z/OS

Cryptographic Services System Secure Sockets Layer Programming PKCS#12 Certificate Store Support - APAR OA45216 Documentation for V1R13

Contents

Chapter 1. Overview

This document update describes PKCS#12 certificate store support and contains alterations to information previously presented in *z/OS Cryptographic Services System Secure Sockets Layer Programming*, SC24-5901-11.

The preceding book documents capabilities provided in support of z/OS Version 1 Release 13.

Technical changes or additions related to PKCS#12 certificate store support in this document update are indicated by a vertical line to the left of the change.

These updates relate to the enhancements made to the z/OS Cryptographic Services product by the application of APAR OA45216.

Chapter 2. Updates to Chapter 2: How System SSL works for secure socket communication

System SSL application overview

Figure 1 on page 5 describes the basic structure of the elements that are needed in your System SSL source program.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

In addition to using the previous SSL programming interfaces in an application, an application is not complete until a key database is available for use by the SSL application. The key database contains certificate information and is a z/OS UNIX System Services file that is built and managed using the **gskkyman** utility. In addition to key database files, a PKCS #12 file, a SAF key ring or a z/OS PKCS #11 token can be utilized for certificate information.

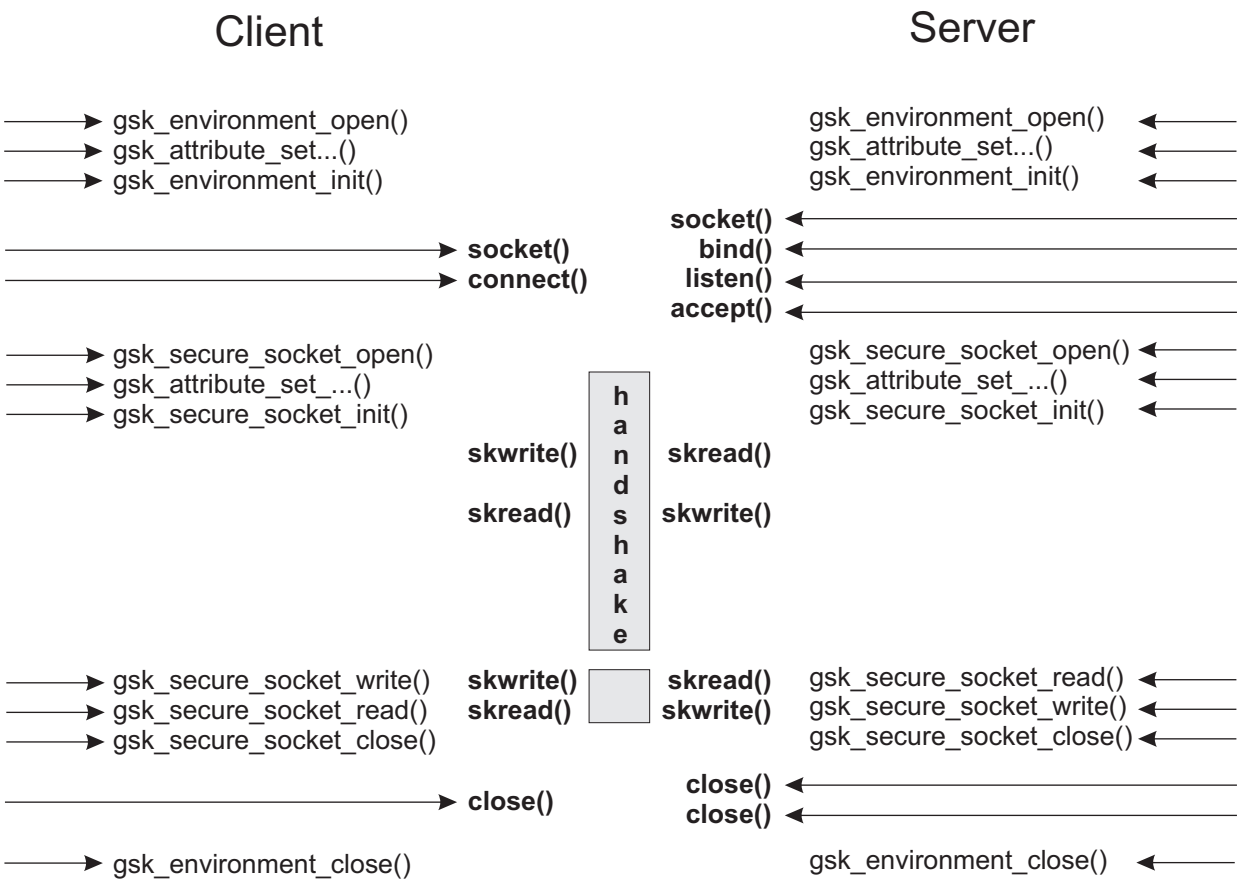

Figure 1. Sockets programming model using System SSL

Chapter 3. Updates to Chapter 4: System SSL and FIPS 140-2

Certificate stores

To use FIPS mode, certificates can be stored in either a SAF key ring, PKCS #11 token, or a FIPS mode key database.

SAF key rings and PKCS #11 tokens

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

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

Key database files

To use a key database in FIPS mode, it must be created as a FIPS mode database. Key databases that are created through **gskkyman** not explicitly specifying FIPS during creation, or created through an application not executing in FIPS mode, cannot be used by an application executing in FIPS mode. To create a FIPS mode key database using the **gskkyman** utility. To create a FIPS mode key database using the Certificate Management Services API, the application must start in FIPS mode.

The following are key points when using FIPS key databases:

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

The **gskkyman** utility automatically detects when a FIPS mode key database is opened, and executes in FIPS mode. This ensures that only certificates or certificate requests that meet the FIPS mode requirements may be added to the key database.

PKCS #12 files

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

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

Chapter 4. Updates to Chapter 5: Writing and building a z/OS System SSL application

Writing and building a z/OS System SSL application

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

In Version 1 Release 2 of z/OS, a new set of functions were added that superseded some functions from previous System SSL releases. The functions that were superseded are referred to collectively as "the deprecated SSL interface". It is suggested that new application programs do not use the deprecated SSL interface.

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

In addition to writing the SSL applications, you must have a certificate repository available for the application. The certificate repository can be a key database file, PKCS #12 file, PKCS #11 token, or SAF key ring. For SAF key rings, see the RACDCERT command information in *z/OS Security Server RACF Command Language Reference* for more information.

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

Create an SSL environment

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

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

gsk_attribute_set...()
: Sets environment attributes such as:

- The SSL protocol version to be used: SSL Version 2.0, SSL Version 3.0, TLS Version 1.0, TLS Version 1.1, and/or TLS Version 1.2.
- The key database to be used. (key database file, PKCS #12 file, SAF key ring or z/OS PKCS #11 token)
- The password for the key database. This can be specified directly by the application or by using a stashed password file.

  Note: When using SAF key rings or z/OS PKCS #11 tokens, the password and stash file must not be specified.
- The amount of time the SSL session identifier information is valid. By using already negotiated and agreed to SSL session identifier information, System SSL can reduce the amount of data exchanged during the SSL handshake that occurs during the **gsk_secure_socket_init()** call.

gsk_environment_init()
Initializes the SSL environment.

This example code illustrates how to call the environment layer programming interface from a client or server System SSL program. In this example, TLS Version 1.0 support is requested, /keyring/key.kdb is the key database that is used, the password for the key database is "password", and default values are taken for the remaining SSL environment variable attributes.

```
gsk_handle env_handle;
int      rc;

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

/* set environment attributes */
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_SSLV2, GSK_PROTOCOL_SSLV2_OFF);
                                  /* By default, SSL V2 protocol is set on */
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_SSLV3, GSK_PROTOCOL_SSLV3_OFF);
                                  /* By default, SSL V3.0 protocol is set on */
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1, GSK_PROTOCOL_TLSV1_ON);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1_1, GSK_PROTOCOL_TLSV1_1_OFF);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1_2, GSK_PROTOCOL_TLSV1_2_OFF);
rc = gsk_attribute_set_buffer(env_handle, GSK_KEYRING_FILE, "/keyring/key.kdb",0);
rc = gsk_attribute_set_buffer(env_handle, GSK_KEYRING_PW, "password",0);

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

This example code illustrates how to create an SSL environment for a server System SSL program supporting TLS Version 1.0, TLS Version 1.1, and TLS Version 1.2.

```
gsk_handle env_handle;
int     rc;

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

/* set environment attributes */
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_SSLV2, GSK_PROTOCOL_SSLV2_OFF);
                                   /* By default, SSL V2.0 protocol is set on */
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_SSLV3, GSK_PROTOCOL_SSLV3_OFF);
                                   /* By default, SSL V3.0 protocol is set on */
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1, GSK_PROTOCOL_TLSV1_ON);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1_1, GSK_PROTOCOL_TLSV1_1_ON);
                                   /* By default, TLS V1.1 protocol is set off */
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1_2, GSK_PROTOCOL_TLSV1_2_ON);
                                   /* By default, TLS V1.2 protocol is set off */
rc = gsk_attribute_set_buffer(env_handle, GSK_KEYRING_FILE, "/keyring/key.kdb",0);
rc = gsk_attribute_set_buffer(env_handle, GSK_KEYRING_PW, "password",0);

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

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

When the System SSL program successfully creates the SSL environment, it must now perform the steps that are needed to allow the program to communicate with a peer program. The exact sockets and System SSL calls required to allow the program to communicate differ depending on whether the program is a client or a server.

Building a z/OS System SSL application

1. Write the System SSL source program.
2. Compile your System SSL source program using the DLL compiler option.
3. Include the `/usr/lib/GSKSSL.x` or `/usr/lib/GSKSSL64.x` sidedeck in the prelink or bind step input.

   If using the Certificate Management APIs, include either the `/usr/lib/GSKCMS31.x` or `/usr/lib/GSKCMS64.x` sidedeck in the prelink or bind step input.
4. Build a key database file or z/OS PKCS #11 token using the **gskkyman** utility, create a SAF key ring or PKCS #11 token using the RACDCERT command, or utilize an existing PKCS #12 file. The name of the key database file, PKCS #12 file, z/OS PKCS #11 token, or SAF key ring must match the name you specified as the GSK_KEYRING_FILE on the **gsk_attribute_set_buffer()** API. For key database files, you need to specify either the password associated with the key file or the stash file name. For PKCS #12 files, you need to specify the password associated with the file. The password must match the password specified on GSK_KEYRING_PW on the **gsk_attribute_set_buffer()** API or must be set to NULL if using a SAF key ring or z/OS PKCS #11 token. Note that the password is case-sensitive.

Running a z/OS System SSL application

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

1. Ensure that pdsename.SIEALNKE, the PDSE that contains the System SSL DLLs, is in the MVS™ search order. If it is not in the linklist or LPA, you can use the STEPLIB DD statement in your JCL or the STEPLIB environment variable in the shell. For example, in the z/OS shell, issue this command:

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

Note:

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

Chapter 5. Updates to Chapter 7: API reference

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

gsk_attribute_get_buffer()

Gets the value of an attribute buffer.

Format

```
#include <gskssl.h>

gsk_status gsk_attribute_get_buffer (
                                     gsk_handle         ssl_handle,
                                     GSK_BUF_ID         buffer_id,
                                     const char **      buffer_value,
                                     int *              buffer_length)
```

Parameters

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

buffer_id
: Specifies the buffer identifier.

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

buffer_length
: Returns the length of the buffer value.

Results

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

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

[GSK_INVALID_HANDLE]
: The handle is not valid.

[GSK_INVALID_STATE]
: The handle is closed.

Usage

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

These buffer identifiers are supported:

GSK_CLIENT_ECURVE_LIST
: Returns the list of elliptic curve specifications supported by the client as a string consisting of 4-character decimal values.
GSK_CLIENT_ECURVE_LIST may be specified for an SSL environment or

an SSL connection. The elliptic curve specifications are used by the client to guide the server as to which elliptic curves can be used when using cipher suites that use Elliptic Curve Cryptography for the TLS V1.0 or higher protocols.

GSK_CONNECT_CIPHER_SPEC
Returns the cipher specification selected for an initialized connection. When using the SSL V2 protocol the cipher specification will be returned as a single character. For other protocols the cipher specification may be returned as either a 2-character or 4-character cipher depending on the setting in GSK_V3_CIPHERS.

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

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

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

GSK_KEYRING_PW
Returns the password for the key database or PKCS #12 file. A NULL address will be returned after the environment is initialized. GSK_KEYRING_PW may be specified only for an SSL environment.

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

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

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

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

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

GSK_SNI_LIST
Returns the address of a list of server names passed to the server by the client for use during server name indication callback routine. Server name

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

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

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

GSK_V2_CIPHER_SPECS
Returns the SSL V2 cipher specifications as a string consisting of 1-character values. GSK_V2_CIPHER_SPECS may be specified for an SSL environment or an SSL connection.

GSK_V3_CIPHER_SPECS
Returns the SSL V3 cipher specifications as a string consisting of 2-character values. GSK_V3_CIPHER_SPECS may be specified for an SSL environment or an SSL connection. The SSL V3 cipher specifications are used for the SSL V3, TLS V1.0, or higher protocols.

GSK_V3_CIPHER_SPECS_EXPANDED
Returns the SSL V3 cipher specifications as a string consisting of 4-character values. GSK_V3_CIPHER_SPECS_EXPANDED may be specified for an SSL environment or an SSL connection. The SSL V3 cipher specifications are used for the SSL V3, TLS V1.0, and higher protocols.

gsk_attribute_set_buffer()

Sets the value of an attribute buffer.

Format

```
#include <gskssl.h>

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

Parameters

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

buffer_id
: Specifies the buffer identifier.

buffer_value
: Specifies the buffer value.

buffer_length
: Specifies the buffer length. Specify 0 for this parameter if the buffer value is a null-delimited character string.

Results

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

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

[GSK_ATTRIBUTE_INVALID_LENGTH]
: The buffer length is not valid.

[GSK_INSUFFICIENT_STORAGE]
: Insufficient storage is available.

[GSK_INVALID_HANDLE]
: The handle is not valid.

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

Usage

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

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

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

These buffer identifiers are supported:

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

Only NIST recommended curves are able to be specified for the attribute. To use Brainpool standard curves for an SSL connection, the buffer must be reinitialized to NULL using either **gsk_attribute_set_buffer()** or the GSK_CLIENT_ECURVE_LIST environment variable.

GSK_KEYRING_FILE
Specifies the name of the key database file, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. A key database or PKCS #12 file is used if a database password is defined using either an environment variable or the **gsk_attribute_set_buffer()** routine. When a stash file is defined, a key database file is used. Otherwise, a SAF key ring or z/OS PKCS #11 token is used. GSK_KEYRING_FILE may be specified only for an SSL environment.

The SAF key ring name is specified as "userid/keyring". The current user ID is used if the user ID is omitted. The user must have READ access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by the user. The user must have UPDATE access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by another user.

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

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

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

GSK_KEYRING_PW
Specifies the password for the key database or PKCS #12 file. GSK_KEYRING_PW may be specified only for an SSL environment.

GSK_KEYRING_STASH_FILE
Specifies the name of the key database password stash file. The stash file name always has an extension of ".sth" and the supplied name is changed if it does not have the correct extension. The GSK_KEYRING_PW value is

used instead of the GSK_KEYRING_STASH value if it is also specified. GSK_KEYRING_STASH_FILE may be specified only for an SSL environment.

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

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

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

GSK_TLS_SIG_ALG_PAIRS
Specifies the list of hash and signature algorithm pair specifications that are supported by the client or server as a string consisting of 1 or more 4-character values in order of preference for use.
GSK_TLS_SIG_ALG_PAIRS may be specified for an SSL environment or an SSL connection. The signature algorithm pair specifications are sent by either the client or server to the session partner to indicate which signature/hash algorithm combinations are supported for digital signatures. Signature algorithm pair specification only has relevance for sessions using TLS V1.2 or higher protocols.

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

GSK_V2_CIPHER_SPECS
Specifies the SSL V2 cipher specifications as a string consisting of 1 or more 1-character values. GSK_V2_CIPHER_SPECS may be specified for an SSL environment or an SSL connection.

GSK_V3_CIPHER_SPECS
Specifies the SSL V3 cipher specifications as a string consisting of 1 or more 2-character values. GSK_V3_CIPHER_SPECS may be specified for an SSL environment or an SSL connection. The SSL V3 cipher specifications are used for the SSL V3, TLS V1.0, or higher protocols.

GSK_V3_CIPHER_SPECS_EXPANDED
Specifies the SSL V3 cipher specifications as a string consisting of 1 or more 4-character values. GSK_V3_CIPHER_SPECS_EXPANDED may be specified for an SSL environment or an SSL connection. The SSL V3 cipher specifications are used for the SSL V3, TLS V1.0, or higher protocols. Applications wanting to use cipher suites that use Elliptic Curve

Cryptography must set an appropriate cipher specification in GSK_V3_CIPHER_SPECS_EXPANDED.

gsk_attribute_set_callback()

Sets an SSL callback.

Format

```
#include <gskssl.h>

gsk_status gsk_attribute_set_callback (
                                        gsk_handle          ssl_handle,
                                        GSK_CALLBACK_ID     callback_id,
                                        void *              callback)
```

Parameters

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

callback_id
: Specifies the callback identifier.

callback
: Specifies the address of the callback parameter.

Results

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

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

[GSK_ATTRIBUTE_INVALID_PARAMETER]
: The attribute parameter value is not valid.

[GSK_INVALID_HANDLE]
: The handle is not valid.

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

Usage

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

These callback identifiers are supported:

GSK_CLIENT_CERT_CALLBACK
: Indicates that the application is providing a routine to be used during a full handshake to prompt a client user to select a certificate from a list during the client authentication process. The *callback* parameter is the address of this routine. The exit routine can obtain the user data address by calling the **gsk_attribute_get_buffer()** routine. The **gsk_attribute_set_buffer()** routine should be called to set the selected key

label before returning from the callback routine. The function return value should be 0 if a key label has been set or GSK_ERR_NO_CERTIFICATE if no client certificate is to be used. GSK_CLIENT_CERT_CALLBACK can be specified only for an SSL environment.

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

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

GSK_IO_CALLBACK

Indicates that the application is providing the routines to perform read, write, and control functions. The *callback* parameter is the address of a *gsk_iocallback* structure. Each entry in the structure overrides the corresponding SSL runtime routine. A NULL entry will cause the current callback routine to be used or the SSL runtime routine will be used if there is no callback routine. GSK_IO_CALLBACK can be specified for an SSL environment or an SSL connection.

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

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

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

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

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

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

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

```
int io_setsocketoptions (
                          int        fd,
                          int        cmd,
                          char *     user_data)
```

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

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

GSK_SESSION_RESET_CALLBACK

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

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

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

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

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

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

GSK_SID_CACHE_CALLBACK

Indicates that the application is providing the routines to maintain the session identifier cache. The callback parameter is the address of a gsk_sidcache_callback structure. GSK_SID_CACHE_CALLBACK can be specified only for an SSL environment and will be used only for SSL servers (the internal cache is always used for SSL clients).

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

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

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

```
gsk_data_buffer * Put (
                      gsk_data_buffer *         ssl_session_data,
                      const unsigned char *     session_id,
                      unsigned int              session_id_length,
                      gsk_sslversion            ssl_version)
```

The routine specified by the *Delete* entry is called to remove an entry from the session identifier cache. The *session_id* parameter is the session identifier, the *session_id_length* parameter is the length of the session identifier, and the *ssl_version* parameter is the SSL protocol version number (GSK_SSLVERSION_V2 or GSK_SSLVERSION_V3).

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

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

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

GSK_SNI_CALLBACK

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

provided by the client by calling the **gsk_attribute_get_buffer()** routine. The **gsk_attribute_set_buffer()** routine should be called to set the selected key label before returning from the callback routine.

The callback routine cannot enforce the required use of the server name indication extension. The failure to select a key label causes a fatal UNRECOGNIZED_NAME alert. To enforce such actions with the callback routine the user must set the GSK_TLS_EXTID_SNI_SERVER_LABELS extension by calling the **attribute_set_tls_extension()** routine. The required and unrecognized_name_fatal fields of the extension must be set appropriately to achieve the requested outcome, although the serverKeyLabel list may be empty.

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

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

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

gsk_attribute_set_enum()

Sets an enumerated value.

Format

```
#include <gskssl.h>

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

Parameters

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

enum_id
Specifies the enumeration identifier.

enum_value
Specifies the enumeration value.

Results

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

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

[GSK_INVALID_HANDLE]
The handle is not valid.

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

Usage

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

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

These enumeration identifiers are supported:

GSK_CERT_VALIDATION_MODE
Specifies the method of certificate validation. RFC 2459 and RFC 3280 describe differing methods of certificate validation. Specify GSK_CERT_VALIDATION_MODE_2459 if certificate validation according

to the RFC 2459 method is required or GSK_CERT_VALIDATION_MODE_3280 if certificate validation according to the RFC 3280 method is required.

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

GSK_CERT_VALIDATION_MODE can only be specified for an SSL environment.

GSK_CRL_SECURITY_LEVEL

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

Three levels of security are available:

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

GSK_CLIENT_AUTH_ALERT

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

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

GSK_CLIENT_AUTH_TYPE

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

When a client's certificate is being requested, the client can be required to provide a certificate by setting GSK_CLIENT_AUTH_ALERT to GSK_CLIENT_NOCERT_ALERT_ON. If no certificate is received, the requested handshake fails.

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

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

GSK_EXTENDED_RENEGOTIATION_INDICATOR

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

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

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

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

GSK_EXTENDED_RENEGOTIATION_INDICATOR can only be specified for an SSL environment.

GSK_PROTOCOL_SSLV2

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

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

When operating in FIPS mode, the SSL Version 2 protocol is not used. Enabling this protocol has no effect.

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

GSK_PROTOCOL_SSLV3

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

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

When operating in FIPS mode, the SSL Version 3 protocol is not used. Enabling this protocol has no effect.

GSK_PROTOCOL_TLSV1

Specifies GSK_PROTOCOL_TLSV1_ON to enable the TLS Version 1.0 protocol or GSK_PROTOCOL_TLSV1_OFF to disable the TLS Version 1.0 protocol.

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

GSK_PROTOCOL_TLSV1_1
Specifies GSK_PROTOCOL_TLSV1_1_ON to enable the TLS Version 1.1 protocol or GSK_PROTOCOL_TLSV1_1_OFF to disable the TLS Version 1.1 protocol.

GSK_PROTOCOL_TLSV1_1 can be specified for an SSL environment or an SSL connection.

GSK_PROTOCOL_TLSV1_2
Specify GSK_PROTOCOL_TLSV1_2_ON to enable the TLS Version 1.2 protocol or GSK_PROTOCOL_TLSV1_2_OFF to disable the TLS Version 1.2 protocol.

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

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

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

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

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

GSK_RENEGOTIATION can only be specified for an SSL environment.

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

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

GSK_RENEGOTIATION_PEER_CERT_CHECK can only be specified for an SSL environment.

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

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

GSK_SYSPLEX_SIDCACHE
Returns GSK_SYSPLEX_SIDCACHE_ON if sysplex session caching is

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

GSK_T61_AS_LATIN1
Specify GSK_T61_AS_LATIN1_ON to use the ISO8859-1 character set when processing a TELETEX string. Specify GSK_T61_AS_LATIN1_OFF to use the T.61 character set. The default is to use the ISO8859-1 character set.

Note: Selecting the incorrect character set can cause strings to be converted incorrectly. GSK_T61_AS_LATIN1 can be specified only for an SSL environment. This setting is global and affects all string conversions for all SSL environments.

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

gsk_attribute_set_tls_extension()

Defines a TLS extension to the SSL environment or connection.

Format

```
#include <gskssl.h>

gsk_attribute_set_tls_extension (
                                  gsk_handle           ssl_handle,
                                  gsk_tls_extension *  tls_extension)
```

Parameters

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

tls_extension
: Specifies the TLS extension structure containing extension data.

Results

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

[GSK_ATTRIBUTE_INVALID_TLS_EXTENSION]
: The TLS extension type identifier is not valid or cannot be used with the specified handle.

[GSK_ATTRIBUTE_INVALID_TLS_EXT_DATA]
: TLS extension data has been incorrectly defined.

[GSK_INVALID_HANDLE]
: The handle is not valid.

[GSK_INVALID_STATE]
: The handle is closed.

Usage

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

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

Note:

1. Setting an extension as required for a server means that all clients connecting to the server must have the extension enabled. Failure for a client to do so results in the server rejecting the connection request from the client. It is recommended that for maximum interoperability, that the required field is not enabled on the server side.
2. The gsk_tls_extension structure contains a 32-byte field, *rsvd*, which is reserved for future use. This field must contain binary zeros; any non-zero data results in **gsk_attribute_set_tls_extension()** returning a GSK_ATTRIBUTE_INVALID_TLS_EXT_DATA error.
3. Definition of TLS extensions for the client when any of the TLS protocols are enabled prevents the SSL V2 protocol from being used.

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

These TLS extension type identifiers are supported:

GSK_TLS_EXTID_SNI_SERVER_LABELS
Specifies the pairings of server name to certificate key label to be used when the TLS server receives a 'Server Name Indication' type TLS extension from the TLS client. The server name/key label pairs are used with the server name details received from the client to determine which certificate from the key database, PKCS #12 file, key ring or token is sent to the client as the servers certificate.

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

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

GSK_TLS_EXTID_SNI_SERVER_LABELS can be defined on both the server and client sides. Its settings, however, are effective when running as a server; it is ignored for clients.

Note:

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

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

with which server the client wants to communicate. The list of server names is defined using a pointer to an array of pointers to strings containing the server names.

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

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

GSK_TLS_EXTID_SNI_CLIENT_SNAMES can be defined on both the server and client sides. Its settings, however, are effective when running as a client; it is ignored for servers.

Note:

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

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

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

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

gsk_environment_init()

Initializes an SSL environment.

Format

```
#include <gskssl.h>

gsk_status gsk_environment_init (
                                    gsk_handle     env_handle)
```

Parameters

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

Results

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

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

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

[GSK_ERR_LDAP]
Unable to initialize the LDAP client.

[GSK_ERR_LDAP_NOT_AVAILABLE]
The LDAP server is not available.

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

[GSK_INSUFFICIENT_STORAGE]
Insufficient storage is available.

[GSK_INVALID_HANDLE]
The environment handle is not valid.

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

[GSK_KEYFILE_INVALID_FORMAT]
The database is not a key database.

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

[GSK_KEYFILE_PASSWORD_EXPIRED]
The key database password is expired.

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

[GSK_NO_KEYFILE_PASSWORD]
The key database password is not available.

Usage

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

gsk_environment_open()

Creates an SSL environment.

Format

```
#include <gskssl.h>

gsk_status gsk_environment_open (
                                    gsk_handle *     env_handle)
```

Parameters

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

Results

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

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

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

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

[GSK_INSUFFICIENT_STORAGE]
: Insufficient storage is available.

Usage

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

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

- SSL V2, SSL V3, and TLS V1.0 are enabled (TLS V1.1 and TLS V1.2 are disabled by default)
- The connection type is set to CLIENT
- The SSL V2 connection timeout is set to 100 seconds
- The SSL V3 connection timeout is set to 86400 seconds
- The SSL V2 cache size is set to 256
- The SSL V3 cache size is set to 512
- The sysplex session cache is disabled
- The default key will be used
- No revoked certificate checking performed
- The default callback routines will be used

- The SSL V2 cipher specification is set to "713642" if United States only encryption is enabled and "642" otherwise
- 2-character cipher definitions in GSK_V3_CIPHER_SPECS will be used for SSL V3 cipher values
- The SSL V3 cipher specification is set to "050435363738392F303132330A1613100D0915120F0C0306020100" if United States only encryption is enabled and "0915120F0C0306020100" otherwise
- The supported elliptic curve list is set to "00210023002400250019"
- The signature algorithm pair list is set to "060106030501050304010403030103030201020302020101"
- No TLS extensions are initialized

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

- TLS V1.0 is enabled (TLS V1.1 and TLS V1.2 are disabled by default)
- The connection type is set to CLIENT
- The connection timeout is set to 86400 seconds
- The cache size is set to 512
- The sysplex session cache is disabled
- The default key will be used
- No revoked certificate checking performed
- The default callback routines will be used
- 2-character cipher definitions in GSK_V3_CIPHER_SPECS will be used for SSL V3 cipher values
- The cipher specification is set to "35363738392F303132330A1613100D"
- The supported elliptic curve list is set to "00210023002400250019"
- The signature algorithm pair list is set to "060106030501050304010403030103030201020302020202"

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

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

- executing in non-FIPS mode with United States only encryption enabled:

  ```
  "000500040035003600370038003900 2F003000310032003300 0A0016001300 10000D000900150012
  000F000C000300060002000 10000"
  ```

- executing in non-FIPS mode with United States only encryption disabled:

  ```
  "00090015001200 0F000C000300060002000 10000"
  ```

- executing in FIPS mode:

  ```
  "00350036003700380039002F003000310032003300 0A0016001300 10000D"
  ```

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

- When using 2-character cipher suites:

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

- When using 4-character cipher suites:

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

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

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

These environment variables are processed:

GSK_CLIENT_ECURVE_LIST
Specifies the list of elliptic curves supported by the client as a string consisting of 1 or more 4-character decimal values in order of preference for use. The list is used by the client to guide the server as to which elliptic curves are preferred when using ECC-based cipher suites for the TLS V1.0, TLS V1.1, and TLS V1.2 protocols.

Only NIST recommended curves are able to be specified. To use Brainpool standard curves for an SSL environment or connection, set GSK_CLIENT_ECURVE_LIST to "" or use **gsk_attribute_set_buffer()** to reinitialize the GSK_CLIENT_ECURVE_LIST buffer to NULL.

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

GSK_EXTENDED_RENEGOTIATION_INDICATOR
Specifies the level of enforcement of renegotiation indication as specified by RFC 5746 during the initial handshake.

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

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

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

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

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

GSK_KEYRING_FILE
Specifies the name of the key database file, PKCS #12 file, SAF key ring or z/OS PKCS #11 token. A key database or PKCS #12 file is used if a database password is defined using either an environment variable or the

gsk_attribute_set_buffer() routine. When a stash file is defined, a key database file is used. Otherwise a SAF key ring or z/OS PKCS #11 token is used. GSK_KEYRING_FILE may be specified only for an SSL environment. See Chapter 6, “Updates to Chapter 10: Certificate/Key management,” on page 63 for a description of the restrictions when using a PKCS #12 file as the key database file.

The SAF key ring name is specified as "userid/keyring". The current user ID is used if the user ID is omitted. The user must have READ access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by the user. The user must have UPDATE access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by another user.

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

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

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

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

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

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

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

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

GSK_PROTOCOL_SSLV2
Specifies whether the SSL V2 protocol is supported. A value of "0", "OFF", or "DISABLED" disables the SSL V2 protocol while a value of "1", "ON", or

"ENABLED" enables the SSL V2 protocol. The SSL V2 protocol should be disabled whenever possible since the SSL V3 protocol provides significant security enhancements.

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

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

GSK_PROTOCOL_SSLV3
Specifies whether the SSL V3 protocol is supported. A value of "0", "OFF", or "DISABLED" disables the SSL V3 protocol while a value of "1", "ON", or "ENABLED" enables the SSL V3 protocol.

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

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

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

GSK_PROTOCOL_TLSV1_2
Specifies whether the TLS V1.2 protocol is supported. A value of "0", "OFF", or "DISABLED" disables the TLS V1.2 protocol while value of "1", "ON", or "ENABLED" enables the TLS V1.2 protocol. The TLS V1.2 protocol uses the same session cache as the SSL V3 protocol. The TLS V1.2 protocol will not use export cipher suites. 40-bit ciphers will be ignored if TLS V1.2 is negotiated as the communications protocol.

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

Specify "NONE" to disable SSL V3 and TLS handshake renegotiation as a server and allow RFC 5746 renegotiation. This is the default.

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

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

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

GSK_RENEGOTIATION_PEER_CERT_CHECK
Specifies if the peer certificate is allowed to change during renegotiation.

Specify "OFF" or "0" to not perform an identity check against the peer's certificate during renegotiation. This allows the peer certificate to change during renegotiation. This is the default.

Specify "ON" or "1" to perform a comparison against the peer's certificate to ensure certificate does not change during renegotiation.

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

GSK_TLS_SIG_ALG_PAIRS
Specifies the list of hash and signature algorithm pair specifications supported by the client or server as a string consisting of 1 or more 4-character values in order of preference for use. The signature algorithm pair specifications are sent by either the client or server to the session partner to indicate which signature/hash algorithm combinations are supported for digital signatures. Signature algorithm pair specification only has relevance for sessions using TLS V1.2 or higher protocols.

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

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

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

GSK_V3_CIPHER_SPECS
Specifies the SSL V3 cipher specifications in order of preference as a null-terminated string consisting of 1 or more 2-character values. The SSL V3 cipher specifications are used for the SSL V3, TLS V1.0, and higher protocols. Valid cipher specifications that are not supported because of the installed cryptographic level will be skipped when the connection is initialized. For protocols TLS V1.1 and above, 40-bit ciphers will be ignored if these protocols are negotiated as the security protocol. For protocols TLS V1.2 and above, the 56-bit DES cipher suites will be ignored if these protocols are negotiated as the communications protocol.

GSK_V3_CIPHER_SPECS_EXPANDED
Specifies the SSL V3 cipher specifications in order of preference as a null-terminated string consisting of 1 or more 4-character values. The SSL V3 cipher specifications are used for the SSL V3.0, TLS V1.0, and higher protocols. Valid cipher specifications that are not supported because of the installed cryptographic level will be skipped when the connection is initialized. For protocols TLS V1.1 and above, 40-bit ciphers will be ignored if these protocols are negotiated as the security protocol. For protocols TLS V1.2 and above, the 56-bit DES cipher suites will be ignored if these protocols are negotiated as the communications protocol.

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

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

gsk_get_update()

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

Format

```
#include <gskssl.h>

gsk_status gsk_get_update (
                            gsk_handle     env_handle,
                            long *         update_flag)
```

Parameters

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

update_flag
Returns 1 if the key database, PKCS #12 file, SAF key ring or z/OS PKCS #11 token has been updated or 0 if it has not been updated.

Results

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

[GSK_INVALID_HANDLE]
The environment handle is not valid.

[GSK_INVALID_STATE]
The environment is not in the initialized state.

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

Usage

The **gsk_get_update()** routine tests if the key database, PKCS #12 file, SAF key ring or z/OS PKCS #11 token associated with the SSL environment has been updated since the last time that **gsk_get_update()** was called or since the environment was initialized if **gsk_get_update()** has not been called yet. If an update has occurred, the application can close the current environment and then create a new environment to pick up the updates.

gsk_secure_socket_init()

Initializes a secure socket connection.

Format

```
#include <gskssl.h>

gsk_status gsk_secure_socket_init(
                                    gsk_handle     soc_handle)
```

Parameters

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

Results

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

[GSK_CERTIFICATE_NOT_AVAILABLE]
No certificates available.

[GSK_ERR_BAD_CERT]
Certificate is not valid.

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

[GSK_ERR_BAD_EC_PARAMS]
EC parameters not supplied.

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

[GSK_ERR_BAD_MAC]
Message verification failed.

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

[GSK_ERR_BAD_MSG_LEN]
Incorrectly-formatted TLS extension data contained within message received from peer application.

[GSK_ERR_BAD_PEER]
Peer application has violated the SSL protocol.

[GSK_ERR_BAD_SIG_ALG_PAIR]
Signature algorithm pairs list is not valid.

[GSK_ERR_BAD_V2_CIPHER]
SSL V2 cipher is not valid.

[GSK_ERR_BAD_V3_CIPHER]
SSL V3 cipher is not valid.

[GSK_ERR_BAD_V3_EXPANDED_CIPHER]
SSL V3 expanded cipher is not valid.

[GSK_ERR_CERT_VALIDATION]
Certificate validation error.

[GSK_ERR_CERTIFICATE_REVOKED]
Peer certificate is revoked.

[GSK_ERR_CRYPTO]
Cryptographic error detected.

[GSK_ERR_EC_PARAMETERS_NOT_SUPPLIED]
EC parameters not supplied.

[GSK_ERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[GSK_ERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

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

[GSK_ERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[GSK_ERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[GSK_ERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[GSK_ERR_INVALID_FRAGMENT_LENGTH]
An unsupported fragment length was received.

[GSK_ERR_IO]
I/O error communicating with peer application.

[GSK_ERR_LDAP]
An LDAP error is detected.

[GSK_ERR_LDAP_NOT_AVAILABLE]
The LDAP server is not available.

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

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

[GSK_ERR_MULTIPLE_DEFAULT]
Multiple keys are marked as the default.

[GSK_ERR_MULTIPLE_LABEL]
Multiple certificates exist for label.

[GSK_ERR_NO_CERTIFICATE]
No certificate received from partner.

[GSK_ERR_NO_CIPHERS]
No cipher specifications.

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

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

[GSK_ERR_SIGNATURE_NOT_SUPPLIED]
Signature not supplied.

[GSK_ERR_SOCKET_CLOSED]
Socket connection closed by peer application.

[GSK_ERR_RNG]
Error encountered when generating random bytes.

[GSK_ERR_UNKNOWN_CA]
A certification authority certificate is missing.

[GSK_ERR_UNRECOGNIZED_NAME]
The requested server name is not recognized.

[GSK_ERR_UNSUPPORTED]
SSL protocol or certificate type is not supported.

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

[GSK_ERR_UNSUPPORTED_REQUIRED_EXTENSION]
A required TLS extension has been rejected.

[GSK_ERR_UNSUPPORTED_EXTENSION]
An unrequested TLS Extension has been encountered.

[GSK_INSUFFICIENT_STORAGE]
Insufficient storage is available.

[GSK_INVALID_HANDLE]
The connection handle is not valid.

[GSK_INVALID_STATE]
The connection is not in the open state or a previous initialization request has failed.

[GSK_RSA_TEMP_KEY_PAIR]
Unable to generate temporary RSA public/private key pair.

[GSK_WOULD_BLOCK_READ]
An attempt to read a handshake message failed with EWOULDBLOCK.

[GSK_WOULD_BLOCK_WRITE]
An attempt to write a handshake message failed with EWOULDBLOCK.

Usage

The **gsk_secure_socket_init()** routine initializes a secure socket connection created by the **gsk_secure_socket_open()** routine. After the connection has been initialized, it can be used for secure data transmission using the **gsk_secure_socket_read()** and **gsk_secure_socket_write()** routines. The **gsk_secure_socket_close()** routine should be called to close the connection when it is no longer needed. The **gsk_secure_socket_close()** routine should also be called if an error is returned by the **gsk_secure_socket_init()** routine.

Before calling the **gsk_secure_socket_init()** routine, the application must create a connected socket and store the socket descriptor in the SSL connection by calling the **gsk_attribute_set_numeric_value()** routine. For a client, this means calling the **socket()** and **connect()** routines. For a server, this means calling the **socket()**, **listen()**, and **accept()** routines. However, SSL does not require the use of TCP/IP for the communications layer. The socket descriptor can be any integer value

which is meaningful to the application. The application must provide its own socket routines if it is not using TCP/IP by calling the **gsk_attribute_set_callback()** routine.

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

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

Protocol Selection

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

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

Note:

1. SSL V2 is not as secure as SSL V3 or TLS and should be disabled whenever possible to avoid attacks that force the client and server to drop back to SSL V2 even though they are capable of using SSL V3, TLS V1.0 or TLS V1.1.
2. When TLS extensions are defined for a client and any of the TLS protocols are enabled for the connection, SSL V2 is not negotiated even if it is enabled.
3. If TLS V1.2 is enabled on the client, establishment of SSL sessions with SSL V2 servers is not supported.

Cipher selection

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

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

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

Note:

1. For protocols TLS V1.1 and above, export cipher suites cannot be used. 40-bit ciphers is ignored if TLS V1.1 or above is negotiated as the security protocol. If TLS V1.1 or above is the intended protocol and only 40-bit ciphers are available, the connection fails with GSK_ERR_NO_CIPHERS.
2. To use a cipher specification that requires a fixed ECDH key exchange (C001, C002, C003, C004, C005, C00B, C00C, C00D, C00E, and C00F), the ECC private key cannot be a secure key that is stored in ICSF PKDS.

Server certificate

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

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

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

If the servers certificate contains a key usage extension during the SSL handshake, it must allow key usage as follows:

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

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

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

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

Client certificate

The SSL server always provides its certificate to the SSL client as part of the handshake. The client always performs server authentication using the certificate that is provided by the server. Depending upon the server handshake type, the server may ask the client to provide its certificate. The key label that is stored in the connection is used to retrieve the certificate from the key database, PKCS #12 file, key ring, or token. The default key is used if no label is set. The key record must contain both an X.509 certificate and a private key.

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

- RSA key exchange - RSA or DSA
- fixed Diffie-Hellman key exchange - RSA, DSA, or Diffie-Hellman
- ephemeral Diffie-Hellman key exchange - RSA or DSA
- fixed ECDH key exchange - RSA, DSA, or ECDSA
- ephemeral ECDH key exchange - RSA, DSA, or ECDSA

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

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

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

gsk_initialize()

Initializes the System SSL runtime environment.

Format

```
#include <gskssl.h>

gsk_status gsk_initialize(
                            gsk_init_data *     init_data)
```

Parameters

init_data
Specifies the data used to initialize the SSL runtime environment.

Results

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

[GSK_ERR_INIT_PARM_NOT_VALID]
An initialization parameter is not valid.

[GSK_ERROR_BAD_MALLOC]
Insufficient storage is available.

[GSK_ERROR_CRYPTO]
Cryptographic error detected.

[GSK_ERROR_LDAP]
Unable to initialize the LDAP client.

[GSK_ERROR_MULTIPLE_LABEL]
Multiple certificates exist for label.

[GSK_ERROR_MULTIPLE_DEFAULT]
Multiple keys are marked as the default.

[GSK_ERROR_PERMISSION_DENIED]
Not authorized to access the key database, PKCS #12 file, key ring or token.

[GSK_INIT_SEC_TYPE_NOT_VALID]
The security type is not valid.

[GSK_INIT_V2_TIMEOUT_NOT_VALID]
The SSL V2 timeout is not valid.

[GSK_INIT_V3_TIMEOUT_NOT_VALID]
The SSL V3 timeout is not valid.

[GSK_KEYFILE_BAD_FORMAT]
Key database, PKCS #12 file, or key ring format is not valid.

[GSK_KEYFILE_BAD_PASSWORD]
Key database or PKCS #12 file password is not correct.

[GSK_KEYFILE_IO_ERROR]
Unable to read the key database, PKCS #12 file, key ring or token.

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

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

[GSK_KEYFILE_PW_EXPIRED]
Key database password is expired.

Usage

The **gsk_initialize()** routine initializes the System SSL runtime environment for the current process. The **gsk_uninitialize()** routine should be called to release the SSL environment when it is no longer needed. Multiple calls to **gsk_initialize()** causes the existing environment to be released before creating the new environment.

Environment variables are processed along with the **gsk_initialize** data structures. Information passed in the key database, key ring or token is read as part of the environment initialization. Upon successful completion of **gsk_initialize()**, the application is ready to begin creating and using secure socket connections.

The gsk_init_data structure contains these fields:

sec_types
Specifies one of these null-terminated character strings:

- "SSLV2" or "SSL20" to use the SSL V2 protocol
- "SSLV3" or "SSL30" to use the SSL V3 protocol
- "TLSV1" or "TLS10" to use the TLS V1.0 protocol
- "SSLV2_OFF" to allow either TLS V1.0 or SSL V3 to be used
- "ALL" to use any supported protocol (SSL V2, SSL V3, and TLS V1.0).

When "SSLV2_OFF" is specified the SSL client/server attempts first to use the TLS V1.0 protocol, before falling back to the most secure protocol supported by its SSL partner, excluding the SSL V2 protocol.

When "ALL" is specified for an SSL client, the client attempts first to use the TLS V1.0 protocol and falls back to the most secure protocol that the server supports, excluding the SSL V2 protocol (the client must explicitly request the SSL V2 protocol if it wants to use this protocol).

When "ALL" is specified for an SSL server, the server accepts any of the supported protocols.

When running in FIPS mode, the minimum requirement is TLS V1.0 protocol. If only the SSL V2 or the SSL V3 protocol is enabled, then a FIPS mode SSL connection is not possible.

keyring
Specifies the name of the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token as a null-terminated character string. When both the password and stash file name are NULL, a SAF key ring or PKCS #11 token is used.

The SAF key ring name is specified as "userid/keyring". The current user ID is used if the user ID is omitted. The user must have READ access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by the user. The user must have UPDATE access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by another user.

Note: Certificate private keys are not available when using a SAF key ring owned by another user, except for SITE certificates where CONTROL

authority is given to IRR.DIGTCERT.GENCERT in the FACILITY class or for user certificates where READ or UPDATE authority is given to *ringOwner.ringName*.LST resource in the RDATALIB class.

The z/OS PKCS #11 token name is specified as *TOKEN*/token-name. *TOKEN* indicates that the specified key ring is actually a token name. The application user ID must have READ access to resource USER.token-name in the CRYPTOZ class in order for the certificate and their private keys, if present, to be read.

keyring_pw
Specifies the password for the key database or PKCS #12 file as a null-terminated character string. Specify NULL to indicate that no password is provided.

keyring_stash
Specifies the name of the password stash file as a null-terminated character string. Specify NULL to indicate no stash file is provided. The password stash file is used if the keyring_pw value is NULL.

V2_session_timeout
Specifies the SSL V2 session cache timeout value in seconds. The valid range is 0 to 100. A short SSL handshake is performed when a cached session exists since the session parameters have already been negotiated between the client and the server.

V3_session_timeout
Specifies the SSL V3 session cache timeout value in seconds. The valid range is 0 to 86400. A short SSL handshake is performed when a cached session exists since the session parameters have already been negotiated between the client and the server.

LDAP_server
Specifies one or more blank-separated LDAP server host names as a null-terminated character string. Each host name can contain an optional port number separated from the host name by a colon. The LDAP server is used for certificate validation . The LDAP server is used only when LDAP_CA_roots is set to GSK_CA_ROOTS_LOCAL_AND_X500 and *auth_type* is not set to GSK_CLIENT_AUTH_LOCAL or GSK_CLIENT_AUTH_PASSTHRU.

LDAP_port
Specifies the LDAP server port. The default LDAP port will be used if 0 is specified.

LDAP_user
Specifies the distinguished name to use when connecting to the LDAP server and is a null-terminated character string. An anonymous bind is done if NULL is specified for this field.

LDAP_password
Specifies the password to use when connecting to the LDAP server and is a null-terminated character string. This field is ignored if NULL is specified for LDAP_user.

LDAP_CA_roots
Specifies the location of CA certificates and certificate revocation lists used to validate certificates. When GSK_CA_ROOTS_LOCAL_ONLY is specified, the CA certificates and certificate revocation lists are obtained from the local database. When GSK_CA_ROOTS_LOCAL_AND_X500 is specified, the CA certificates and certificate revocation lists are obtained from the

LDAP server if they are not found in the local database. Even when an LDAP server is used, root CA certificates must be found in the local database since the LDAP server is not a trusted data source.

auth_type
Specifies the client authentication type. This field is ignored unless LDAP_CA_roots is set to GSK_CA_ROOTS_LOCAL_AND_X500. The client certificate is not validated when GSK_CLIENT_AUTH_PASSTHRU is specified. The client certificate is validated using just the local database when GSK_CLIENT_AUTH_LOCAL is specified. CA certificates and certificate revocation lists not found in the local database are obtained from the LDAP server when GSK_CLIENT_AUTH_STRONG or GSK_CLIENT_AUTH_STRONG_OVER_SSL is specified (the local database must still contain the root CA certificates). There is no difference between GSK_CLIENT_AUTH_STRONG and GSK_CLIENT_AUTH_STRONG_OVER_SSL.

gsk_initialize() Supported environment variables:

Environment variables are processed along with the information passed in the gsk_init_data structure during environment initialization. Also, during environment initialization, the key database, PKCS #12 file, key ring, or token is read.

The **gsk_initialize()** routine supports these environment variables:

GSK_EXTENDED_RENEGOTIATION_INDICATOR
Specifies the level of enforcement of renegotiation indication as specified by RFC 5746 during the initial handshake.

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

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

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

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

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

Specify "NONE" to disable SSL V3 and TLS handshake renegotiation as a server and allow RFC 5746 renegotiation. This is the default.

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

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

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

GSK_RENEGOTIATION_PEER_CERT_CHECK
Specifies if the peer certificate is allowed to change during renegotiation.

Specify "OFF" or "0" to not perform an identity check against the peer's certificate during renegotiation. This allows the peer certificate to change during renegotiation. This is the default.

Specify "ON" or "1" to perform a comparison against the peer's certificate to ensure that certificate does not change during renegotiation.

GSKV2CACHESIZE
Specifies the number of entries in the SSL V2 session cache with a range of 0 to 32000. The value that is specified by the GSK_V2_SIDCACHE_SIZE environment variable is used if the GSKV2CACHESIZE variable is not defined. The default value is 256 if neither environment variable is defined.

GSKV3CACHESIZE
Specifies the number of entries in the SSL V3 session cache with a range of 0 to 64000. The value that is specified by the GSK_V3_SIDCACHE_SIZE environment variable is used if the GSKV3CACHESIZE variable is not defined. The default value is 512 if neither environment variable is defined. The SSL V3 session cache is used for both the SSL V3 and TLS V1.0 protocols.

The environment variables that are overridden with information passed in the gsk_init_data structure are:

- GSK_KEYRING_FILE
- GSK_KEYRING_PW
- GSK_KEYRING_STASH
- GSK_LDAP_SERVER
- GSK_LDAP_PASSWORD
- GSK_LDAP_PORT
- GSK_LDAP_USER
- GSK_PROTOCOL_SSLV2
- GSK_PROTOCOL_SSLV3
- GSK_PROTOCOL_TLSV1
- GSK_V2_SESSION_TIMEOUT
- GSK_V3_SESSION_TIMEOUT

gsk_secure_soc_init()

Initializes a secure socket connection.

Format

```
#include <gskssl.h>

gsk_soc_data * gsk_secure_soc_init(
                                    gsk_soc_init_data *     init_data)
```

Parameters

init_data
Specifies the socket connection initialization data.

Results

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

[GSK_ERR_INIT_PARM_NOT_VALID]
A connection initialization parameter is not valid.

[GSK_ERROR_BAD_CERT]
A certificate is not valid.

[GSK_ERROR_BAD_DATE]
A certificate is not valid yet or is expired.

[GSK_ERROR_BAD_MAC]
Message verification failed.

[GSK_ERROR_BAD_MALLOC]
Insufficient storage is available.

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

[GSK_ERROR_BAD_PEER]
Peer application has violated the SSL protocol.

[GSK_ERROR_BAD_STATE]
The SSL environment has not been initialized.

[GSK_ERROR_CRYPTO]
Cryptographic error detected.

[GSK_ERROR_INCOMPATIBLE_KEY]
The certificate key is not compatible with the negotiated cipher suite.

[GSK_ERROR_IO]
I/O error communicating with peer application.

[GSK_ERROR_LDAP]
An LDAP error is detected.

[GSK_ERROR_LDAP_NOT_AVAILABLE]
The LDAP server is not available.

[GSK_ERROR_NO_CIPHERS]
No cipher specifications.

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

[GSK_ERROR_RNG]
Error encountered when generating random bytes.

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

[GSK_ERROR_SOCKET_CLOSED]
Socket connection closed by peer application.

[GSK_ERROR_UNKNOWN_CA]
A certification authority certificate is missing.

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

[GSK_ERROR_VALIDATION]
Certificate validation error.

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

[GSK_KEYFILE_BAD_LABEL]
The DName field of the gsk_soc_init_data structure is an empty string. If the default key is to be used, the DName field must be NULL.

[GSK_KEYFILE_DUPLICATE_NAME]
The key database contains multiple certificates with the same subject name as the distinguished name specified in the connection initialization data.

[GSK_SOC_BAD_V2_CIPHER]
SSL V2 cipher is not valid.

[GSK_SOC_BAD_V3_CIPHER]
SSL/TLS V3 cipher is not valid.

[GSK_SOC_NO_READ_FUNCTION]
No read function is specified in the connection initialization data.

[GSK_SOC_NO_WRITE_FUNCTION]
No write function is specified in the connection initialization data.

Usage

The **gsk_secure_soc_init()** routine initializes a secure socket connection. The **gsk_initialize()** routine must be called before any secure socket connections can be initialized. After the connection has been initialized, it can be used for secure data transmission using the **gsk_secure_soc_read()** and **gsk_secure_soc_write()** routines. The **gsk_secure_soc_close()** routine should be called to close the connection when it is no longer needed. The **gsk_secure_soc_close()** routine should not be called if an error is returned by the **gsk_secure_soc_init()** routine.

Before calling the **gsk_secure_soc_init()** routine, the application must create a connected socket. For a client, this means calling the **socket()** and **connect()** routines. For a server, this means calling the **socket()**, **listen()**, and **accept()** routines. However, SSL does not require the use of TCP/IP for the communications layer. The socket descriptor can be any integer value that is meaningful to the application. The application must provide its own socket routines if it is not using TCP/IP.

An SSL handshake is performed as part of the processing of the **gsk_secure_soc_init()** routine. This establishes the server identity and optionally the client identity. It also negotiates the cryptographic parameters to be used for the connection.

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

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

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

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

The SSL server always provides its certificate to the SSL client as part of the handshake. Depending upon the server handshake type, the server may ask the client to provide its certificate. The key label that is stored in the connection is used to retrieve the certificate from the key database, PKCS #12 file, key ring, or token. The default key will be used if no label is set. The key record must contain both an X.509 certificate and a private key.

These SSL V2 cipher specifications are supported in non-FIPS mode only:

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

These SSL V3 cipher specifications are supported in non-FIPS mode only:

- "00" = No encryption or message authentication and RSA key exchange
- "01" = No encryption with MD5 message authentication and RSA key exchange

- "02" = No encryption with SHA-1 message authentication and RSA key exchange
- "03" = 40-bit RC4 encryption with MD5 message authentication and RSA key exchange
- "04" = 128-bit RC4 encryption with MD5 message authentication and RSA key exchange
- "05" = 128-bit RC4 encryption with SHA-1 message authentication and RSA key exchange
- "06" = 40-bit RC2 encryption with MD5 message authentication and RSA key exchange
- "09" = 56-bit DES encryption with SHA-1 message authentication and RSA key exchange
- "0C" = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate
- "0F" = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "12" = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate
- "15" = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate

These SSL V3 cipher specifications are supported in FIPS mode and non-FIPS mode:

- "0A" = 168-bit Triple DES encryption with SHA-1 message authentication and RSA key exchange
- "0D" = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate
- "10" = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "13" = 168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate
- "16" = 168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
- "2F" = 128-bit AES encryption with SHA-1 message authentication and RSA key exchange
- "30" = 128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate
- "31" = 128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "32" = 128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate
- "33" = 128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
- "35" = 256-bit AES encryption with SHA-1 message authentication and RSA key exchange
- "36" = 256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate
- "37" = 256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "38" = 256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate

- "39" = 256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate

The client sends a list of ciphers it supports during the SSL handshake. The server application uses this list, and the defined ciphers supported by the server, to determine the cipher to be used during the SSL handshake. This selection is done by looking through the servers cipher list for a match in the clients list. The first matching cipher is used.

Environment variables are processed along with the information passed in the gsk_init_data structure during environment initialization. Also during environment initialization, the key database, PKCS #12 file, key ring or token is read.

The environment variables that are overridden by non-NULL values in the gsk_soc_init_data structure are:

- GSK_KEY_LABEL
- GSK_V2_CIPHER_SPECS
- GSK_V3_CIPHER_SPECS

The gsk_soc_init_data structure contains these fields:

fd Specifies the socket descriptor for the secure connection. The socket must remain open until after the **gsk_secure_soc_close()** routine has been called to close the secure connection.

hs_type
Specifies the intended handshake type as follows:

GSK_AS_CLIENT
Performs a client SSL handshake

GSK_AS_CLIENT_NO_AUTH
Performs a client SSL handshake but do not provide a client certificate to the SSL server

GSK_AS_SERVER
Performs a server SSL handshake

GSK_AS_SERVER_WITH_CLIENT_AUTH
Performs a server SSL handshake with client authentication

DName
Specifies either the distinguished name or the key label of the local certificate. Specify NULL to use the default key for the key database, key ring or token.

sec_type
Returns the selected security protocol as "SSLV2", "SSLV3", or "TLSV1". This is a static string and must not be modified or freed by the application.

cipher_specs
Specifies the SSL V2 cipher specifications as a null-terminated string consisting of 1 or more 1-character values. Specify NULL to use the default cipher specifications ("713642" if Security Level 3 FMID encryption is enabled and "642" otherwise). Valid cipher specifications that are not supported because of the installed cryptographic level will be skipped when the connection is initialized. The SSL V2 protocol can only be used when executing in non-FIPS mode.

v3cipher_specs
Specifies the SSL V3 cipher specifications as a null-terminated string consisting of 1 or more 2-character values. Specify NULL to use the default cipher specifications ("050435363738392F303132330A1613100D0915120F0C0306020100" if Security Level 3 FMID is installed and in non-FIPS mode, "35363738392F303132330A1613100D" if Security Level 3 FMID is installed and in FIPS mode, and "0915120F0C0306020100" otherwise). The SSL V3 cipher specifications are used for both the SSL V3 and TLS V1.0 protocols. Valid cipher specifications that are not supported because of the installed cryptographic level are skipped when the connection is initialized. The SSL V3 protocol can only be used when executing in non-FIPS mode.

skread Specifies the address of the read routine used during the SSL handshake. See “gsk_attribute_set_callback()” on page 21 for additional information about the I/O callback routines.

skwrite Specifies the address of the write routine used during the SSL handshake. See “gsk_attribute_set_callback()” on page 21 for additional information about the I/O callback routines.

cipherSelected
Returns the selected cipher for the SSL V2 protocol as a 3-byte binary value:

- 0x010080 - 128-bit RC4 encryption with MD5 message authentication
- 0x020080 = 128-bit RC4 export encryption with MD5 message authentication
- 0x030080 = 128-bit RC2 encryption with MD5 message authentication
- 0x040080 = 128-bit RC2 export encryption with MD5 message authentication
- 0x060040 = 56-bit DES encryption with MD5 message authentication
- 0x0700c0 = 168-bit Triple DES encryption with MD5 message authentication

v3cipherSelected
Returns the selected cipher for the SSL V3 or TLS V1.0 protocol as a 2-byte character value with no string delimiter:

- "00" = No encryption or message authentication
- "01" = No encryption with MD5 message authentication and RSA key exchange
- "02" = No encryption with SHA-1 message authentication and RSA key exchange
- "03" = 40-bit RC4 encryption with MD5 message authentication and RSA key exchange
- "04" = 128-bit RC4 encryption with MD5 message authentication and RSA key exchange
- "05" = 128-bit RC4 encryption with SHA-1 message authentication and RSA key exchange
- "06" = 40-bit RC2 encryption with MD5 message authentication and RSA key exchange
- "09" = 56-bit DES encryption with SHA-1 message authentication and RSA key exchange
- "0A" = 168-bit Triple DES encryption with SHA-1 message authentication and RSA key exchange

- "0C" = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate
- "0D" = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate
- "0F" = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "10" = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "12" = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate
- "13" = 168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate
- "15" = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
- "16" = 168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
- "2F" = 128-bit AES encryption with SHA-1 message authentication and RSA key exchange
- "30" = 128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate
- "31" = 128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "32" = 128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate
- "33" = 128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
- "35" = 256-bit AES encryption with SHA-1 message authentication and RSA key exchange
- "36" = 256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate
- "37" = 256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "38" = 256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate
- "39" = 256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate

failureReasonCode
Returns the **gsk_secure_soc_init()** error code.

cert_info
Returns peer certificate information. The application must not modify or free this information.

gsk_data
This field is ignored. The key database information is set when **gsk_initialize()** is called.

Chapter 6. Updates to Chapter 10: Certificate/Key management

This topic discusses the use of the z/OS shell-based gskkyman utility to manage private keys, certificates, and tokens.

Introduction

SSL connections use public/private key mechanisms for authenticating each side of the SSL session and agreeing on bulk encryption keys to be used for the SSL session. To use public/private key mechanisms (termed PKI), public/private key pairs must be generated. In addition, X.509 certificates (which contain public keys) might need to be created, or certificates must be requested, received, and managed.

System SSL supports three methods for managing PKI private keys and certificates:

- A z/OS shell-based program called **gskkyman**. **gskkyman** creates, completes, and manages either a z/OS file or z/OS PKCS #11 token that contains PKI private keys, certificate requests, and certificates. The z/OS file is called a **key database** and, by convention, has a file extension of **.kdb**.
- The z/OS Security Server (RACF®) RACDCERT command. RACDCERT installs and maintains PKI private keys and certificates in RACF. See *z/OS Security Server RACF Command Language Reference* for details about the RACDCERT command. RACF supports multiple PKI private keys and certificates to be managed as a group. These groups are called **key rings** or z/OS PKCS #11 tokens.
- PKCS #12 standard files created according to PKCS #12 V3.0. These files must be created as binary format files whose fully qualified file name does not exceed 251 characters in length and does not end with .kdb, .rdb, or .sth.

  System SSL supports PKCS #12 certificate and private key objects types. Any other object types within the file are ignored. All certificates within the files are treated as trusted certificates and no certificate can be identified as a default certificate.

  The PKCS #12 file is protected by a password and the integrity of the file is ensured by a SHA-1 message authentication value.

  Because PKCS #12 files do not have labels as certificates in key database files (SAF key rings or PKCS #11 tokens can have labels as certificates in key database files), when the certificates are read into storage, they are assigned a label using either the PKCS #12 friendly name, if one exists, or the certificate's subject distinguished name. When the friendly name or the subject distinguished name value is greater than 127 characters, only the first 127 characters are used. If multiple certificates have the same friendly name value, the first encountered certificate is read into storage. Any other certificate with that friendly name is ignored. If a certificate is encountered that does not contain a friendly name and the subject distinguished name is empty, the processing of the PKCS #12 fails. As with key database files, SAF key rings and PKCS #11 tokens, the label is case sensitive.
- RACF key rings or z/OS PKCS #11 tokens are the preferred method for managing PKI private keys and certificates for System SSL.

The System SSL application uses the GSK_KEYRING_FILE parameter of the **gsk_attribute_set_buffer()** API or the GSK_KEYRING_FILE environment variable to specify the locations of the PKI private keys and certificates to System SSL. If

you are using a z/OS key database or a PKCS #12 file, the name is passed in this parameter. If you are using a RACF key ring or z/OS PKCS #11 token, the name of the key ring or token is passed in this parameter.

gskkyman command line mode syntax

This topic describes the format and options of the **gskkyman** command.

gskkyman

The **gskkyman** utility is used for key database management, z/OS PKCS #11 token management, and to display the certificates within a PKCS #12 file.

Format

gskkyman

```
gskkyman -dc -k filename -l label
gskkyman -dc -t token-name -l label
gskkyman -dc -p12 filename -l label

gskkyman -dcv -k filename -l label
gskkyman -dcv -t token-name -l label
gskkyman -dcv -p12 filename -l label

gskkyman -dk -k filename

gskkyman -e -k filename -l label -p filename
gskkyman -e -t token-name -l label -p filename

gskkyman -g -x days -cr filename -ct filename -k filename -l label -kt keytype -ca -ic
gskkyman -g -x days -cr filename -ct filename -t token-name -l label -kt keytype -ca -ic

gskkyman -h

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

gskkyman -s -k filename
```

Parameters

function
The function to be performed. It must follow the command name. The acceptable values are:

-dc
Display certificate details

-dcv
Display certificate verbose details

-dk
Display key database expiration and record length

-e Export a certificate and its associated private key

-g Sign a certificate for a certificate request

-h Display the command syntax

-i Import a certificate and its associated private key

-s Store the database password in the stash file

option
The parameters necessary to accomplish the function. If the option provides a value, then the value must follow the option:

The acceptable values are:

-ca
A certification authority certificate is generated if **-ca** is specified. An end user certificate is generated if **-ca** is not specified.

-cr
Specifies the name of the certificate request file. You are prompted for the file name if this option is not specified.

-ct
Specifies the name of the output generated signed certificate file. You are prompted for the file name if this option is not specified. You may specify any name. If you specify an existing file name, the file is overwritten.

-ic
The certification chain certificates are included in the certificate file if **-ic** is specified. Otherwise, just the signed certificate is included in the certificate file.

-k Specifies the name of the key database. This option is mutually exclusive with the **-t** option and the **-p12** option. You are prompted for the key database file name if either this option or the **-t** option or the **-p12** option is specified. The length of the fully qualified file name cannot exceed 251 characters. If the file name does not end with an extension of 1-3 characters, the length of the fully qualified file name cannot exceed 247 characters. Finally, the key database name cannot end with .rdb or .sth.

-kt
Specifies the key type of the certificate to be created. This option is valid when signing an end user certificate or certificate request containing an ECC public key and affects the settings of the keyUsage extension of the certificate created. Valid key type options are ecgen, ecdsa and ecdh. ecgen creates a certificate with digitalSignature, nonRepudiation and keyAgreement set, ecdsa creates a certificate with digitalSignature and nonRepudiation set, and ecdh creates a certificate with keyAgreement set. If the **-kt** option is not specified for an end user ECC certificate or certificate request, the default option is ecgen. For other certificate types the **-kt** option is ignored.

-l Specifies the certificate label. The label must be enclosed in double quotation marks if it contains one or more spaces. If the certificate is being used to sign a certificate request (sign function), the certificate must be a CA. The label for the default key is used if this option is not specified (export or sign function) or you are prompted for the label (import function). If more than one certificate with the specified label exists (can occur for tokens), the user is prompted to either cancel or choose the required certificate from a list that summarizes significant fields in the certificate.

-p Specifies the name of the PKCS #12 file. You are prompted for the file name if this option is not specified.

-p12
Specifies the name of the PKCS #12 file containing the certificates to be displayed. This option is mutually exclusive with the **-k** option and the **-t** option. The length of the fully qualified file name cannot exceed 251

characters. If the file name does not end with an extension of 1-3 characters, the length of the fully qualified file name cannot exceed 247 characters. Lastly, the PKCS #12 file cannot end with .kdb, .rdb or .sth.

-t Specifies the name of the token to be managed. This option is mutually exclusive with the **-k** option and the **-p12** option. The name must consist of characters that are alphanumeric, national (@ x5B, # x7B, $ x7C) or period (.x4B). The first character must be alphabetic or national. Lowercase letters are allowed, but are folded to uppercase.

-x Specifies the number of days until the signed certificate expires and must be between 1 and 9999 days. The certificate expires in 365 days if this option is not specified.

Results

If **gskkyman** is specified with no arguments the interactive menu-driven interface is used.

Usage

The **gskkyman** utility is used to manage a token, a key database and its associated request database, or to list the contents of a PKCS #12 file. Interactive menus are displayed if no command options are specified. Otherwise, the requested token/database/PKCS #12 file function is performed and the **gskkyman** utility exits.

Note: The ability to display the contents of a PKCS #12 file is not supported through the interactive menu-driven interface.

If the command specifies the **-t** (token name) option, then the requested function is performed for the identified token.

If the command specifies the **-p12** (PKCS #12 file) option on the display functions **-dc** or **-dcv**, if **-l** option is used, the certificate with matching label is displayed. If **-l** option is not used, all certificates within the file are displayed.

If the command does not specify the **-t** or the **-p12** option, then it is assumed that the function is to be performed for a key database. If the **-k** option, the **-t** option, and the **-p12** option are not supplied, the user is prompted for a key database file name.

If any combination of **-k**, **-t**, and **-p12** is specified, the command is rejected and an error message is displayed.

For commands applied to a key database:

The key database contains certificates and private keys and normally has a file name extension of '.kdb'. The request database contains requests for new certificates and always has a file name extension of '.rdb'. The database stash file contains the masked database password and always has a file name extension of '.sth'. Access to these files should be restricted to the database owner.

A certificate or request database consists of fixed-length records. The record length is specified when the database is created and must be large enough to contain the

largest certificate entry. A record length of 5000 should be sufficient for most applications. The record length can be increased if necessary after the database is created.

A temporary database file is created when a database is updated during **gskkyman** processing. The temporary database file is created using the same name as the database file with ".new" appended to the name. The database file is then rewritten and the temporary database file is deleted upon successful completion of the rewrite operation. The temporary database file is not deleted if an error occurs while rewriting the database file. If this happens, you can replace the database file with the temporary database file to recover from the error. If an error does occur and you do not rename or delete the temporary file, you receive an error on the next database update operation indicating the backup file exists.

If all certificates in a key database are displayed with the **-dc** or **-dcv** command, then all certificates with private keys are outputted, followed by all certificates without private keys. When displaying all certificates in a token, the certificates are displayed in the order that is returned from the token so that certificates with private keys might be interspersed with certificates without private keys.

Chapter 7. Updates to Chapter 13: Messages and codes

This topic lists the messages and codes issued by System SSL.

SSL function return codes

6 **Key label is not found.**

Explanation: The requested key label is not found in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. When using a PKCS #12 file, this error can also occur when the file is being processed during the establishment of the SSL/TLS environment when a certificate is encountered where there is no friendly name PKCS #12 attribute and the certificate's subject distinguished name is empty.

User response: Specify a label that exists in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. If encountered when establishing a SSL/TLS environment using a PKCS #12 file, verify any certificate that has no subject distinguished name is assigned a PKCS #12 friendly name attribute.

7 **No certificates available.**

Explanation: The key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token does not contain any certificates, or the SSL client application does not have a certificate available when authentication is requested by the server.

User response: Check for available certificates and add the user certificate and any necessary certification authority certificates to the key database, SAF key ring, or z/OS PKCS #11 token if necessary. If using a PKCS #12 file, ensure that the file contains the necessary certificates. If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available. Specify a certificate for the client application to use.

8 **Certificate validation error.**

Explanation: An error is detected while validating a certificate. This error can occur if a root CA certificate is not found in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token or if the certificate is not marked as a trusted certificate or if the certificate requires an algorithm or key size that is non-FIPS while executing in FIPS mode.

User response: Verify that the root CA certificate is in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token and is marked as trusted. Check all certificates in the certification chain and verify that they are trusted and are not expired. If the error occurred while executing in FIPS mode, check that only FIPS algorithms and key sizes are used by the certificate. If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available. Collect a System SSL trace that contains the error and then contact your service representative if the problem persists.

102 **Error detected while reading certificate database**

Explanation: An error is detected while reading the key database, the PKCS #12 file, or retrieving entries from the SAF key ring or z/OS PKCS #11 token.

User response: If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

202 **Error detected while opening the certificate database.**

Explanation: An error is detected while opening the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. This error can occur if no name is supplied or the database, PKCS #12 file, key ring, or token does not exist. When using a PKCS #12 file, the file name cannot end with .kdb, .rdb or .sth.

User response: Verify that the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token exists and is accessible by the application. This value is case-sensitive. Ensure that the case is preserved with your request. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

---

407 Key label does not exist.

Explanation: The supplied label or the default key is not found in the key database or the certificate is not trusted or the certificate uses algorithms or key sizes that are non-FIPS while executing in FIPS mode. If using a PKCS #12 file as the certificate database, the label is either the certificate's friendly name or the subject's distinguished name.

User response: Supply a valid label or define a default key in the key database or specify a label for a certificate that uses FIPS algorithms or key sizes if executing in FIPS mode. If using a PKCS #12 file, use the **gskkyman** command line option **-dc** or **-dcv** to display the contents of the PKCS #12 file. The friendly name or subject distinguished name values is displayed in the label field.

---

417 Self-signed certificate cannot be validated.

Explanation: A self-signed certificate cannot be validated because it is not in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token.

User response: Add the self-signed certificate to the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

---

442 Multiple certificates exist for label.

Explanation: Access of certificate/key from label could not be resolved because multiple certificates/keys exist with the label.

User response: Correct certificate/key store so that label specifies a unique record.

If using a PKCS #12 file, use the **gskkyman** command line option **-dc** or **-dcv** to display the contents of the PKCS #12 file. The friendly name or subject distinguished name value is displayed as the label. Ensure the specified label is unique in the PKCS #12 file.

---

490 PKCS #12 file content not valid

Explanation: When processing the PKCS #12 file, a format error was detected. This can occur if the file is not properly ASN.1 encoded, been modified if transferred, or the PKCS #12 file is not a Version 3 binary file. PKCS #12 Version 1 files and files in Base64 format are not supported.

User response: If the current file is either a PKCS #12 Version 1 file or a Base64 encoded file, it must be replaced with a PKCS #12 Version 3 file. If transferring the file, be sure to transfer the file in binary format. Correct the PKCS #12 file or obtain a new PKCS #12 file. If the problem persists, collect a System SSL Trace containing the error and then contact your service representative.

Deprecated SSL function return codes

---

1 Error detected while reading certificate database

Explanation: An error is detected while reading the key database, the PKCS #12 file, or retrieving entries from the SAF key ring or z/OS PKCS #11 token.

User response: Collect a System SSL trace containing the error and then contact your service representative.

---

2 Error detected while opening the certificate database.

Explanation: An error is detected while opening the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. This error can occur if no name is supplied or the database, key ring, or token does not exist. When using a PKCS #12 file, the file name cannot end with .kdb, .rdb or .sth.

User response: Verify that the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token exists and is

accessible by the application. This value is case-sensitive. Ensure that the case is preserved with your request. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

---

9 Key label does not exist.

Explanation: The supplied label or the default key is not found in the key database or the certificate is not trusted. If using a PKCS #12 file as the certificate database, the label is either the certificate's friendly name or the subject's distinguished name. A default key does not exist in a PKCS #12 file.

User response: Supply a valid label or define a default key in the key database. If using a PKCS #12 file, use the **gskkyman** command line option **-dc** or **-dcv** to display the contents of the PKCS #12 file. The friendly name or subject distinguished name values is displayed in the label field.

---

12 Key label is not found.

Explanation: The requested key label is not found in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. When using a PKCS #12 file, this error can also occur when the file is being processed during the establishment of the SSL/TLS environment when a certificate is encountered where there is no friendly name PKCS #12 attribute and the certificate's subject distinguished name is empty.

User response: Specify a label that exists in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. If encountered when establishing a SSL/TLS environment using a PKCS #12 file, verify any certificate that has no subject distinguished name is assigned a PKCS #12 friendly name attribute.

-18 Self-signed certificate cannot be validated.

Explanation: A self-signed certificate cannot be validated because it is not in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token.

User response: Add the self-signed certificate to the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

-35 Certificate validation error.

Explanation: An error is detected while validating a certificate. This error can occur if a root CA certificate is not found in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token or if the certificate is not marked as a trusted certificate or if the certificate requires an algorithm or key size that is non-FIPS while executing in FIPS mode.

User response: Verify that the root CA certificate is in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token and is marked as trusted. Check all certificates in the certification chain and verify that they are trusted and are not expired. Collect a System SSL trace containing the error and then contact your service representative if the problem persists. If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

-56 Multiple certificates exist for label.

Explanation: Access of certificate/key from label could not be resolved because multiple certificates/keys exist with the label.

User response: Correct certificate/key store so that label specifies a unique record.

If using a PKCS #12 file, use the **gskkyman** command line option **-dc** or **-dcv** to display the contents of the PKCS #12 file. The friendly name or subject distinguished name value is displayed as the label. Ensure that the specified label is unique in the PKCS #12 file.

-124 PKCS #12 file content not valid

Explanation: When processing the PKCS #12 file, a format error was detected. This can occur if the file is not properly ASN.1 encoded, been modified if transferred, or the PKCS #12 file is not a Version 3 binary file. PKCS #12 Version 1 files and files in Base64 format are not supported.

User response: If the current file is either a PKCS #12 Version 1 file or a Base64 encoded file, it must be replaced with a PKCS #12 Version 3 file. If transferring the file, be sure to transfer the file in binary format. Correct the PKCS #12 file or obtain a new PKCS #12 file. If the problem persists, collect a System SSL Trace containing the error and then contact your service representative.

CMS status codes

033530AB PKCS #12 input certificate has no subject DN or friendly name

Explanation: When reading the certificates of the specified PKCS #12 file, a certificate was encountered that had no subject distinguished name or PKCS #12 friendly name.

User response: Verify that all certificates within the provided PKCS #12 file have either a subject distinguished name or a friendly name attribute. The friendly name attribute or the subject distinguished name is used to create the certificate's label.

033530AC PKCS #12 file name may not end with .kdb, .rdb or .sth

Explanation: A PKCS #12 file name cannot end with .kdb, .rdb or .sth.

User response: Verify that the PKCS #12 file name does not end with .kdb, .rdb or .sth. If it does, it needs to be renamed.

Chapter 8. Updates to Appendix A: Environment variables

This topic discusses the use of the z/OS shell-based gskkyman utility to manage private keys, certificates, and tokens.

Environment variables

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

Table 1. SSL-Specific environment variables

Environment variables	Usage	Valid values
GSK_KEYRING_FILE	Specifies the name of the key database file, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. A key database or PKCS #12 file is used if the GSK_KEYRING_PW environment variable is also specified. A key database file is used if GSK_KEYRING_STASH environment variable is also specified. Otherwise, a SAF key ring or z/OS PKCS #11 token is used. Note that certificate private keys are not available when using a SAF key ring owned by another user. The user must have READ access to resource USER.tokenname in the CRYPTOZ class.	The SAF key ring name is specified as "userid/keyring". The current user ID is used if the user ID is omitted. The z/OS PKCS #11 token name is specified as "*TOKEN*/token-name". If no certificate source is specified, defaults to NULL.
GSK_KEYRING_PW	Specifies the password for the key database or PKCS #12 file.	NULL or value consisting of up to 128 characters. The default value is NULL

Printed in USA