key Management Menu

       Database: /home/sufwl1/ssl_cmd/mykey.kdb
       Expiration Date: 2008/12/02  10:11:12

   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.
 ===>
```

```
       Token Management Menu

       Token: TOKENABC

       Manufacturer:  z/OS PKCS11 API
       Model:  HCR7750
       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 13. Receiving a Certificate Issued for your Request

You are prompted for the name of the file that contains the Base64-encoded certificate that was returned to you by the certificate authority in response to a previously submitted certificate request. After receiving the certificate, you press Enter to continue working with the **Key Management Menu**. Upon completion of this step and prior to the System SSL APIs using the certificate during the SSL handshake processing, you need to determine whether the certificate should be marked as the database's default certificate. Setting the certificate as the default certificate allows the certificate to be used by the SSL APIs without having to specify its label.

When receiving the signed (renewal) certificates into a FIPS key database file only certificates signed with FIPS signature algorithms using FIPS-approved key sizes may be imported. It is the responsibility of the receiver to ensure that the certificate has been signed by a source meeting FIPS 140-2 criteria in order to maintain adherence to the FIPS criteria.

Managing Keys and Certificates

Once certificates are added to the key database or z/OS PKCS #11 token, these are some common operations that can be performed with the certificates.

- Show certificate/key information
- Mark a certificate (and private key) as the default certificate for the key database or z/OS PKCS #11 token
- Export a certificate to a file, key database or z/OS PKCS #11 token
- Remove a certificate (and private key) from a key database or z/OS PKCS #11 token
- Change a certificate label

Copying a Certificate (and Private Key) to a Different Key Database or z/OS PKCS #11 Token

Once your certificates have been created, it may be necessary for you to transfer a certificate to another key database or z/OS PKCS #11 token on your system or a remote system. This transfer maybe necessary for these reasons:

- The remote system requires the signing certificate to be in its key database or z/OS PKCS #11 token for validation purposes. The certificate does not need to contain the private key information. These certificates are normally certificate authority (CA) certificates but may also be a self-signed certificate.
- The server or client certificate is being used by another application in a separate key database file or z/OS PKCS #11 token.

Note: The source key database file or z/OS PKCS #11 token and the target key database file or z/OS PKCS #11 token must exist before the certificate can be copied. If the target is a z/OS PKCS #11 token, then only certificates with RSA keys of 1024 or 2048 bits may be copied. If the target is a FIPS database, then only a FIPS database can be the source.

Copying a Certificate Without its Private Key

To copy a certificate to a different platform or to a different system without its private key (certificate validation), from the **Key Management Menu** or the **Token Management Menu**, select **1 - Manage keys and certificates** to display the **Key and Certificate List** or the **Token Key and Certificate List** respectively. Find the label of the certificate to be copied and enter the number associated with the label. In the **Key and Certificate Menu** or the **Token Key and Certificate Menu**, enter option **6** to export the certificate to a file. The **Export File Format** menu appears:

```
      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 14. 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 on receiving the certificate into the key database file or z/OS PKCS #11 token, see “Importing a Certificate from a File as a Trusted CA Certificate” on page 196). Upon successfully receiving the certificate, the certificate can now be used to validate the SSL’s partner certificate. This means that a client with the imported certificate can now validate the server’s certificate, while a server with the imported certificate can validate the client’s certificate when client authentication is requested.

You will also need to determine whether the certificate should be marked as the default certificate. Setting the certificate as the default certificate allows the certificate to be used by the SSL APIs without having to specify its label.

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:

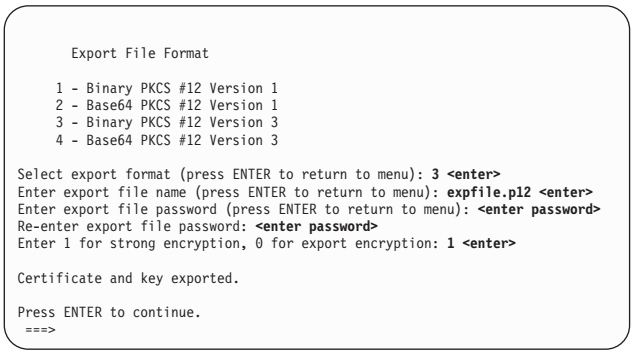

Figure 15. Copying a Certificate and Private key to a Different Key Database

The second display applies to z/OS PKCS #11 tokens.

You will then be prompted for what file format you would like for the exported certificate information.

The file format is determined by the support on the receiving system. In most cases the format to be used is Binary PKCS #12 Version 3. When the receiving system implementation is z/OS System SSL V1R2 or earlier, the selected format **must** be Binary PKCS #12 Version 1. z/OS PKCS #11 tokens only support
| Version 3 PKCS #12 export. Export from a FIPS database must be PKCS #12 Version 3 using strong
| encryption.

After selecting the export format, you will be asked for a file name and password. You then will receive a message indicating that the certificate was exported. You can now transfer this file to the system and import the certificate into the key database file or z/OS PKCS #11 token. If copying to a remote system, this file can now be transferred (in binary) to the remote system. For information on receiving the certificate into the key database file, see “Importing a Certificate from a File with its Private Key” on page 198). Upon successfully receiving the certificate, the certificate can now be used to identify the program. For example, the certificate can be used as the SSL server program’s certificate or it can be used as the SSL client program’s certificate.

Copying a Certificate and its Private Key to a Key Database on the Same System

To copy a certificate and its private key from one key database to another key database on the same
| system, you will need to know the target key database file name and password. If the source database is
| a FIPS database, then the target database must also be a FIPS database. If the source database is a
| non-FIPS database, then the target must also be a non-FIPS database. From the **Key Management Menu**, select **1 - Manage keys and certificates** to display the **Key and Certificate Menu**. Find the label of the certificate to be copied and enter the number associated with the label. From the **Key and Certificate Menu**, enter **5** to copy a certificate and key to another database:

```
      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):  5 <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 16. 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.

Copying a Certificate and its Private Key from a z/OS PKCS #11 Token on the Same System

To copy a certificate and its private key from a z/OS PKCS #11 token to another z/OS PKCS #11 token or key database file on the same system, from the **Token Management Menu**, select **1 - Manage Keys and Certificates** to display the Token Key and Certificate List. Find the label of the certificate to be copied and enter the number associated with the label. From the **Token Key and Certificate Menu** enter **5** to copy a certificate and key to another token or a key database file. If the target is a key database on the same system, you will need to know the target's file name and password.

```
       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 17. 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 17 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.

Importing a Certificate from a File as a Trusted CA Certificate

If you are using a certificate authority for generating your certificates that is not one of the default certificate authorities for which certificates are already stored in the key database, or if you are using a z/OS PKCS #11 token for which no default certificates exist, then you must import the certificate authority's certificate into your key database file or z/OS PKCS #11 token before you use the System SSL APIs. If you are using **client authentication**, then the CA certificate must be imported into the key database or z/OS PKCS #11 token of the server program. The client program's key database file or z/OS PKCS #11 token must have the CA certificate imported regardless of whether or not the SSL connection uses **client authentication**.

If you are using a self-signed certificate as the SSL server program's certificate and your SSL client program is also using the System SSL APIs, then you must import the server's self-signed certificate without its private key into the client program's key database file or z/OS PKCS #11 token.

If you are using a self-signed certificate as the SSL client program's certificate and your SSL server program is also using the System SSL APIs with client authentication requested, then you must import the client's self-signed certificate without its private key into the server program's key database file or z/OS PKCS #11 token.

If the CA certificate being imported was signed by another CA certificate, the complete chain must be present in the key database file or z/OS PKCS #11 token prior to the import.

If using a key database file, a number of well-known certificate authority (CA) certificates are stored in the key database when the key database is created. To get a certificate list, select **2 - Manage certificates** from the **Key Management Menu**. Figure 18 on page 197 and Figure 19 on page 197 contain lists of CAs

for which certificates are stored on key database creation:

```
       Certificate List

       Database: /home/sufwl1/ssl_cmd/mykey.kdb
       Expiration Date: 2008/12/02  10:11:12

   1 - VeriSign Class 1 Public Primary CA
   2 - VeriSign Class 2 Public Primary CA
   3 - VeriSign Class 3 Public Primary CA
   4 - RSA Secure Server CA
   5 - Thawte Server CA
   6 - Thawte Premium Server CA
   7 - Thawte Personal Basic CA
   8 - Thawte Personal Freemail CA
   9 - Thawte Personal Premium CA

   0 - Return to selection menu

Enter label number (ENTER for more labels, p for previous list):
 ===>
```

Figure 18. Certificate List (part 1)

```
       Certificate List

       Database: /home/sufwl1/ssl_cmd/mykey.kdb

   1 - Equifax Secure Certificate Authority
   2 - Equifax Secure eBusiness CA-1
   3 - Equifax Secure eBusiness CA-2
   4 - Equifax Secure Global eBusiness CA-1

   0 - Return to selection menu

Enter label number (ENTER to return to selection menu, p for previous list):
 ===>
```

Figure 19. Certificate List (part 2)

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/sufwl1/ssl_cmd/mykey.kdb
      Expiration Date: 2008/12/02  10:11:12

   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.
====>
```

```
      Token Management Menu

      Token: TOKENABC

    Manufacturer: z/OS PKCS11 API
    Model: HCR7750
    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 20. Importing a Certificate from a File

You will be prompted to enter the certificate file name and your choice of a unique label that will be assigned to the certificate.

Once the certificate is imported, you will receive a message indicating the import was successful. The certificate is treated as ″trusted″ so that it can be used in verifying incoming certificates. For a program acting as an SSL server, this certificate is used during the verification of a client's certificate. For a program acting as an SSL client, this certificate is used to verify the server's certificate which is sent to the client during SSL handshake processing.

When adding certificates from the import file to a FIPS key database file only certificates signed with FIPS signature algorithms using FIPS-approved key sizes may be imported. When processing a chain of certificates, processing of the chain will terminate if a non-FIPS certificate is encountered. Certificates processed prior to the failing certificate will be added to the key database file. It is the responsibility of the importer to ensure that the file came from a source meeting FIPS 140-2 criteria in order to maintain adherence to the FIPS criteria.

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 193 for more information). PKCS #12 files are password-protected to allow encryption of the private key information. If the CA certificate being imported was signed by another CA certificate, the complete chain must be present in the key database file or z/OS PKCS #11 token prior to the import. From the **Key Management Menu** or **Token Management Menu** , enter **8** to import a certificate and a private key:

```
      Key Management Menu

      Database: /home/sufwl1/ssl_cmd/anne.kdb
      Expiration Date: 2008/12/02  10:11:12

   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.
 ===>
```

```
      Token Management Menu

      Token: TOKENABC

    Manufacturer: z/OS PKCS11 API
    Model: HCR7750
    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 21. Importing a Certificate and Private Key from a File

You will be prompted to enter the certificate file name, password and your choice of a unique label to be assigned to the certificate.

Once the certificate is imported, you will receive a message indicating that import was successful. The next step is to determine whether the certificate should be marked as the database's or token's default certificate. Setting the certificate as the default certificate allows the certificate to be used by the SSL APIs without having to specify its label.

A certificate and key can be imported into a FIPS key database providing it is a PKCS #12 Version 3 with strong encryption format. When adding certificates from the import file to a FIPS key database file only certificates signed with FIPS signature algorithms using FIPS-approved key sizes may be imported. When processing a chain of certificates, processing of the chain will terminate if a non-FIPS certificate is encountered. Certificates processed prior to the failing certificate will be added to the key database file. It is the responsibility of the importer to ensure that the file came from a source meeting FIPS 140-2 criteria in order to maintain adherence to the FIPS criteria.

Chapter 7. SSL Started Task

The SSL started task (GSKSRVR) provides sysplex session cache support, dynamic trace support and notification when changing from hardware to software cryptography. The SSL started task is an optional component of System SSL and does not need to be configured and started in order to use System SSL.

The default home directory for the SSL started task is /etc/gskssl/server. A different home directory can be specified by changing the definition of the HOME environment variable in the GSKSRVR procedure. The SSL started task will read the envar file in the home directory to set the environment variables. This file is a variable-length file where each line consists of a variable name and variable value separated by '='. Trailing blanks are removed from the variable value. Blanks lines and lines beginning with '#' are ignored.

GSKSRVR Environment Variables

These environment variables are processed by the System SSL started task.

GSK_LOCAL_THREADS
: Specifies the maximum number of threads which will be used to handle program call requests from SSL applications running on the same system as the GSKSRVR started task. The default value is 5 and the minimum value is 2. The default of 5 will be used if a valid value is not specified.

GSK_SIDCACHE_SIZE
: Specifies the size of the sysplex session cache in megabytes and is between 1 and 512 with a default of 20. The default of 20 will be used if a valid value is not specified.

GSK_SIDCACHE_TIMEOUT
: Specifies the sysplex session cache entry timeout in minutes and is between 1 and 1440 with a default of 60. The default of 60 will be used if a valid value is not specified.

GSK_FIPS_STATE
: Specifies that the System SSL started task is to execute in FIPS mode. The only value supported is GSK_FIPS_STATE_ON. If any other value is specified, message GSK01054E is issued with a status code of zero, and GSKSRVR executes in non-FIPS mode.

  In order for the started task to perform sysplex session ID caching for FIPS mode application servers, the envar file must contain GSK_FIPS_STATE=GSK_FIPS_STATE_ON. If the started task executes in FIPS mode, then message GSK01057I is output to STDOUT. See Chapter 2, "System SSL and FIPS 140-2," on page 3 for setup requirements necessary to execute in FIPS mode.

  In order to have GSKSRVR execute in non-FIPS mode and only provide sysplex session ID caching for non-FIPS application servers, remove or comment out this environment variable. GSKSRVR will start in non-FIPS mode without issuing GSK01054E or GSK01057I messages.

Configuring the SSL Started Task

1. Create the home directory for the SSL started task (the default is /etc/gskssl/server)
2. Copy the sample envar file from /usr/lpp/gskssl/examples/gsksrvr.envar to /etc/gskssl/server/envar (change the directory name to match the home directory created) and modify the LANG, TZ, and NLSPATH values to meet local installation requirements.
3. Copy the sample started procedure from GSK.SGSKSAMP(GSKSRVR) to SYS1.PROCLIB(GSKSRVR)
4. Create the GSKSRVR user and associate it with the GSKSRVR started procedure. Replace 'nnnnnn' in the ADDUSER command with a non-zero value which is not assigned to another user.

```
ADDUSER GSKSRVR DFLTGRP(SYS1) NOPASSWORD OMVS(UID(nnnnnn) PROGRAM(/bin/sh) HOME(/etc/gskssl/server))

RDEFINE STARTED GSKSRVR.** STDATA(USER(GSKSRVR) GROUP(SYS1) TRUSTED)

SETROPTS RACLIST(STARTED) REFRESH
```

5. Ensure that the pdsename.SIEALNKE and CEE.SCEERUN datasets are APF-authorized and are either in the link list concatenation or are specified as a STEPLIB for the GSKSRVR procedure.
6. Optionally, set up a message processing exit to automatically start the GSKSRVR started task. The GSK.SGSKSAMP(GSKMSGXT) program is a sample message processing exit for this purpose. In order to activate the exit, add this to the appropriate MPFLSTxx member in SYS1.PARMLIB.

   ```
   BPXI004I,SUP(NO),USEREXIT(STARTSSL)
   ```

   This will start GSKSRVR when OMVS initialization is complete, assuming the GSKMSGXT program was linked as STARTSSL and placed in a LNKLST dataset.
7. Optionally, set up an automatic restart management (ARM) policy for the GSKSRVR started task if the default ARM policy values are not appropriate. The element type is SYSSSL and should be assigned to restart level 2. The element name is GSKSRVR_sysname. For example, the element name for the GSKSRVR started task on system DCESEC4 would be GSKSRVR_DCESEC4. Since the normal operating mode is to run the GSKSRVR started task on each system in the sysplex, the GSKSRVR started task will register with ARM to be restarted only if the started task fails and not if the current system fails. The TERMTYPE parameter of the ARM policy can be used to override this registration if desired.

Server Operator Commands

These operator commands are supported by the System SSL server:

STOP GSKSRVR or P GSKSRVR
Causes an orderly shutdown of the server.

MODIFY GSKSRVR,parameters or F GSKSRVR,parameters
Causes a command to be executed by the server. Some parameters are:

DISPLAY CRYPTO
Displays the available encryption algorithms, whether hardware cryptographic support is available and the maximum encryption key size. '--' will be displayed if the encryption algorithm is not available.

This command can be abbreviated as 'D CRYPTO'

DISPLAY LEVEL
Displays the current System SSL service level.

This command can be abbreviated as 'D LEVEL'

DISPLAY SIDCACHE
Displays the current and maximum data space sizes in megabytes followed by the session cache users and the number of cache entries for each user. The count will include expired cache entries until they are removed from the cache during an update to the hash list containing the expired entry. Each GSKSRVR started task maintains its own session cache for sessions created on that system. The 'DISPLAY SIDCACHE' command must be issued for each started task to display the cache entries for the entire sysplex. This can be done by issuing 'RO *ALL,F GSKSRVR,D SIDCACHE'.

This command can be abbreviated as 'D SIDCACHE'

DISPLAY XCF
Displays the status of all instances of the GSKSRVR started task in the sysplex.

This command can be abbreviated as 'D XCF'

STOP Causes an orderly shutdown of the server. This is the same as entering the ″STOP GSKSRVR″ command.

TRACE OFF
Turns off tracing for the System SSL started task.

TRACE ON,level
Turns on tracing for the System SSL started task. The trace output is written to the file specified by the GSK_TRACE_FILE environment variable or to the default trace file if the GSK_TRACE_FILE environment variable is not defined. The level value specifies the trace level. Refer to the descriptions of the GSK_TRACE and GSK_TRACE_FILE environment variables for more information about SSL tracing.

Sysplex Session Cache Support

The sysplex session cache support makes SSL server session information available across the sysplex. An SSL session established with a server on one system in the sysplex may be resumed using a server on another system in the sysplex as long as the SSL client presents the session identifier obtained for the first session when initiating the second session. A server executing in FIPS mode cannot resume a session cached in non-FIPS mode. SSL V3 and TLS V1.0 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.

In order to use the sysplex session cache, each system in the sysplex must be using the same external security manager (for example, z/OS Security Server RACF) and a userid on one system in the sysplex must represent the same user on all other systems in the sysplex (that is, userid ZED on System A has the same access rights as userid ZED on System B). The external security manager must support the RACROUTE REQUEST=EXTRACT,TYPE=ENVRXTR and RACROUTE REQUEST=FASTAUTH functions.

The sysplex session cache must be enabled for each application server that is to use the support. This can be done by defining the GSK_SYSPLEX_SIDCACHE environment variable or by calling the **gsk_attribute_set_enum()** routine to set the GSK_SYSPLEX_SIDCACHE attribute. The session information for each new SSL V3 or TLS V1.0 session created by the SSL server will then be stored in the sysplex session cache and can be referenced by other SSL servers in the sysplex. The RACF user associated with the SSL server becomes the owner of the session information. Any SSL server running with the same RACF user can access the session information. SSL servers running with a different RACF user can access the session information if they have at least READ access to the GSK.SIDCACHE.<owner> profile in the FACILITY class.

For example, session information created by RACF user APPLSRV1 can be accessed by RACF user APPLSRV2 if APPLSRV2 has READ access to the GSK.SIDCACHE.APPLSRV1 profile in the FACILITY class. These RACF commands grant this access:

```
RDEFINE FACILITY GSK.SIDCACHE.APPLSRV1 UACC(NONE)
PERMIT GSK.SIDCACHE.APPLSRV1 CLASS(FACILITY) ID(APPLSRV2) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

Component Trace Support

For information about component trace support, see *z/OS Cryptographic Services System SSL Programming*, SC24-5901.

Hardware Cryptography Failure Notification

For information about cryptographic hardware failure notification, see *z/OS Cryptographic Services System SSL Programming*, SC24-5901.

Chapter 8. Messages and Codes

This topic contains information for the various forms of messages and codes you may encounter after APAR OA26457 is installed:

- SSL Function Return Codes
- Deprecated SSL Function Return Codes
- CMS Status Codes (03353xxx)
- SSL Started Task Messages (GSK01nnn)

SSL Function Return Codes

This topic describes the SSL function return codes changed or added by APAR OA26457.

8 Certificate validation error.

Explanation: An error is detected while validating a certificate. This error can occur if a root CA certificate is not found in the key database, SAF keyring or z/OS PKCS #11 token or if the certificate is not marked as a trusted certificate or if the certificate requires an algorithm or key size that is non-FIPS while executing in FIPS mode.

User response: Verify that the root CA certificate is in the key database, SAF keyring or z/OS PKCS #11 token and is marked as trusted. Check all certificates in the certification chain and verify that they are trusted and are not expired. If the error occurred while executing in FIPS mode, check that only FIPS algorithms and key sizes are used by the certificate. If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure the latest changes are available. Collect a System SSL trace containing the error and then contact your service representative if the problem persists.

For more information, see Chapter 2, "System SSL and FIPS 140-2," on page 3.

9 Cryptographic processing error.

Explanation: An error is detected by a cryptographic function. This error may also occur if key sizes that are non-FIPS are used during an SSL handshake while operating in FIPS mode.

User response: If the error occurred while executing in FIPS mode, check that only FIPS key sizes are used. Collect a System SSL trace containing the error and then contact your service representative.

For more information, see Chapter 2, "System SSL and FIPS 140-2," on page 3.

402 No SSL cipher specifications.

Explanation: The client and server cipher specifications do not contain at least one value in common. Client and server cipher specifications may be limited depending on which System SSL FMIDs are installed. Server cipher specifications are dependent on the type of algorithms used by the server certificate (RSA, DSA and/or Diffie-Hellman), which may limit the options available during cipher negotiation. This error can also occur if no SSL protocols are enabled or if all of the enabled protocols have empty cipher specifications or if the TLS protocol is not enabled while executing in FIPS mode.

User response: Ensure that the client and the server have at least one cipher specification in common.

407 Key label does not exist.

Explanation: The supplied label or the default key is not found in the key database or the certificate is not trusted or the certificate uses algorithms or key sizes that are non-FIPS while executing in FIPS mode.

User response: Supply a valid label or define a default key in the key database or specify a label for a certificate that uses FIPS algorithms or key sizes if executing in FIPS mode.

For more information, see Chapter 2, "System SSL and FIPS 140-2," on page 3.

412 SSL protocol or certificate type is not supported.

Explanation: The SSL handshake is not successful due to an unsupported protocol or certificate type. This error can occur if there is no enabled SSL protocol shared by both the client and the server. When executing in FIPS mode, specifying the SSL V2 or SSL V3 protocol is ignored.

User response: Ensure that the desired SSL protocol is enabled on both the client and the server. Collect a System SSL trace containing a dump of the failing handshake and then contact your service representative if the problem persists.

428 **Key entry does not contain a private key.**

Explanation: The key entry does not contain a private key or the private key is not usable. This error can also occur if the private key is stored in ICSF and ICSF services are not available, if the private key size is greater than the supported configuration limit or the application is executing in FIPS mode. Certificates that are meant to represent a server or client must be connected to a SAF keyring with a USAGE value of PERSONAL and either be owned by the userid of the application or be SITE certificates. This error can occur when using z/OS PKCS #11 tokens if the userid of the application does not have appropriate access to the CRYPTOZ class.

User response: Ensure that the ICSF started task has been started prior to the application if the private key is stored in ICSF. When using z/OS PKCS #11 tokens, ensure the userid has appropriate access to the CRYPTOZ class.

If executing in FIPS mode, ensure that the certificate being used does not have its private key stored in ICSF.

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.

Deprecated SSL Function Return Codes

This topic describes the deprecated SSL function return codes changed or added with APAR OA26457.

-27 Key entry does not contain a private key.

Explanation: The key entry does not contain a private key or the private key is not usable. This error can also occur if the private key is stored in ICSF and ICSF services are not available, if the private key size is greater than the supported configuration limit or the application is executing in FIPS mode. Certificates that are meant to represent a server or client must be connected to a SAF key ring with a USAGE value of PERSONAL and either be owned by the userid of the application or be SITE certificates. This error can occur when using z/OS PKCS #11 tokens if the userid of the application does not have appropriate access to the CRYPTOZ class.

User response: Specify a key entry containing a private key value. Ensure that the ICSF started task is running if the private key is stored in ICSF. When using z/OS PKCS #11 tokens ensure the userid has appropriate access to the CRYPTOZ class.

If executing in FIPS mode, ensure that the certificate being used does not have its private key stored in ICSF.

-35 Certificate validation error.

Explanation: An error is detected while validating a certificate. This error can occur if a root CA certificate is not found in the key database, SAF keyring or z/OS PKCS #11 token or if the certificate is not marked as a trusted certificate or if the certificate requires an algorithm or key size that is non-FIPS while executing in FIPS mode.

User response: Verify that the root CA certificate is in the key database, SAF key ring or z/OS PKCS #11 token and is marked as trusted. Check all certificates in the certification chain and verify that they are trusted and are not expired. Collect a System SSL trace containing the error and then contact your service representative if the problem persists. If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure the latest changes are available.

For more information, see Chapter 2, "System SSL and FIPS 140-2," on page 3.

-36 Cryptographic processing error.

Explanation: An error is detected by a cryptographic function. This error may also occur if key sizes that are non-FIPS are used during an SSL handshake while operating in FIPS mode.

User response: If the error occurred while executing in FIPS mode, check that only FIPS key sizes are used. Collect a System SSL trace containing the error and then contact your service representative.

For more information, see Chapter 2, "System SSL and FIPS 140-2," on page 3.

-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.

CMS Status Codes (03353xxx)

This topic describes some CMS status codes changed or added with APAR OA26457.

03353003 Cryptographic algorithm is not supported.

Explanation: An X.509 cryptographic algorithm is not supported by the current level of the System SSL runtime. This error can also occur if the current operation does not support the specified cryptographic algorithm. When running in FIPS mode, this error may occur if an attempt is made to use an algorithm not supported in FIPS mode.

User response: Ensure that the cryptographic algorithm is supported for the requested operation or that it is supported if executing in FIPS mode. Upgrade the System SSL runtime if a later software level supports the cryptographic algorithm.

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.

03353034 Encryption key size is not supported.

Explanation: The encryption key size is not supported by the System SSL runtime.

User response: Refer to the System SSL information to determine which key sizes are supported. In general, when executing in non-FIPS mode, 40-bit keys and 128-bit keys are supported for RC2 and RC4, 56-bit keys are supported for DES, 168-bit keys are supported for Triple DES, and 128-bit keys and 256-bit keys are supported for AES. RSA keys must be between 512 and 4096 bits, DSS keys must be between 512 and 1024 bits, and Diffie-Hellman keys must be between 512 and 2048 bits.

When executing in FIPS mode, 168-bit keys are supported for Triple DES, and 128-bit keys and 256-bit keys are supported for AES. RSA keys must be between 1024 and 4096 bits, DSS keys must be 1024 bits, and Diffie-Hellman keys must be 2048 bits.

This error can also occur if the requested key size is not compatible with the supplied key generation parameters. Refer to the System SSL information to determine which key sizes are supported.

03353066 Generate random bytes produced duplicate output.

Explanation: The Random Number Generator has produced identical consecutive blocks of output data. If in FIPS mode, any further attempts to use System SSL will continue to fail until the application is restarted or the executing process is reinitialized.

User response: Restart the SSL application or process to reinitialize the SSL DLLs. If the problem persists, collect a System SSL trace containing the error and contact your service representative.

03353067 A Known Answer Test has failed.

Explanation: A Known Answer Test has failed to match the expected results. Any further attempts to use System SSL will continue to fail until the application is restarted or the executing process is reinitialized.

User response: Restart the SSL application or process to reinitialize the SSL DLLs. If the problem persists, collect a System SSL trace containing the error and contact your service representative.

03353068 The API is not supported.

Explanation: The API is not supported. An attempt was made to use an API that is not supported in the current mode of operation (FIPS or non-FIPS).

User response: Ensure that the API being utilized is supported in the mode in which the application is executing. If you are invoking a FIPS-only API, you will need to restart your application in FIPS mode.

03353069 Key database is not a FIPS mode database.

Explanation: While executing in FIPS mode, an attempt was made to open a key database that is non-FIPS.

User response: Specify a key database that meets FIPS 140-2 criteria if running in FIPS mode.

0335306A Key database can only be opened for update if running in FIPS mode.

Explanation: While executing in non-FIPS mode, an attempt was made to open a FIPS key database for update.

User response: To open a FIPS key database for update, you must be executing in FIPS mode.

0335306B Switching from non-FIPS mode to FIPS mode is not supported.

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.

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 will continue to fail until the application is restarted or the executing process is reinitialized.

User response: Restart the SSL application or process to reinitialize the SSL DLLs. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

SSL Started Task Messages (GSK01nnn)

This topic describes SSL started task messages added with APAR OA26457.

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: Refer to “CMS Status Codes (03353xxx)” on page 208 for information on the reported status code. Collect a System SSL trace of the failing application and contact your service representative if the error persists.

GSK01054E SSL server starting in non-FIPS mode. 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 specified for the environment variable was not GSK_FIPS_STATE_ON, consequently FIPS mode was not attempted. If the indicated CMS status code is non-zero, an attempt was made to set FIPS mode but failed.

The System SSL started task will continue to execute in non-FIPS mode. In non-FIPS mode, GSKSRVR does not provide sysplex session ID caching for FIPS mode application servers. Sysplex session ID caching is provided only for non-FIPS mode application servers.

User response: If the indicated status is zero, correct the environment variable GSK_FIPS_STATE so that it either specifies the value 'GSK_FIPS_STATE_ON' or remove the environment variable if FIPS mode is not required for the started task. If the indicated status is non-zero, refer to “CMS Status Codes (03353xxx)” on page 208 for information on the reported status code.

Collect a System SSL trace of the failing application and contact your service representative if the error persists.

GSK01057I SSL server starting in FIPS mode.

Explanation:
GSK_FIPS_STATE=GSK_FIPS_STATE_ON was specified in the envar file in the GSKSRVR home directory.

The System SSL started task has initialized successfully and will execute in FIPS mode. In FIPS mode, GSKSRVR provides sysplex session ID caching for both FIPS mode and non-FIPS mode application servers.

User response: None

Appendix. Environment Variables

These tables contain the environment variables affected by APAR OA26457.

Table 2. SSL-Specific Environment Variables

Environment Variables	Usage	Valid Values
GSK_PROTOCOL_SSLV2	Specifies whether the SSL V2 protocol is supported. The SSL V2 protocol should be disabled whenever possible since the SSL V3 protocol provides significant security enhancements. This variable has no effect when operating in FIPS mode.	A value of ″0″, ″OFF″ or ″DISABLED″ will disable the SSL V2 protocol while a value of ″1″, ″ON″ or ″ENABLED″ will enable the SSL V2 protocol.
GSK_PROTOCOL_SSLV3	Specifies whether the SSL V3 protocol is supported. This variable has no effect when operating in FIPS mode.	A value of ″0″, ″OFF″ or ″DISABLED″ will disable the SSL V3 protocol while a value of ″1″, ″ON″ or ″ENABLED″ will enable the SSL V3 protocol.

Table 2. SSL-Specific Environment Variables (continued)

Environment Variables	Usage	Valid Values
GSK_V3_CIPHER_SPECS	Specifies the SSL V3 cipher specifications in order of preference as a string consisting of 1 or more 2-character values. The SSL V3 cipher specifications are used for the SSL V3 and TLS V1 protocols. These cipher specifications are supported: 00 = No encryption or message authentication and RSA key exchange 01 = No encryption with MD5 message authentication and RSA key exchange 02 = No encryption with SHA-1 message authentication and RSA key exchange 03 = 40-bit RC4 encryption with MD5 message authentication and RSA key exchange 04 = 128-bit RC4 encryption with MD5 message authentication and RSA key exchange 05 = 128-bit RC4 encryption with SHA-1 message authentication and RSA key exchange 06 = 40-bit RC2 encryption with MD5 message authentication and RSA key exchange 09 = 56-bit DES encryption with SHA-1 message authentication and RSA key exchange 0A = 168-bit Triple DES encryption with SHA-1 message authentication and RSA key exchange 0C = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate 0D = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate 0F = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate 10 = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate	If executing in non-FIPS mode, the default is ″050435363738392F3031 32330A1613100D0915 120F0C0306020100″ when security level 3 is installed, ″0915120F0C0306020100″ otherwise. If executing in FIPS mode, the default is ″35363738392F303132 330A1613100D″ .

Table 2. SSL-Specific Environment Variables (continued)

Environment Variables	Usage	Valid Values
GSK_V3_CIPHER_SPECS (Continued)	• 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	If executing in non-FIPS mode, the default is ″050435363738392F3031 32330A1613100D0915 120F0C0306020100″ when security level 3 is installed, ″0915120F0C0306020100″ otherwise. If executing in FIPS mode, the default is ″35363738392F303132 330A1613100D″ .

Table 2. SSL-Specific Environment Variables (continued)

Environment Variables	Usage	Valid Values
GSK_V3_CIPHER_SPECS (Continued)	• 38 = 256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate • 39 = 256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate	If executing in non-FIPS mode, the default is ″050435363738392F303132330A1613100D0915120F0C0306020100″ when security level 3 is installed, ″0915120F0C0306020100″ otherwise. If executing in FIPS mode, the default is ″35363738392F303132330A1613100D″.

Index

K

L

N

P

Q

R

S

Program Number: 5694-A01

Printed in USA