z/OS

Cryptographic Services System Secure Sockets Layer Programming - OA22451 - R9

SC24-5901

z/OS

Cryptographic Services System Secure Sockets Layer Programming - OA22451 - R9

SC24-5901

Contents

Figures

Tables

Chapter 1. Introduction

This book update contains alterations to information previously presented in *z/OS System Secure Sockets Layer Programming*, SC24-5901-06, which supports z/OS Version 1 Release 9.

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

The updates relate to changes made to the System SSL product by the application of APAR OA22451, which introduces full support for creating and using X.509 certificates that make use of the SHA-2 suite of digest algorithms in the certificate signature. In particular, APAR OA22451 introduces support for the generation of SHA-224, SHA-256, SHA-384 and SHA-512 digests in hardware or software, and adjusts System SSL, the gskkyman utility and the CMS API set to take advantage of the new function.

This book contains updates for the following chapters:

- Chapter 3. Using Hardware Cryptographic Features with System SSL with details of SHA-2 hardware support on the latest available hardware.
- Chapter 7. Certificate Management Services (CMS) API Reference with 4 new APIs and adjustments to existing APIs to support the new function
- Chapter 9. Certificate/Key Management with changes to the gskkyman utility to allow creation of SHA-2 based certificates
- Chapter 12. Messages and Codes with changes and additions to CMS status codes
- Appendix A. Environment Variables with changes to the GSK_HW_CRYPTO environment variable

Chapter 2. Certificate/Key Management

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 1. 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.

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, 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. 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.

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.

Creating a Self-Signed Server or Client Certificate

If your organization does not use a certificate authority (within the organization or outside the organization), a self-signed certificate can be generated for use by the program acting as an SSL server or client. In addition, since root CA certificates are also self signed certificates that are permitted to be used to sign other certificates (certificate requests), these procedures can also be used to create a root CA certificate. See *z/OS Cryptographic Services System SSL Programming*.

Programs acting as SSL servers (i.e. acting as the server side of the SSL handshake protocol) must have a certificate to use during the handshake protocol. A program acting as an SSL client requires a certificate when the SSL server requests client authentication as part of the SSL handshake.

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

Note: gskkyman supports the creation of X.509 Version 3 certificates.

When creating a self-signed certificate to be used to identify a server or client, from the **Key Management Menu** or **Token Management Menu**, enter **6**. You will be prompted for a number of items to define the certificate. First you will be asked to select the type of certificate to be created.

```
      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): 6 <enter>
 ===>
```

```
      Token Management Menu

      Token: TOKEN1

      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): 6 <enter>
```

Figure 2. Creating a Self-Signed Certificate

Certificates that are intended to be used directly by a server or client are considered to be end-user certificates. Certificates intended to be used to sign other certificates are considered to be CA certificates. RSA key certificates are the most common. DSA key certificates represent certificates that follow the FIPS-186 government standard. The larger the key size, the more secure the generated key will be. The most commonly used size is 1024. Note that CPU usage increases as the key size increases. For example, CPU usage will increase by a factor of 6 when the key size is doubled.

| If an RSA-based certificate is selected, you will be prompted to select the digest type for the signature
| algorithm from a list of SHA-based digest types, which includes SHA-1, SHA-224, SHA-256, SHA-384 and
| SHA-512. SHA digests range from 160 bits in length (SHA-1) to 512 bits (SHA-512), and provide
| increasing security accordingly. However, care should be taken to ensure that peer SSL implementations

support the chosen digest type. The default, if you press enter, is SHA-1. Figure 3 is an example using the default. Previous releases of gskkyman created certificates using the SHA-1 digest type.

Once the certificate type is determined, you will be prompted to enter:

- a label to uniquely identify the key and certificate within the key database
- the individual fields within the subject name
- certificate expiration. The valid expiration range is 1 to 9999 days. The default value is 365 days.
- the subject alternate names (optional)

Figure 3 shows the creation of a self-signed certificate to be used as a server or client certificate in a key database file or z/OS PKCS #11 token. If creating a server or client certificate in a z/OS PKCS #11 token, the certificate type list will consist of supported options 1, 2, 5 and 6.

```
      Certificate Type

   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

Select certificate type (press ENTER to return to menu): 5 <enter>

                  Signature Digest Type

   1 - SHA-1
   2 - SHA-224
   3 - SHA-256
   4 - SHA-384
   5 - SHA-512

Select digest type (default SHA-1): <enter>
Enter label (press ENTER to return to menu): Server Cert <enter>
Enter subject name for certificate
  Common name (required): My Server Certificate <enter>
  Organizational unit (optional): ID <enter>
  Organization (required): IBM <enter>
  City/Locality (optional): Endicott <enter>
  State/Province (optional): NY <enter>
  Country/Region (2 characters - required): US <enter>
Enter number of days certificate will be valid (default 365): 244 <enter>
Enter 1 to specify subject alternate names or 0 to continue: 0 <enter>

Please wait .....

Certificate created.

Press ENTER to continue.
 ===>
```

Figure 3. Creating a Self-Signed Certificate

Once the certificate is created, the next step is to determine whether the certificate should be marked as the database's or z/OS PKCS #11 token's default certificate. Setting the certificate as the default certificate allows the certificate to be used by the SSL APIs without having to specify its label. For more information on setting the default certificate, see *z/OS Cryptographic Services System SSL Programming*.

In order for the SSL handshake to successfully validate the use of the self-signed certificates, the partner application needs to know about the signer of the certificate. For self-signed certificates, this means the

self-signed certificate must be imported into the partner's database or z/OS PKCS #11 token. For more information on importing certificates, see *z/OS Cryptographic Services System SSL Programming*.

gskkyman Command Line Mode Examples

Command mode is entered when the **gskkyman** command is entered with parameters. The requested token/database function will be performed and then the command will exit.

- Store the database password in the stash file

  ```
  gskkyman -s -k filename
  ```

  The database password is masked and written to the key stash file. The file name is the same as the key database file name but has an extension of '.sth'. You will be prompted for the key database file name if the '-k' option is not specified. The '-t' option is invalid for the '-s' function.

- Export a certificate and the associated private key

  ```
  gskkyman -e -k filename -l label -p filename
  ```

  The certificate and associated private key identified by the record label are exported to a file in PKCS #12 Version 3 format using strong encryption. The default key will be exported if the '-l' option is not specified. You will be prompted for the key database file name if the '-k' and the '-t' option is not specified. You will be prompted for the export file name if the '-p' option is not specified.

- Import a certificate and associated private key

  ```
  gskkyman -i -t token-name -l label -p filename
  ```

  A certificate and associated private key are imported from a file in PKCS #12 format. You will be prompted for the label if the '-l' option is not specified. You will be prompted for the key database file name if the '-k' and the '-t' option is not specified. You will be prompted for the import file name if the '-p' option is not specified.

- Create a signed certificate for a certificate request

  ```
  gskkyman -g -x days -cr filename -ct filename -k filename -l label -ca -ic
  ```

  The certificate request identified by the -cr parameter is processed and a signed certificate is created and written to the certificate file identified by the -ct parameter. The -x parameter specifies the number of days until the certificate expires and defaults to 365 days. The certificate is signed using the default key if the -l parameter is not specified. You will be prompted for the key database file name if the '-k' option is not specified. You will be prompted for the certificate request file name if the '-cr' option is not specified. You will be prompted for the signed certificate file name if the '-ct' option is not specified.

  The signed certificate will be an end user certificate unless the -ca option is specified. A certification authority certificate will have basic constraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have basic constraints and key usage extensions which allow the certificate to be used for authentication, digital signatures, and data encryption (a DSA key cannot be used for data encryption).

  Any certificate can be used to sign the new certificate as long as the certificate has a private key, the basic constraints certificate extension (if present) has the CA indicator set, and the key usage certificate extension (if present) allows signing certificates. However, depending upon how the new certificate is subsequently used, it may fail the validation checking if the signing certificate is not a valid certification authority certificate.

  The signing algorithm that will be used to sign the new certificate is derived from the key algorithm of the signing certificate and the most secure and compatible SHA-based hash in use in either the signing certificate or the certificate request. Possible signing algorithms are:

  - x509_alg_sha1WithRsaEncryption
  - x509_alg_sha224WithRsaEncryption
  - x509_alg_sha256WithRsaEncryption
  - x509_alg_sha384WithRsaEncryption
  - x509_alg_sha512WithRsaEncryption
  - x509_alg_dsaWithSha1

The certificate file will contain the generated X.509 certificate in DER-encoded Base64 format if the -ic option is not specified. The certificate file will contain the generated X.509 certificate and the certification chain certificates as a PKCS #7 message in Base64 format if the -ic option is specified.

Chapter 3. Certificate Management Services (CMS) API Reference

This topic describes the Certificate Management Services (CMS) APIs. These APIs can be used to create/manage your own key database files in a similar function to the SSL gskkyman utility, use certificates stored in the key database file or key ring for purposes other than SSL, and basic PKCS #7 message support.

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

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_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*.

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}

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_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}

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.

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_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}

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_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.

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}

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.

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_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_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.

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}

gsk_create_database_signed_certificate()

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

Format

```
#include <gskcms.h>

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

Parameters

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

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

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

key_algorithm
Specifies the certificate key algorithm.

key_size
Specifies the certificate key size in bits.

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

signature_algorithm
Specifies the signature algorithm used for the certificate signature.

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

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

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

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

Results

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

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

[CMSERR_BACKUP_EXISTS]
The backup file already exists.

[CMSERR_BAD_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.

Usage

The **gsk_create_database_signed_certificate()** 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 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}

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_idPublicNumber
Diffie-Hellman (DH) - {1.2.840.10046.2.1}

RSA keys

- Can be used for both CA certificates and end user certificates
- Key size between 512 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 between 512 and 1024 bits rounded up to a multple of 64
- 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 between 512 and 2048 bits rounded up to a multiple of 64
- 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 signing 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_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_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).

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_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.

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}

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.

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).

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_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.

Usage

The **gsk_create_signed_certificate_record()** routine will generate 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 *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}

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. The subject key identifier from the signing certificate 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 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.

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_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).

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_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 2459 (X.509 Public Key Infrastructure). 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}

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 subject key identifier from the signing certification will be
| copied to the signed CRL as the authority key identifier.

| 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_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}

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.

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}

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.

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}

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.

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}

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.

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_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. 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}

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_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. 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}

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_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. 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_md5Sha1WithRsaEncryption
RSA encryption with combined MD5 and SHA-1 digests

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

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

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_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.

Chapter 4. Using Hardware Cryptographic Features with System SSL

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

In order for System SSL to use the hardware support provided through ICSF, the ICSF started task must be running prior to the application and the application userid must be authorized to the appropriate resources in the RACF CSFSERV class (when the class is defined), either explicitly or through a generic resource profile. See Table 1 and Table 2 for supported crypto algorithms and required RACF classes for each hardware processor. In addition to the CSFSERV class, the application userid needs access to the RACF CSFKEYS class when SAF key rings are being used and the application's certificate keys are stored in ICSF. For more information on access to CSFKEYS, see the RACDCERT command in *z/OS Security Server RACF Command Language Reference*.

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

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

CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement, feature 3863, provides for clear key DES and TDES instructions. On the z9 EC, z9 BC and System z10 EC, this feature also includes clear key AES for 128-bit keys. ICSF is not required in order for System SSL to use the CPACF.

Table 1 describes the cryptographic function exploited through ICSF and the required CSFSERV resource class accesses. For more information, refer to Controlling Who Can Use Cryptographic Keys and Services in *z/OS Cryptographic Services ICSF Administrator's Guide*.

Table 1. Hardware cryptographic functions exploited by System SSL

Function	**z800, z900**	**z890, z990, z9 EC, z9 BC and System z10 EC**
DES	CSFCKI, CSFDEC, CSFENC	
TDES	CSFCKM, CSFDEC, CSFENC	
PKA (RSA) Decrypt	CSFPKD	CSFPKD
PKA (RSA) Encrypt	CSFPKE	CSFPKE
Digital Signature Generate	CSFPKI, CSFDSG	CSFPKI, CSFDSG
Digital Signature Verify	CSFDSV	CSFDSV

Table 1 describes the hardware cryptographic functions that will be exploited by System SSL.

Note: Cryptographic function being performed against RSA keys with sizes greater than 2048 will be done in software. Encrypt/decrypt using the CCF is independent on the size of the data. For DES, when the data is less than or equal to 256 bytes, DES is performed in software. For TDES, when the data is less than or equal to 64 bytes, TDES is performed in software.

System SSL contains software implementations for algorithms in Table 2.

Table 2. Hardware cryptographic functions used by System SSL

	z800, z900	z890, z990		z9 EC, z9 BC and System z10 EC			
Algorithm	**CCF(s)**	**CPACF**	**PCIXCC/ CEX2C**	**CPACF (z9)**	**CPACF (System z10 EC)**	**CEX2C**	**CEX2A**
DES	X	X		X	X		
TDES	X	X		X	X		
AES 128-bit				X	X		
AES 256-bit					X		
SHA-1		X		X	X		
SHA-2 (SHA-224)				X	X		
SHA-2 (SHA-256)				X	X		
SHA-2 (SHA-384)					X		
SHA-2 (SHA-512)					X		
PKA (RSA) Decrypt	X		X			X	X
PKA (RSA) Encrypt	X		X			X	X
Digital Signature Generate	X		X			X	
Digital Signature Verify	X		X			X	X

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

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

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

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

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

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

Chapter 5. Messages and Codes

CMS Status Codes (03353xxx)

0335303D Digest data size is not correct.

Explanation: The length of the digest data is not correct. Digest data size by algorithm is:

- MD2 – 16 bytes
- MD5 – 16 bytes
- SHA-1 – 20 bytes
- SHA-224 – 28 bytes
- SHA-256 – 32 bytes
- SHA-384 – 48 bytes
- SHA-512 – 64 bytes

User response: Verify that the data has not been truncated. Contact your service representative if the error persists.

03353064 Digest type and key type are incompatible.

Explanation: The specified digest algorithm and the key algorithm are incompatible.

User response: Specify a digest algorithm that is compatible with the signing key algorithm.

Chapter 6. Environment Variables

These tables contain all the environment variables used by the system application and read during the startup of the application.

Table 3. SSL-Specific Environment Variables

Environment Variables	Usage	Valid Values
GSK_HW_CRYPTO	Specifies whether the hardware cryptographic support will be used. Note that ICSF (Integrated Cryptographic Service Facility) must be configured and running in order for System SSL to use the hardware cryptographic support. SHA-1, SHA-2, DES, Triple DES and AES hardware functions can be used without ICSF if the zArchitecture message-security assist is installed. For more information on hardware cryptographic support, refer to Chapter 4, "Using Hardware Cryptographic Features with System SSL," on page 69. Selected hardware cryptographic functions can be disabled by setting the appropriate bits to zero in the GSK_HW_CRYPTO value. The corresponding software algorithms will be used when a hardware function is disabled. These bit assignments are defined: 1 = SHA-1 digest generation 2 = 56-bit DES encryption/decryption 4 = 168-bit Triple DES encryption/decryption 8 = Public key encryption/decryption 16 = AES 128-bit encryption/decryption 32 = SHA-256 digest generation 64 = AES-256-bit encryption/decryption 128 = SHA-224 digest generation 256 = SHA-384 digest generation 512 = SHA-512 digest generation **Note:** If a hardware function bit is set on and the hardware function is unavailable, processing will take place in software.	A value of '0' will disable the use of the hardware support while a value of '65535' will enable the use of the hardware support. Available hardware support will be used if this environment variable is not defined.

Index

Program Number: 5694-A01

Printed in USA