Common Security Services Manager

Certificate Library Interface (CLI) Specification

Release 1.0
Updated November 1996

Subject to Change Without Notice

Specification Disclaimer and Limited Use License

This specification is for release version 1.0, October 1996.

THIS SPECIFICATION IS PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Some aspects of this Specification may be covered under various United States or foreign patents. No license, express or implied, by estoppel or otherwise, to any other intellectual property rights is granted herein.

Intel disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information in this specification. Intel doesn't warrant or represent that such implementation(s) will not infringe such rights.

If you are interested in receiving an appropriate license to Intel's intellectual property rights relating to the interface defined in this specification, contact us for details at cdsa@ibeam.intel.com.

Intel Corporation, 5200 N.E. Elam Young Parkway, Hillsboro, OR 97124-6497

*Other product and corporate names may be trademarks of other companies and are used only for explanation and to the owner’s benefit, without intent to infringe.

Table of Contents

List of Figures

1. Introduction

1.1 CDSA Overview

The Common Data Security Architecture (CDSA) defines the infrastructure for a complete set of security services. CDSA is an extensible architecture that provides mechanisms to manage add-in security modules that use cryptography as a computational base to build security protocols and security systems. Figure 1 shows the four basic layers of the Common Data Security Architecture: Applications, System Security Services, the Common Security Services Manager, and Security Add-in Modules. The Common Security Services Manager (CSSM) is the core of CDSA. It provides a means for applications to directly access security services through the CSSM Security API, or to indirectly access security services via layered security services and tools implemented over the CSSM API. CSSM manages the add-in security modules and directs application calls through the CSSM API to the selected add-in module that will service the request. Add-in modules perform various aspects of security services, including:

- Cryptographic Services
- Trust Policy Services
- Certificate Library Services
- Data-Storage Library Services

Cryptographic Service Providers (CSPs) are add-in modules that perform cryptographic operations including encryption, decryption, digital signaturing, key pair generation, random number generation, and key exchange. Trust Policy (TP) modules implement policies defined by authorities and institutions, such as VeriSign* (as a certificate authority) or MasterCard* (as an institution). Each trust policy module embodies the semantics of a trust model based on using digital certificates as credentials. Applications may use a digital certificate as an identity credential and/or an authorization credential. Certificate library (CL) modules provide format-specific, syntactic manipulation of memory-resident digital certificates and certificate revocation lists. Data-Storage Library (DL) modules provide persistent storage for certificates and certificate revocation lists.

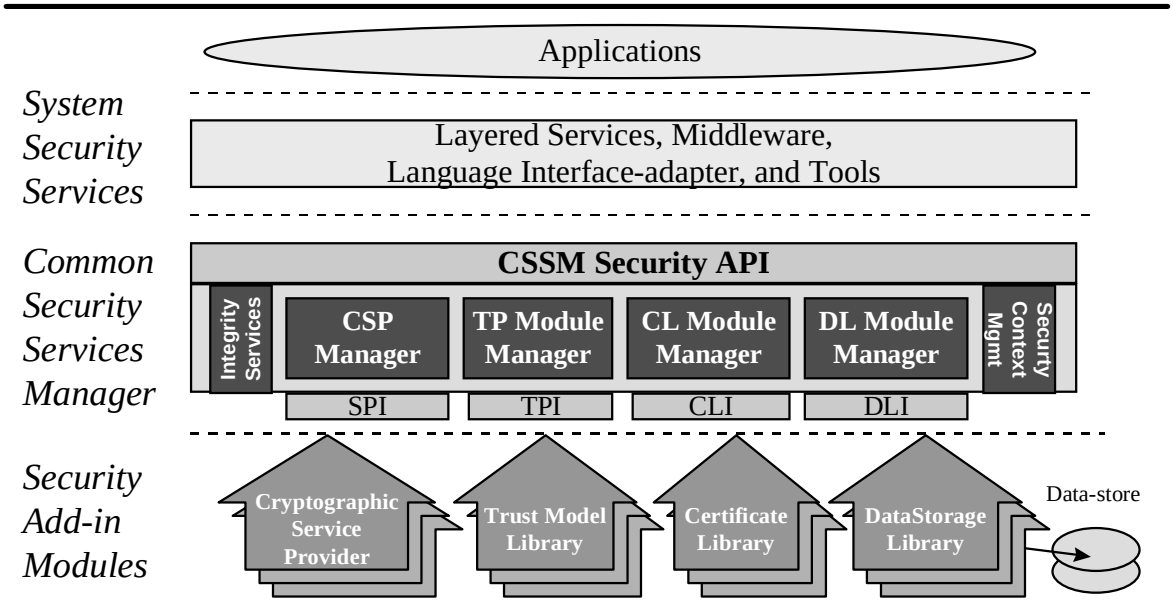

Figure 1. The Common Data Security Architecture for all platforms.

Applications directly or indirectly select the modules used to provide security services to the application. These add-in modules will be provided by independent software and hardware vendors. The functionality of the add-in module may be extended beyond the services defined by the CSSM API, by exporting additional services to applications via the CSSM PassThrough mechanism.

The API calls defined for add-in modules are categorized as service operations, module management operations, and module-specific operations. Service operations include functions that perform a security operation, such as encrypting data, inserting a certificate revocation list into a data-source, or verifying that a certificate is trusted. Module management functions support installation of modules, registration of module features and attributes, and queries to retrieve information on module availability and features. Module-specific operations are enabled in the API through pass-through functions whose behavior and use is defined by the add-in module developer.

CSSM also provides integrity services and security context management. CSSM applies the integrity check facility to itself to ensure that the currently executing instance of CSSM code has not been altered.

Security context management provides secured runtime caching of user-specific state information and secrets. The manager focuses on caching state information and parameters for performing cryptographic operations. Examples of secrets that must be cached during application execution include the application's private key and the application's digital certificate.

In summary, the CSSM provides these services through its API calls:

- Certificate-based services and operations
- Comprehensive, extensible SPIs for cryptographic service provider modules, trust policy modules, certificate library modules, and data storage modules
- Registration and management of available cryptographic service provider modules, trust policy modules, certificate library modules, and data storage modules
- Caching of keys and secrets required as part of the runtime context of a user application
- Call-back functions for disk, screen, and keyboard I/O supported by the operating system
- A test-and-check function to ensure CSSM integrity
- Management of concurrent security operations

1.2 Certificate Library Overview

A certificate library (CL) module performs syntactic manipulations on certificates and their associated certificate revocation lists (CRLs). The certificate library encapsulates the format-specific knowledge into a library, which an application can access via CSSM. These libraries allow applications and add-in modules to interact with certificates and CRLs for services such as signing, verification, creation and revocation without requiring knowledge of the certificate and CRL formats.

1.2.1 Application Interaction

An application determines the availability and basic capabilities of a certificate library by querying the CSSM Registry. When a new CL is installed on a system, the certificate types and certificate fields that it supports must be registered with CSSM. An application uses that information to find an appropriate CL and to request that CSSM attach to the CL. When CSSM attaches to the CL, it will return a CL handle to the application that uniquely identifies the pairing of the application thread to the CL module instance. This handle will be used by the application to identify the CL in future function calls.

CSSM passes function calls from an application to a certificate library by making use of the CL's function table. The function table consists of pointers to the subset of certificate functions from the CSSM API that are supported by the CL. When an application attaches the CL to CSSM, the certificate library registers its function table with CSSM using *CSSM_CL_RegisterServices*. During future function calls from the application, CSSM will use these function pointers to appropriately direct the call.

The allocation and de-allocation of all memory that is passed into or out of the certificate library module is the responsibility of the calling application. The application must register memory allocation and de-allocation upcalls with CSSM when it requests a CL attach. These upcalls and the handle identifying the application/CL pairing are passed to the CL when it calls *CSSM_CL_RegisterServices*. Whenever a certificate library allocates or de-allocates memory that belongs to or will belong to the application, these functions must be used. The CL handle will be used as the mechanism to associate the application with the instance of the certificate library module.

1.2.2 CL Structure and Use

A certificate library is composed of functions that are invoked when the CL is attached and detached and of functions that mirror the CSSM API for certificate functions. When the CL is attached, it registers its function table with the CSSM registry and CSSM provides the CL with a set of upcalls for memory management. When the CL is detached, any necessary cleanup actions are performed. The remainder of

the certificate library functions perform basic syntactic operations on certificates and certificate revocation lists (CRLs). These operations include creating and querying certificates and CRLs, signing and verifying, and performing module-specific syntactic operations. Module-specific operations are supported by the CL_PassThrough function. This function will pass a CL module-defined operation identifier and parameters to the CL.

Certificate Libraries may make use of other CSSM add-in modules to implement their functionality. For example, a certificate library may use the capabilities of a CSP add-in module to perform the cryptographic operations of sign and verify. In that case, the certificate library module could package the certificate or CRL fields to be signed or verified, attach to the appropriate CSP add-in module, and call *CSSM_SignData* or *CSSM_VerifyData* to perform the operation. More information about CSP modules can be found in the *Common Data Security Architecture Specification* and in the *CSSM Cryptographic Service Provider Interface Specification.*

Similarly, Certificate Libraries may be used by other CSSM add-in modules to implement their functionality. Trust Policy modules may choose to perform the syntactic verification of trust by calling a certificate library. Data-Storage Library modules may use a certificate library to obtain the individual fields of a certificate or CRL to be stored. More information about these modules can be found in the *Common Data Security Architecture Specification*, the *CSSM Trust Policy Interface Specification,* and the *CSSM Data-Storage Library Interface Specification*.

1.3 CSSM Certificate Library Interface Specification

1.3.1 Intended Audience

This document is intended for use by Independent Software Vendors (ISVs) who will develop their own Certificate Libraries to support a certificate and/or CRL format. These ISVs will be highly experienced software and security architects, advanced programmers, and sophisticated users. They are familiar with network operating systems, high-end cryptography, and digital certificates. It is assumed that this audience is familiar with the basic capabilities and features of the protocols they are considering.

1.3.2 Document Organization

This document is divided into the following sections:
Section 2, Certificate Library Interface, describes the functions that a certificate library makes available to applications via the CSSM.

Section 3, Certificate Library Structure & Management, describes important considerations in developing a certificate library. It also gives examples of how several certificate library functions might be implemented.

1.4 References

PKCS* — *The Public-Key Cryptography Standards*, RSA Laboratories, Redwood City, CA: RSA Data Security, Inc.

X.509 — *CCITT. Recommendation X.509: The Directory – Authentication Framework*. 1988. CCITT stands for Comite Consultatif Internationale Telegraphique et Telphonique (International Telegraph and Telephone Consultative Committee)

SPKI — *Simple Public Key Certificate,* InternetDraft, Carl M. Ellison, Bill Frantz, Brian M. Thomas, 1996

SDSI — *SDSI - A Simple Distributed Security Infrastructure,* R. Rivest and B. Lampson, 1996

CDSA	*Common Data Security Architecture Specification,* Intel Architecture Labs, 1996
CSSM API	*CSSM Application Programming Interface,* Intel Architecture Labs, 1996
CSSM SPI	*CSSM Cryptographic Service Provider Interface Specification,* Intel Architecture Labs, 1996
CSSM TPI	*CSSM Trust Policy Interface Specification,* Intel Architecture Labs, 1996
CSSM CLI	*CSSM Certificate Library Interface Specification,* Intel Architecture Labs, 1996
CSSM DLI	*CSSM Data-Storage Library Interface Specification,* Intel Architecture Labs, 1996
CSSM Java	*CSSM Java Application Programming Interface Specification,* Intel Architecture Labs, 1996

2. Certificate Library Interface

2.1 Overview

The Certificate Library Interface (CLI) specifies the functions that a certificate library may make available to applications via CSSM in order to support a certificate and a certificate revocation list (CRL) format. These functions mirror the CSSM API for certificates and certificate revocation lists. They include the basic areas of functionality expected of a certificate library: certificate operations, certificate revocation list operations, extensibility functions, and module management functions. The certificate library developer may choose to implement some or all of these CLI functions. The available functions will be made known to CSSM at attach time when it receives the certificate library's function table. In the function table, any unsupported function will have a NULL function pointer. It is the responsibility of the certificate library module developer to make its certificate format and general functionality known to application developers.

Certificate operations fall into three general areas:

- **Cryptographic operations** - These operations include signing a certificate and verifying the signature on a certificate. It is expected that the certificate library will determine the certificate fields to be signed or verified and will manage the interaction with a cryptographic service provider to perform the signing or verification.
- **Certificate field management** - Fields are added to a certificate when it is created. After the certificate is signed, the fields cannot be modified in any way. However, they can be queried for their values using the CSSM certificate interface.
- **Certificate format translation** - In the heterogeneous world of multiple certificate formats, CL modules may want to provide the service of translating between certificate formats. This translation would involve mapping the fields from one certificate format into another certificate format, while maintaining the original format for integrity verification purposes. For example, an X509V1 certificate may be exported to a SDSI format or imported into an X509V3 certificate, but the original data and signature must somehow be maintained. The supported import and export types are registered with CSSM as part of CL installation.

To support new certificate types and new uses of certificates, the sign and verify operations in the Certificate Library Interface support a scope parameter. The scope parameter enables an application to sign a portion of the certificate, namely the fields identified by the scope. This enables future certificate models, which are expected to allow field signing. CL modules that support existing certificate formats, such as X.509 Version 1, which sign and verify a pre-defined portion of the certificate, will ignore this parameter.

The CL module's certificate format is exposed via its fields. These fields will consist of tag/value pairs, where the tag is an object identifier (OID). These OIDs reference specific data types or data structures within the certificate or CRL. OIDs are defined by the certificate library developer at a granularity appropriate for the expected usage of the CL.

Operations on certificate revocation lists are comprised of cryptographic operations and field management operations on the CRL as a whole and on individual revocation records. The entire CRL can be signed or verified. This will ensure the integrity of the CRL's contents as it is passed between systems. Individual revocation records are signed when they are revoked and verified when they are queried. Certificates may be revoked and unrevoked by adding or removing them from the CRL at any time prior to its being signed. The contents of the CRL can be queried for all of its revocation records, specific certificates, or individual CRL fields.

A pass-through function is included in the Certificate Library Interface to allow certificate libraries to expose additional services beyond what is currently defined in the CSSM API. These services should be syntactic in nature, meaning that they should be dependent on the data format of the certificates and CRLs manipulated by the library. CSSM will pass an operation identifier and input parameters from the application to the appropriate certificate library. Within the CL_PassThrough function in the certificate library, the input parameters will be interpreted and the appropriate operation performed. The certificate library developer is responsible for making known to the application the identity and parameters of the supported pass-through operations.

A certificate library module must provide support for the CL_Initialize and CL_Uninitialize operations. The CL_Initialize operation is used by CSSM to verify that the CL module version that is attached is compatible with the CL module version requested by the calling application. It is called as part of *CSSM_CL_Attach*, immediately following the certificate library's registration of its function table. If the versions are incompatible, CSSM will detach the CL and the *CSSM_CL_Attach* operation will fail. CL_Uninitialize is called by CSSM as part of the *CSSM_CL_Detach* operation, immediately prior to detaching the CL module. CL_Uninitialize should be used by the CL module to cleanup any residual state information.

2.1.1 Certificate Operations

This section provides a more detailed look at the functions that compose the certificate operations in the CLI. It gives a high-level overview of each function's expected operation, its parameter definitions where necessary, and potential differences between CL module implementations.

CL_CertSign ( ) - This function will create a digital signature for the subject certificate using the signer's certificate. The cryptographic context handle indicates the algorithm and parameters to be used for signing. Which field or fields should be signed will depend on the implementation of the CL module. A CL module that supports X.509 Version 1 certificates will sign all of the certificate fields, ignoring the SignScope parameter. A CL module that supports field signing would sign the subset of fields specified by the SignScope parameter.

CL_CertUnsign ( ) - This function will remove the signer certificate's signature from the subject certificate. If the certificate library supports multiple signatures on the same certificate, the SignScope may be used to uniquely identify the signature to be removed.

CL_CertVerify ( ) - This function will verify the signer certificate's signature on the subject certificate. The cryptographic context handle indicates the algorithm and parameters to be used for verification. If the certificate library module supports field signing, the VerifyScope parameter may be used to identify the fields that were signed.

CL_CertCreate ( ) - This function creates a certificate in the CL module's native certificate format from the OID/value pairs provided by the application. The CL module makes its supported OIDs available to the application via the CertTemplate registered with CSSM and via the CL_CertDescribeFormat function. The CL Module is responsible for indicating which fields are required to create a certificate. The returned certificate will not be a valid certificate until it has been signed.

CL_CertView ( ) - This function returns an array of all of the viewable fields in the certificate. The fields are identified by OID/value pairs. The field values are in a displayable format.

CL_CertGetFirstFieldValue ( ) - This function returns the first field in the certificate that matches the input OID. If the certificate contains more than one instance of the requested OID, the CL module will return a handle to be used to obtain the additional instances and a count of the total number of instances of this OID in the certificate. The application obtains the additional matching instances by repeated calls to CL_CertGetNextFieldValue.

CL_CertGetNextFieldValue ( ) - This function returns the next field that matched the OID given in the CL_CertGetFirstFieldValue function. It will only be supported by certificate library modules that allow multiple instances of an OID in a single certificate.

CL_CertAbortQuery ( ) - This function releases the handle that was assigned by the CL_CertGetFirstFieldValue function to identify the results of a certificate query. It will only be supported by certificate library modules that allow multiple instances of an OID in a single certificate.

CL_CertGetKeyInfo ( ) - This function retrieves the public key information stored in the certificate. In most certificate formats this includes multiple fields, but it may not include all of the fields defined by the CSSM_KEY data structure. Each CL module is responsible for making known which portions of the CSSM_KEY data structure will be returned.

CL_CertGetAllFields ( ) - This function returns a list of all the fields in the input certificate, as described by their OID/value pairs.

CL_CertImport ( ) - This function translates a certificate from a foreign certificate type to the native certificate type manipulated by the CL module.

CL_CertExport ( ) - This function translates a certificate from the native certificate type manipulated by the CL module into a foreign certificate type.

CL_CertDescribeFormat ( ) - This function returns a list of object identifiers corresponding to the data objects composing the CL module's native certificate format.

2.1.2 Certificate Revocation List Operations

This section provides a more detailed look at the functions that compose the certificate revocation list operations in the CLI. This section gives a high-level overview of each function's expected operation, its parameter definitions where necessary, and potential differences between CL module implementations.

CL_CrlCreate ( ) - This function creates an empty CRL in the native format of the CL module. CRL queries may be performed on both signed and unsigned CRLs.

CL_CrlAddCert ( ) - This function revokes the input certificate by adding a record representing the certificate to the CRL. It then uses the revoker's certificate to sign the new record. The updated CRL is returned to the calling application.

CL_CrlRemoveCert ( ) - This function unrevokes the input certificate by removing the record representing the certificate from the CRL. The updated CRL is returned to the calling application.

CL_CrlSign ( ) - This function will create a digital signature for the entire CRL using the signer's certificate. The cryptographic context handle indicates the algorithm and parameters to be used for signing. The field or fields of the CRL that should be signed will depend on the implementation of the CL module. A CL module may choose to ignore the SignScope parameter if the fields to be signed are pre-defined. A CL module that supports field signing would sign the subset of fields specified by the SignScope parameter. Typically, this function will be used to sign the entire CRL prior to distributing it to other systems. The signature will be used to quickly detect tampering of the CRL. CRL queries may be performed on both signed and unsigned CRLs.

CL_CrlVerify ( ) - This function will check the signer certificate's signature on the subject CRL to determine whether any record in the CRL has been tampered with and whether the signer's certificate was actually used to sign the CRL. The cryptographic context handle indicates the algorithm and parameters to be used for verification. If the certificate library supports field signing on a CRL, the VerifyScope may be used to identify the fields that were signed.

CL_IsCertInCrl ( ) - This function searches the CRL for a record corresponding to the input certificate.

CL_CrlGetFirstFieldValue ( ) - This function returns the first field in the CRL that matches the input OID. It is likely that the CRL will support multiple instances of an OID that represents a revoked certificate record. If an application requests an OID that has multiple instances within the CRL, a results handle and a count of the number of matching instances will be returned along with the first instance of the OID. The application uses the results handle to obtain the additional matching instances by repeated calls to CL_CrlGetNextFieldValue. For example, given the OID for "revocation record", this function would return the first revocation record in the CRL. The remaining revocation records could be obtained by successive calls to CL_CrlGetNextFieldValue.

CL_CrlGetNextFieldValue ( ) - This function returns the next field that matches the OID given in the CL_CrlGetFirstFieldValue function.

CL_CrlAbortQuery ( ) - This function releases a handle that was assigned by the CL_CrlGetFirstFieldValue function to identify the results of a CRL query.

CL_CrlDescribeFormat ( ) - This function returns a list of the object identifiers that represent the fields in the certificate revocation list format manipulated by the CL module.

2.1.3 Extensibility Functions

CL_PassThrough ( ) - This performs the CL module-specific function indicated by the operation ID. The operation ID specifies an operation that the CL has exported for use by an application or module. Such operations should be specific to the data format of the certificates and CRLs manipulated by the CL module.

2.1.4 Module Management Functions

CL_Initialize ( ) - This function checks whether the version of the attached CL module is compatible with the input version number and performs CL module setup activities. It is called by the CSSM Core as part of the *CSSM_CL_Attach* routine. It is called immediately after the CL module's function table is registered with CSSM. If the versions are incompatible, the CL module is detached, a CSSM_INCOMPATIBLE_VERSION error is set, and a NULL handle is returned to the calling application.

CL_Uninitialize ( ) - This function checks performs CL module cleanup activities. It is called by the CSSM Core as part of the *CSSM_CL_Detach* routine. It is called immediately prior to the detach of the CL module.

2.2 Data Structures

This section describes the data structures that may be passed to or returned from a certificate library function. They will be used by applications to prepare data to be passed as input parameters into CSSM API function calls which will be passed without modification to the appropriate CL. The CL is then responsible for interpreting them and returning the appropriate data structure to the calling application via CSSM. These data structures are defined in the header file cssm.h distributed with CSSM.

2.2.1 CSSM_CL_HANDLE

The CSSM_CL_HANDLE is used to identify the association between an application thread and an instance of a CL module. It is assigned when an application causes CSSM to attach to a certificate library. It is freed when an application causes CSSM to detach from a certificate library. The application uses the CSSM_CL_HANDLE with every CL function call to identify the targeted CL. The CL module uses the CSSM_CL_HANDLE to identify the appropriate application's memory management routines when allocating memory on the application's behalf.

```
typedef uint32 CSSM_CL_HANDLE
```

2.2.2 CSSM_CERT_TYPE

This variable specifies the type of certificate format supported by a certificate library and the types of certificates understood for import and export. They are expected to define such well-known certificate formats as X.509 Version 3 and SDSI as well as custom certificate formats.

```
typedef uint32 CSSM_CERT_TYPE, *CSSM_CERT_TYPE_PTR
```

2.2.3 CSSM_DATA

The CSSM_DATA structure is used to associate a length, in bytes, with an arbitrary block of contiguous memory. This memory must be allocated and freed using the memory management routines provided by the calling application via CSSM.

```
typedef struct cssm_data {
    uint32 Length;
    uint8* Data;
} CSSM_DATA, *CSSM_DATA_PTR
```

Definition:

Length - The length, in bytes, of the memory block pointed to by *Data*

Data - A pointer to a contiguous block of memory.

2.2.4 CSSM_OID

The object identifier (OID) is used to identify the data types and data structures that comprise the fields of a certificate or CRL.

```
typedef CSSM_DATA CSSM_OID
```

2.2.5 CSSM_FIELD

This structure contains the OID/value pair for a certificate or CRL field.

```
typedef struct cssm_field {
    CSSM_OID FieldOid;
    CSSM_DATA FieldValue;
}CSSM_FIELD, *CSSM_FIELD_PTR
```

Definition:

FieldOid - The object identifier that identifies the certificate or CRL data type or data structure.

FieldValue - A CSSM_DATA type that contains the value of the specified OID in a contiguous block of memory.

2.2.6 CSSM_KEYHEADER

```
typedef struct CSSM_KeyHeader{
    CSSM_GUID CspId;
    uint32 BlobType;
    uint32 FormatVersion;
    uint32 AlgorithmId;
    uint32 AlgorithmMode;
    uint32 SizeInBits;
    uint32 WrapMethod;
    uint32 Reserved;
} CSSM_KEYHEADER, *CSSM_KEYHEADER_PTR
```

Definition:

CspId - Globally unique Id of the CSP that generated the key (if appropriate).

BlobType - Key blob type. The key blob types currently-defined are CSSM_SESSION_KEY_BLOB, CSSM_RSA_PUBLIC_KEY_BLOB, CSSM_RSA_PRIVATE_KEY_BLOB, CSSM_DSA_PUBLIC_KEY_BLOB, and CSSM_DSA_PRIVATE_KEY_BLOB.

FormatVersion - Version number of the key blob format. Current value is 0x01.

AlgorithmId - Algorithm identifier for the key contained by the key blob.

AlgorithmMode - Algorithm mode for the key contained by the key blob

SizeInBits - Size of the key in bits.

WrapMethod - Key wrapping scheme. The key wrapping methods currently-defined are CSSM_KEYWRAP_NONE, CSSM_KEYWRAP_MD5WithDES, CSSM_KEYWRAP_MD5WithIDEA, CSSM_KEYWRAP_SHAWithDES, and CSSM_KEYWRAP_SHAWithIDEA.

Reserved - Reserved for future use.

2.2.7 CSSM_KEYBLOB

This is the data structure which contains both information about the key and the key data itself. This structure allows the passage of keys as one contiguous unit of data.

```
typedef struct cssm_keyblob{
       CSSM_KEYHEADER KeyHeader;
       uint8 KeyData[MAX_KEYBLOB_LEN];
} CSSM_KEYBLOB, *CSSM_KEYBLOB_PTR;
```

Definition:

KeyHeader - Key header for the key.

KeyData - Data representation of the key.

2.2.8 CSSM_KEY

```
typedef struct cssm_key{
    uint32             KeyBlobLength;
    CSSM_KEYBLOB_PTR   KeyBlob;
} CSSM_KEY, *CSSM_KEY_PTR
```

Definition:

KeyBlobLength - Length of the key blob.

KeyBlob - Pointer to a key blob which holds all of the data associated with the key.

2.2.9 CSSM_REVOKE_REASON

This list defines the possible reasons why a certificate may be revoked.

```
typedef enum cssm_revoke_reason {
    CSSM_REVOKE_CUSTOM,
    CSSM_REVOKE_UNSPECIFIC,
    CSSM_REVOKE_KEYCOMPROMISE,
    CSSM_REVOKE_CACOMPROMISE,
    CSSM_REVOKE_AFFILIATIONCHANGED,
    CSSM_REVOKE_SUPERCEDED,
    CSSM_REVOKE_CESSATIONOFOPERATION,
    CSSM_REVOKE_CERTIFICATEHOLD,
    CSSM_REVOKE_CERTIFICATEHOLDRELEASE,
    CSSM_REVOKE_REMOVEFROMCRL
} CSSM_REVOKE_REASON
```

2.3 Certificate Operations

This section describes the function prototypes and error codes expected for the functions in the CLI. The functions will be exposed to CSSM via a function table, so the function names may vary at the discretion of the certificate library developer. However, the function parameter list and return type must match the prototypes given in this section in order to be used by applications. The error codes given in this section constitute the generic error codes that are defined by CSSM for use by all certificate libraries in describing common error conditions. A certificate library developer may also define their own module-specific error codes, as described in Section 3.5.2.

2.3.1 CL_CertSign

CSSM_DATA_PTR CSSMCLI CL_CertSign (CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR SubjectCert,
const CSSM_DATA_PTR SignerCert,
const CSSM_FIELD_PTR SignScope,
uint32 ScopeSize)

This function signs the fields of the input certificate as indicated by the *SignScope* array.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

CCHandle (input)
The handle that describes the context of this cryptographic operation.

SubjectCert (input)
A pointer to the CSSM_DATA structure containing the certificate to be signed.

SignerCert (input)
A pointer to the CSSM_DATA structure containing the certificate to be used to sign the subject certificate.

SignScope (input)
A pointer to the CSSM_FIELD array containing the tag/value pairs of the fields to be signed. A null input signs all the fields in the certificate.

ScopeSize (input)
The number of entries in the sign scope list.

Return Value

A pointer to the CSSM_DATA structure containing the signed certificate. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid Certificate Library Handle
CSSM_CL_INVALID_CC_HANDLE	Invalid Cryptographic Context Handle
CSSM_CL_INVALID_DATA_POINTER	Invalid pointer input

CSSM_CL_INVALID_CONTEXT	Invalid context for the requested operation
CSSM_CL_UNKNOWN_FORMAT	Unrecognized certificate format
CSSM_CL_INVALID_SIGNER_CERTIFICATE	Revoked or expired signer certificate
CSSM_CL_INVALID_SCOPE	Invalid scope
CSSM_CL_MEMORY_ERROR	Not enough memory
CSSM_CL_CERT_SIGN_FAIL	Unable to sign certificate

See Also

CL_CertUnsign, CL_CertVerify

2.3.2 CL_CertUnsign

CSSM_DATA_PTR CSSMCLI CL_CertUnsign (CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR SubjectCert,
const CSSM_DATA_PTR SignerCert,
const CSSM_FIELD_PTR SignScope,
uint32 ScopeSize)

This function removes a signature from a signed, memory-resident certificate.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

CCHandle (input)
The handle that describes the context of this cryptographic operation.

SubjectCert (input)
A pointer to the CSSM_DATA structure containing the certificate from which to remove a signature.

SignerCert (input)
A pointer to the CSSM_DATA structure containing the signer's certificate. This certificate will be used to identify the signature to be removed.

SignScope (input)
A pointer to the CSSM_FIELD array containing the tag/value pairs of the fields that were signed. A null input indicates that all the fields in the certificate were signed.

ScopeSize (input)
The number of entries in the sign scope list.

Return Value

A pointer to the CSSM_DATA structure containing the newly-unsigned certificate. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid Certificate Library Handle
CSSM_CL_INVALID_CC_HANDLE	Invalid Cryptographic Context Handle
CSSM_CL_INVALID_DATA_POINTER	Invalid pointer input
CSSM_CL_INVALID_SCOPE	Invalid scope
CSSM_CL_MEMORY_ERROR	Not enough memory
CSSM_CL_CERT_UNSIGN_FAIL	Unable to unsign certificate

See Also

CL_CertSign

2.3.3 CL_CertVerify

CSSM_BOOL CSSMCLI CL_CertVerify (CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR SubjectCert,
const CSSM_DATA_PTR SignerCert,
const CSSM_FIELD_PTR VerifyScope,
uint32 ScopeSize)

This function verifies that the signed certificate has not been altered since it was signed by the designated signer. It does this by verifying the digital signature on the VerifyScope fields.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

CCHandle (input)
The handle that describes the context of this cryptographic operation.

SubjectCert (input)
A pointer to the CSSM_DATA structure containing the signed certificate.

SignerCert (input)
A pointer to the CSSM_DATA structure containing the certificate used to sign the subject certificate.

VerifyScope (input)
A pointer to the CSSM_FIELD array containing the tag/value pairs of the fields to be verified. A null input verifies all the fields in the certificate.

ScopeSize (input)
The number of entries in the verify scope list.

Return Value

CSSM_TRUE if the certificate verified. CSSM_FALSE if the certificate did not verify or an error condition occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid Certificate Library Handle
CSSM_CL_INVALID_CC_HANDLE	Invalid Cryptographic Context Handle
CSSM_CL_INVALID_DATA_POINTER	Invalid pointer input
CSSM_CL_INVALID_CONTEXT	Invalid context for the requested operation
CSSM_CL_UNKNOWN_FORMAT	Unrecognized certificate format
CSSM_CL_INVALID_SCOPE	Invalid scope
CSSM_CL_CERT_VERIFY_FAIL	Unable to verify certificate

See Also

CL_CertSign

2.3.4 CL_CertCreate

CSSM_DATA_PTR CSSMCLI CL_CertCreate (CSSM_CL_HANDLE CLHandle,
const CSSM_FIELD_PTR CertTemplate,
uint32 NumberOfFields)

This function allocates and initializes memory for a certificate based on the input OID/value pairs. The memory is allocated using the calling application's memory management routines.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

CertTemplate (input)
A pointer to an array of OID/value pairs that identify the field values of the new certificate.

NumberOfFields (input)
The number of certificate fields being input. This number should indicate the length of the *CertTemplate* array.

Return Value

A pointer to the CSSM_DATA structure containing the new certificate. If the return pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid Certificate Library Handle
CSSM_CL_INVALID_FIELD_POINTER	Invalid pointer input
CSSM_CL_INVALID_TEMPLATE	Invalid template for this certificate type
CSSM_CL_MEMORY_ERROR	Not enough memory
CSSM_CL_CERT_CREATE_FAIL	Unable to create certificate

See Also

CL_CertSign, CL_CertGetFirstFieldValue

2.3.5 CL_CertView

CSSM_FIELD_PTR CSSMCLI CL_CertView (CSSM_CL_HANDLE CLHandle,
const CSSM_DATA_PTR Cert,
uint32 *NumberOfFields)

This function returns the displayable fields of the input certificate.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

Cert (input)
A pointer to the CSSM_DATA structure containing the certificate whose displayable fields will be returned.

NumberOfFields (output)
The number of certificate fields being output. This number indicates the length of the output array.

Return Value

A pointer to an array of CSSM_FIELD structures (tag/value pairs) that identify the displayable field values of the *Cert*. If the return pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid Certificate Library Handle
CSSM_CL_INVALID_FIELD_POINTER	Invalid pointer input
CSSM_CL_INVALID_DATA_POINTER	Invalid pointer input
CSSM_CL_UNKNOWN_FORMAT	Unrecognized certificate format
CSSM_CL_MEMORY_ERROR	Not enough memory
CSSM_CL_CERT_VIEW_FAIL	Unable to view certificate

See Also

CL_CertGetFirstFieldValue, CL_CertGetAllFields

2.3.6 CL_CertGetFirstFieldValue

CSSM_DATA_PTR CSSMCLI CL_CertGetFirstFieldValue (CSSM_CL_HANDLE CLHandle,
const CSSM_DATA_PTR Cert,
const CSSM_OID_PTR CertField,
CSSM_HANDLE_PTR ResultsHandle,
uint32 *NumberOfMatchedFields)

This function returns the value of the designated certificate field. If more than one field matches the *CertField* OID, the first matching field will be returned. The number of matching fields is an output parameter, as is the ResultsHandle to be used to retrieve the remaining matching fields.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

Cert (input)
A pointer to the CSSM_DATA structure containing the certificate.

CertField (input)
A pointer to an object identifier that identifies the field value to be extracted from the *Cert*.

ResultsHandle (output)
A pointer to the CSSM_HANDLE that should be used to obtain any additional matching fields.

NumberOfMatchedFields (output)
The number of fields that match the *CertField* OID.

Return Value

A pointer to the CSSM_DATA structure containing the value of the requested field. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid Certificate Library Handle
CSSM_CL_INVALID_DATA_POINTER	Invalid pointer input
CSSM_CL_UNKNOWN_FORMAT	Unrecognized certificate format
CSSM_CL_UNKNOWN_TAG	Unknown field tag
CSSM_CL_MEMORY_ERROR	Not enough memory
CSSM_CL_NO_FIELD_VALUES	No field values for this results handle
CSSM_CL_CERT_GET_FIELD_VALUE_FAIL	Unable to get field value

See Also

CL_CertGetNextFieldValue, CL_CertAbortQuery, CL_CertGetAllFields, CL_CertDescribeFormat

2.3.7 CL_CertGetNextFieldValue

CSSM_DATA_PTR CSSMCLI CL_CertGetNextFieldValue (CSSM_CL_HANDLE CLHandle, CSSM_HANDLE ResultsHandle)

This function returns the next certificate field that matched the OID in a call to CL_CertGetFirstFieldValue.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

ResultsHandle (input)
The handle that identifies the results of a certificate query.

Return Value

A pointer to the CSSM_DATA structure containing the value of the requested field. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid Certificate Library Handle
CSSM_CL_INVALID_RESULTS_HANDLE	Invalid results handle
CSSM_CL_MEMORY_ERROR	Not enough memory
CSSM_CL_NO_FIELD_VALUES	No field values for this results handle
CSSM_CL_CERT_GET_FIELD_VALUE_FAIL	Unable to get field value

See Also

CL_CertGetFirstFieldValue, CL_CertAbortQuery

2.3.8 CL_CertAbortQuery

CSSM_RETURN CSSMCLI CL_CertAbortQuery (CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE ResultsHandle)

This function terminates the query initiated by CL_CertGetFirstFieldValue and allows the CL to release all intermediate state information associated with the query.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

ResultsHandle (input)
The handle that identifies the results of a certificate query.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error condition occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid Certificate Library Handle
CSSM_CL_INVALID_RESULTS_HANDLE	Invalid results handle
CSSM_CL_CERT_ABORT_QUERY_FAIL	Unable to abort query

See Also

CL_CertGetFirstFieldValue, CL_CertGetNextFieldValue

2.3.9 CL_CertGetKeyInfo

CSSM_KEY_PTR CSSMCLI CL_CertGetKeyInfo (CSSM_CL_HANDLE CLHandle, const CSSM_DATA_PTR Cert)

This function obtains information about the certificate's public key. Ideally, this information comprises the key fields the application needs to create a cryptographic context that uses this certificate's key.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

Cert (input)
A pointer to the CSSM_DATA structure containing the certificate from which to extract the public key information.

Return Value

A pointer to the CSSM_KEY structure containing the public key and possibly other key information. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid Certificate Library Handle
CSSM_CL_INVALID_DATA_POINTER	Invalid pointer input
CSSM_CL_UNKNOWN_FORMAT	Unrecognized certificate format
CSSM_CL_UNKNOWN_TAG	Unknown field tag
CSSM_CL_MEMORY_ERROR	Not enough memory
CSSM_CL_CERT_GET_KEY_INFO_FAIL	Unable to get key information

See Also

CL_CertGetFirstFieldValue

2.3.10 CL_CertGetAllFields

CSSM_FIELD_PTR CSSMCLI CL_CertGetAllFields(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA_PTR Cert,
uint32 *NumberOfFields)

This function returns a list of the fields in the input certificate, as described by their OID/value pairs.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

Cert (input)
A pointer to the CSSM_DATA structure containing the certificate whose fields will be returned.

NumberOfFields (output)
The length of the output CSSM_FIELD array.

Return Value

A pointer to an array of CSSM_FIELD structures that describe the contents of the certificate using OID/value pairs. If the return pointer is NULL, an er ror has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid handle
CSSM_CL_INVALID_POINTER	Invalid pointer
CSSM_CL_MEMORY_ERROR	Error allocating memory
CSSM_CL_CERT_GET_ALL_FIELDS_FAIL	Unable to return the list of fields

See Also

CL_CertGetFirstFieldValue, CL_CertView

2.3.11 CL_CertImport

CSSM_DATA_PTR CSSMCLI CL_CertImport (CSSM_CL_HANDLE CLHandle,
CSSM_CERT_TYPE ForeignCertType,
const CSSM_DATA_PTR ForeignCert)

This function imports a certificate from the input format into the native format of the specified certificate library.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

ForeignCertType (input)
A unique value that identifies the type of the certificate being imported.

Cert (input)
A pointer to the CSSM_DATA structure containing the certificate to be imported into the native type.

Return Value

A pointer to the CSSM_DATA structure containing the native-type certificate imported from the foreign certificate. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid Certificate Library Handle
CSSM_CL_INVALID_DATA_POINTER	Invalid pointer input
CSSM_CL_UNKNOWN_FORMAT	Unrecognized certificate format
CSSM_CL_MEMORY_ERROR	Not enough memory
CSSM_CL_CERT_IMPORT_FAIL	Unable to import certificate

See Also

CL_CertExport

2.3.12 CL_CertExport

CSSM_DATA_PTR CSSMCLI CL_CertExport (CSSM_CL_HANDLE CLHandle,
CSSM_CERT_TYPE TargetCertType,
const CSSM_DATA_PTR NativeCert)

This function exports a certificate from the native format of the specified certificate library into the specified target certificate format.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

TargetCertType (input)
A unique value that identifies the target type of the certificate being exported.

NativeCert (input)
A pointer to the CSSM_DATA structure containing the certificate to be exported.

Return Value

A pointer to the CSSM_DATA structure containing the target-type certificate exported from the native certificate. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid Certificate Library Handle
CSSM_CL_INVALID_DATA_POINTER	Invalid pointer input
CSSM_CL_UNKNOWN_FORMAT	Unrecognized certificate format
CSSM_CL _MEMORY_ERROR	Not enough memory
CSSM_CL_CERT_EXPORT_FAIL	Unable to export certificate

See Also

CL_CertImport

2.3.13 CL_CertDescribeFormat

CSSM_OID_PTR CSSMCLI CL_CertDescribeFormat (CSSM_CL_HANDLE CLHandle,
uint32 *NumberOfFields)

This function returns a list of the object identifiers used to describe the certificate format supported by the specified CL.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

NumberOfFields (output)
The length of the output OID array.

Return Value

A pointer to the array of CSSM_OID structures which are supported for certificate operations in the specified CL module. If the return pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid handle
CSSM_CL_INVALID_POINTER	Invalid pointer
CSSM_CL_MEMORY_ERROR	Error allocating memory
CSSM_CL_CERT_DESCRIBE_FORMAT_FAIL	Unable to return the list of OIDs

See Also

CL_CertGetFirstFieldValue

2.4 CRL Operations

This section describes the function prototypes and error codes expected for the functions in the CLI. The functions will be exposed to CSSM via a function table, so the function names may vary at the discretion of the certificate library developer. However, the function parameter list and return type must match the prototypes given in this section in order to be used by applications. The error codes given in this section constitute the generic error codes that are defined by CSSM for use by all certificate libraries in describing common error conditions. A certificate library developer may also define their own module-specific error codes, as described in Section 3.5.2.

2.4.1 CL_CrlCreate

CSSM_DATA_PTR CSSMCLI CL_CrlCreate (CSSM_CL_HANDLE CLHandle)

This function creates an empty, memory-resident CRL.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

Return Value

A pointer to the CSSM_DATA structure containing the new CRL. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid CL handle
CSSM_CL_MEMORY_ERROR	Not enough memory to allocate for the CRL
CSSM_CL_CRL_CREATE_FAIL	Unable to create CRL

See Also

2.4.2 CL_CrlAddCert

CSSM_DATA_PTR CSSMCLI CL_CrlAddCert (CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR Cert,
const CSSM_DATA_PTR RevokerCert,
CSSM_REVOKE_REASON RevokeReason,
const CSSM_DATA_PTR OldCrl)

This function revokes the input certificate by adding a record representing the certificate to the CRL. It uses the revoker's certificate to sign the new record in the CRL. The reason for revoking the certificate may also be stored in the revocation record.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

CCHandle (input)
The handle that describes the context of this cryptographic operation.

Cert (input)
A pointer to the CSSM_DATA structure containing the certificate to be revoked.

RevokerCert (input)
A pointer to the CSSM_DATA structure containing the revoker's certificate.

RevokeReason (input)
The reason for revoking the certificate.

OldCrl (input)
A pointer to the CSSM_DATA structure containing the CRL to which the newly revoked certificate will be added.

Return Value

A pointer to the CSSM_DATA structure containing the updated CRL. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid CL handle
CSSM_CL_INVALID_CC_HANDLE	Invalid Context Handle
CSSM_CL_INVALID_CERTIFICATE_PTR	Invalid Certificate
CSSM_CL_INVALID_CRL	Invalid CRL
CSSM_CL_MEMORY_ERROR	Not enough memory to allocate the CRL
CSSM_CL_CRL_ADD_CERT_FAIL	Unable to add certificate to CRL

See Also

CL_CrlRemoveCert

2.4.3 CL_CrlRemoveCert

CSSM_DATA_PTR CSSMCLI CL_CrlRemoveCert (CSSM_CL_HANDLE CLHandle,
const CSSM_DATA_PTR Cert,
const CSSM_DATA_PTR OldCrl)

This function unrevokes a certificate by removing it from the input CRL.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

Cert (input)
A pointer to the CSSM_DATA structure containing the certificate to be unrevoked.

OldCrl (input)
A pointer to the CSSM_DATA structure containing the CRL from which the certificate is to be removed.

Return Value

A pointer to the CSSM_DATA structure containing the updated CRL. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid CL handle
CSSM_CL_INVALID_CERTIFICATE_PTR	Invalid Certificate
CSSM_CL_INVALID_CRL	Invalid CRL
CSSM_CL_MEMORY_ERROR	Not enough memory to allocate the CRL
CSSM_CL_CRL_REMOVE_CERT_FAIL	Unable to remove certificate from CRL

See Also

CL_CrlAddCert

2.4.4 CL_CrlSign

CSSM_DATA_PTR CSSMCLI CL_CrlSign (CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR UnsignedCrl,
const CSSM_DATA_PTR SignerCert,
const CSSM_FIELD_PTR SignScope,
uint32 ScopeSize)

This function signs, in accordance with the specified cryptographic context, the fields of the CRL indicated in the *SignScope* parameter.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

CCHandle (input)
The handle that describes the context of this cryptographic operation.

UnsignedCrl (input)
A pointer to the CSSM_DATA structure containing the CRL to be signed.

SignerCert (input)
A pointer to the CSSM_DATA structure containing the certificate to be used to sign the CRL.

SignScope (*input*)
A pointer to the CSSM_FIELD array containing the tag/value pairs of the fields to be signed. A null input signs all the fields in the CRL.

ScopeSize (*input*)
The number of entries in the sign scope list.

Return Value

A pointer to the CSSM_DATA structure containing the signed CRL. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid CL handle
CSSM_CL_INVALID_CC_HANDLE	Invalid Context Handle
CSSM_CL_INVALID_CERTIFICATE_PTR	Invalid Certificate
CSSM_CL_INVALID_CRL_PTR	Invalid CRL pointer
CSSM_CL_INVALID_SCOPE_PTR	SignScope pointer is invalid
CSSM_CL_MEMORY_ERROR	Not enough memory to allocate the CRL
CSSM_CL_CRL_SIGN_FAIL	Unable to sign CRL

See Also

CL_CrlVerify

2.4.5 CL_CrlVerify

CSSM_BOOL CSSMCLI CL_CrlVerify (CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR SubjectCrl,
const CSSM_DATA_PTR SignerCert,
const CSSM_FIELD_PTR VerifyScope,
uint32 ScopeSize)

This function verifies that the signed CRL has not been altered since it was signed by the designated signer. It does this by verifying the digital signature on the VerifyScope fields.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

CCHandle (input)
The handle that describes the context of this cryptographic operation.

SubjectCrl (input)
A pointer to the CSSM_DATA structure containing the CRL to be verified.

SignerCert (input)
A pointer to the CSSM_DATA structure containing the certificate used to sign the CRL.

VerifyScope (*input*)
A pointer to the CSSM_FIELD array containing the tag/value pairs of the fields to be verified. A null input verifies all the fields in the CRL.

ScopeSize (*input*)
The number of entries in the verify scope list.

Return Value

A CSSM_TRUE return value signifies that the certificate revocation list verifies successfully. When CSSM_FALSE is returned, either the CRL verified unsuccessfully or an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid CL handle
CSSM_CL_INVALID_CC_HANDLE	Invalid Context Handle
CSSM_CL_INVALID_CERTIFICATE_PTR	Invalid Certificate
CSSM_CL_INVALID_CRL_PTR	Invalid CRL pointer
CSSM_CL_INVALID_SCOPE_PTR	VerifyScope pointer is invalid
CSSM_CL_MEMORY_ERROR	Not enough memory to allocate the CRL
CSSM_CL_CRL_VERIFY_FAIL	Unable to verify CRL

See Also

CL_CrlSign

2.4.6 CL_IsCertInCrl

CSSM_BOOL CSSMCLI CL_IsCertInCrl (CSSM_CL_HANDLE CLHandle,
const CSSM_DATA_PTR Cert,
const CSSM_DATA_PTR Crl)

This function searches the CRL for a record corresponding to the certificate.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

Cert (input)
A pointer to the CSSM_DATA structure containing the certificate to be located.

Crl (input)
A pointer to the CSSM_DATA structure containing the CRL to be searched.

Return Value

A CSSM_TRUE return value signifies that the certificate is in the CRL. When CSSM_FALSE is returned, either the certificate is not in the CRL or an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid CL handle
CSSM_CL_INVALID_CERTIFICATE_PTR	Invalid Certificate
CSSM_CL_INVALID_CRL_PTR	Invalid CRL pointer

See Also

2.4.7 CL_CrlGetFirstFieldValue

CSSM_DATA_PTR CSSMCLI CL_CrlGetFirstFieldValue (CSSM_CL_HANDLE CLHandle,
const CSSM_DATA_PTR Crl,
const CSSM_OID_PTR CrlField,
CSSM_HANDLE_PTR ResultsHandle,
uint32 *NumberOfMatchedCrls)

This function returns the value of the designated CRL field. If more than one field matches the *CrlField* OID, the first matching field will be returned. The number of matching fields is an output parameter, as is the ResultsHandle to be used to retrieve the remaining matching fields.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

Crl (input)
A pointer to the CSSM_DATA structure that contains the CRL from which the first revocation record is to be retrieved.

CrlField (input)
A pointer to an object identifier that identifies the field value to be extracted from the *Crl*.

ResultsHandle (output)
A pointer to the CSSM_HANDLE, which should be used to obtain any additional matching fields.

NumberOfMatchedFields (output)
The number of fields that match the *CrlField* OID.

Return Value

Returns a pointer to a CSSM_DATA structure containing the first field that matched the *CrlField*. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid Certificate Library Handle
CSSM_CL_INVALID_DATA_POINTER	Invalid pointer input
CSSM_CL_UNKNOWN_FORMAT	Unrecognized CRL format
CSSM_CL_UNKNOWN_TAG	Unknown field tag
CSSM_CL_MEMORY_ERROR	Not enough memory
CSSM_CL_NO_FIELD_VALUES	No field values for this results handle
CSSM_CL_CRL_GET_FIELD_VALUE_FAIL	Unable to get field value

See Also

CL_CrlGetNextFieldValue, CL_CrlAbortQuery

2.4.8 CL_CrlGetNextFieldValue

CSSM_DATA_PTR CSSMCLI CL_CrlGetNextFieldValue (CSSM_CL_HANDLE CLHandle, CSSM_HANDLE ResultsHandle)

This function returns the next CRL field that matched the OID in a call to CL_CrlGetFirstFieldValue.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

ResultsHandle (input)
The handle that identifies the results of a CRL query.

Return Value

Returns a pointer to a CSSM_DATA structure containing the next field in the CRL, which matched the *CrlField* specified in the CL_CrlGetFirstFieldValue function. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid Certificate Library Handle
CSSM_CL_INVALID_RESULTS_HANDLE	Invalid results handle
CSSM_CL_MEMORY_ERROR	Not enough memory
CSSM_CL_NO_FIELD_VALUES	No field values for this results handle
CSSM_CL_CRL_GET_FIELD_VALUE_FAIL	Unable to get field value

See Also

CL_CrlGetFirstFieldValue, CL_CrlAbortQuery

2.4.9 CL_CrlAbortQuery

CSSM_RETURN CSSMCLI CL_CrlAbortQuery (CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE ResultsHandle)

This function terminates the query initiated by CL_CrlGetFirstFieldValue and allows the CL to release all intermediate state information associated with the query.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

ResultsHandle (input)
The handle that identifies the results of a CRL query.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error condition occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid Certificate Library Handle
CSSM_CL_INVALID_RESULTS_HANDLE	Invalid results handle
CSSM_CL_CRL_ABORT_QUERY_FAIL	Unable to abort query

See Also

CL_CrlGetFirtsFieldValue, CL_CrlGetNextFieldValue

2.4.10 CL_CrlDescribeFormat

CSSM_OID_PTR CSSMCLI CL_CrlDescribeFormat (CSSM_CL_HANDLE CLHandle,
uint32 *NumberOfFields)

This function returns a list of the object identifiers used to describe the CRL format supported by the specified CL.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

NumberOfFields (output)
The length of the output array.

Return Value

A pointer to the array of CSSM_OID structures which are supported for CRL operations in the specified CL module. If the return pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid handle
CSSM_CL_INVALID_POINTER	Invalid pointer
CSSM_CL_MEMORY_ERROR	Error allocating memory
CSSM_CL_CRL_DESCRIBE_FORMAT_FAIL	Unable to return the list of fields

See Also

2.5 Extensibility Functions

The CL_PassThrough function is provided to allow CL developers to extend the certificate and CRL format-specific functionality of the CSSM API. Because it is only exposed to CSSM as a function pointer, its name internal to the certificate library can be assigned at the discretion of the CL module developer. However, its parameter list and return value must match what is shown below. The error codes given in this section constitute the generic error codes, which may be used by all certificate libraries to describe common error conditions. Certificate library developers may also define their own module-specific error codes, as described in Section 3.5.2.

2.5.1 CL_PassThrough

CSSM_DATA_PTR CSSMCLI CL_PassThrough (CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
uint32 PassThroughId,
const CSSM_DATA_PTR InputParams)

This function allows applications to call certificate library module-specific operations.

Parameters

CLHandle (input)
The handle that describes the add-in certificate library module used to perform this function.

CCHandle (input)
The handle that describes the context of the cryptographic operation.

PassThroughId (input)
An identifier assigned by the CL module to indicate the function to perform.

InputParams (input)
A pointer to an array of CSSM_DATA structures containing parameters to be interpreted in a function-specific manner by the requested CL module.

Return Value

A pointer to the CSSM_DATA structure containing the output from the pass-through function. The output data must be interpreted by the calling application based on externally available information. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INVALID_CL_HANDLE	Invalid Certificate Library Handle
CSSM_CL_INVALID_CC_HANDLE	Invalid Cryptographic Context Handle
CSSM_CL_INVALID_DATA_POINTER	Invalid pointer input
CSSM_CL_UNSUPPORTED_OPERATION	Add-in does not support this function
CSSM_CL_PASS_THROUGH_FAIL	Unable to perform pass through

2.6 Module Management Functions

The CL_Initialize function is used by the CSSM Core to determine whether the CL module version being attached is compatible with the CL module version being requested and to perform any module-specific setup activities. The CL_Uninitialize function is used to perform any module-specific cleanup activities prior to module detach. Because these functions are only exposed to CSSM as function pointers, their names internal to the certificate library can be assigned at the discretion of the CL module developer. However, their parameter lists and return values must match what is shown below. The error codes given in this section constitute the generic error codes, which may be used by all certificate libraries to describe common error conditions. Certificate library developers may also define their own module-specific error codes, as described in Section 3.5.2.

2.6.1 CL_Initialize

CSSM_RETURN CSSMCLI CL_Initialize (uint32 VerMajor,
uint32 VerMinor)

This function checks whether the current version of the CL module is compatible with the input version and performs any module-specific setup activities.

Parameters

VerMajor (input)
The major version number of the CL module expected by the calling application.

VerMinor (input)
The minor version number of the CL module expected by the calling application.

Return Value

A CSSM_OK return value signifies that the current version of the CL module is compatible with the input version numbers and all setup operations were successfully performed. When CSSM_FAIL is returned, either the current CL module is incompatible with the requested CL module version or an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_INITIALIZE_FAIL	Unable to perform module initialization

See Also

CL_Uninitialize

2.6.3 CL_Uninitialize

CSSM_RETURN CSSMCLI CL_Uninitialize (void)

This function performs any module-specific cleanup activities.

Parameters

None

Return Value

A CSSM_OK return value signifies that all cleanup operations were successfully performed. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CL_UNINITIALIZE_FAIL	Unable to perform module cleanup

See Also

CL_Initialize

3. Certificate Library Structure and Management

3.1 Introduction

A certificate library is an add-in module, which can be used by applications via CSSM to perform syntactic operations on certificates and certificate revocation lists (CRLs). Because all certificate and CRL format-specific information is encapsulated in the certificate library, the application can focus on interesting uses of certificates and CRLs, rather than their format and management. The availability of certificate libraries also allows a CL developer to easily customize and extend their certificate and CRL formats to meet changing market requirements.

This section is provided to clarify key aspects of the structure and management of Certificate Libraries. It covers the composition of a certificate library, CL installation, the expected behavior of a CL on attach, and some basic services expected of CL functions. This section also includes examples of the code structure for several CL functions.

3.2 Certificate Library Composition

A certificate library is a dynamically linkable library, which is composed of functions that implement some or all of the CSSM CLI described in Section 2. The CL must also contain a function or functions that are called when the CL is attached and detached. Within the certificate library, the attach function will be responsible for registering a function table with CSSM, accepting the memory management upcalls, and performing any module-specific setup. The detach function will be responsible for any cleanup required by the module. The remaining functions consist of some subset of the CLI determined by the CL developer.

The certificate library composition can be broadly classified into the following categories:

- Registration with CSSM
- Memory Management
- Certificate Cryptographic Operations
- Certificate Field Management
- Certificate Type Translation
- CRL Cryptographic Operations
- CRL Field Management
- PassThrough Operation Support

3.3 Certificate Library Installation

Before a certificate library can be used by an application, its name, location, and certificate characteristics must be registered with CSSM by an installation application. The name of a certificate library module is given by both a logical name and a globally unique identifier (GUID). The logical name is a string chosen by the certificate library developer to describe the CL module. The GUID is a structure used to differentiate between library modules in the CSSM registry. GUIDs are discussed in more detail below. The location of the CL module is required on installation so that CSSM can locate the module when an application requests an attach. The certificate characteristics are registered with CSSM at install time so that an application can query for CL module availability and features.

3.3.1 Global Unique Identifiers (GUIDs)

Each certificate library must have a globally unique identifier (GUID) which will be used by CSSM, applications, and CL modules to uniquely identify a CL. The CL GUID will be used by the CSSM registry to expose add-in module availability to applications. The application will use the CL GUID to identify a targeted certificate library in all CL function calls. The CL module will use its GUID to identify itself when it sets an error.

A GUID is defined as:

```
typedef struct cssm_guid {
    uint32 Data1;
    uint16 Data2;
    uint16 Data3;
    uint8 Data4[8];
} CSSM_GUID, *CSSM_GUID_PTR;
```

GUID generators are publicly available for Windows* 95, Windows NT*, and on many UNIX* platforms.

3.3.2 Certificate characteristics

Certificate Libraries have certain common characteristics, which should be exposed to applications. These characteristics are registered with CSSM during installation so that they may be queried by applications. The characteristics that should be registered with CSSM include the version of the certificate library, the type of certificate that it recognizes, a template of OIDs to describe its certificate format, and a listing of the certificate types that can be translated into its native format.

The version of a certificate library can be used by applications to determine the compatibility of the installed CL with its required CL. If the compatible versions are unknown to the application, the application can pass the version number that it understands to the CL at attach time. At that time, the CL should check for compatibility and either attach or fail accordingly.

The type of certificate recognized by a certificate library module is identified by a 32 bit unsigned integer. These types are expected to include such well-known certificate formats as X.509 Version 1, X.509 version 3 and SDSI as well as custom certificate types. These same certificate types are used to identify the certificates that can be imported into and exported from the CL module's native certificate format.

Advanced applications may require knowledge of the fields of the certificate. These fields are accessible to applications via object identifiers (OIDs). These object identifiers can then be used by applications to create and perform queries on certificates and CRLs.

3.3.3 Object Identifiers (OIDs)

A certificate library makes its fields known to applications via object identifiers (OIDs). These OIDs are used to reference specific data types or data structures within a given certificate or CRL format. OIDs are defined by the certificate library developer at a granularity appropriate for the expected usage of the CL.

If a certificate format potentially contains more than one instance of a given OID, CL_CertGetFirstFieldValue and CL_CertGetNextFieldValue have been included in the CLI to aid the application in obtaining all instances of the requested OID. For example, the Intel CLM certificate format allows for multiple signers on a single certificate. The application can obtain all of the signatures by a call to CL_CertGetFirstFieldValue followed by multiple calls to CL_CertGetNextFieldValue.

3.4 Attaching a Certificate Library

Before an application can use the functions of a specific CL, it must attach the CL to CSSM using the *CSSM_CL_Attach* function. On attach, the certificate library uses the *CSSM_CL_RegisterServices* function to register its function table with CSSM and to obtain the application's memory management upcalls from CSSM. CSSM will use the CL module's function table to direct calls from the application to the correct function in the certificate library module. The CL module uses the memory management upcalls to allocate any memory that will be returned to the calling application and to free any memory that it received from the calling application.

When CSSM attaches to or detaches from a certificate library module, it initiates a function in the CL that performs the necessary setup and cleanup operations. The attach and detach functions will vary depending on the target operating system for the certificate library module. For example, DllMain would be used to implement these functions in a CL targeted to Windows NT*. _init and _fini would be used to implement these functions in a CL targeted to SunOS*.

3.4.1 The CL module function table

The function table for a certificate library module is a structure that contains pointers to the CL module's implementation of the functions specified in the Certificate Library Interface. This structure is specified as a part of the CSSM header file, cssm.h. If a CL does not support some function in the CLI, the pointer to that function should be set to NULL.

3.4.2 Memory management upcalls

All memory allocation and de-allocation for data passed between the application and the CL module via CSSM is ultimately the responsibility of the calling application. Since the CL module will need to allocate memory in order to return data to the application, the application must provide the CL module a means of allocating memory, which the application has the ability to free. It does this by providing the CL module with memory management upcalls.

Memory management upcalls are simply pointers to the memory management functions used by the calling application. They are provided to the CL module via CSSM as a structure of function pointers. The functions will be the calling application's equivalent of malloc, free, re-alloc and calloc and will be expected to have the same behavior as those functions. The function parameters will consist of a CL handle followed by the normal parameters for that function. The CL handle is used by CSSM to direct the memory operation to the target application. The function return values should be interpreted in the standard manner. The CL module is responsible for making the memory management functions available to all of its internal functions.

3.5 Certificate Library Basic Services

3.5.1 Function Implementation

A certificate library developer may choose to implement some or all of the functions specified in the CLI. The expected behavior of each function is detailed in Section 2 of this document.

A certificate library developer may choose to leverage the capabilities of another CL module to implement certain functions. To do this, the CL would attach to another CL using *CSSM_CL_Attach*. Subsequent function calls to the first CL would call the corresponding function in the second CL for some or all of its implementation.

3.5.2 Error handling

When an error occurs, the function in the CL module should call the *CSSM_SetError* function. The *CSSM_SetError* function takes the module's GUID and an error number as inputs. The module's GUID will be used to identify where the error occurred. The error number will be used to describe the error.

The error number set by the CL module should fall into one of two ranges. The first range of error numbers is pre-defined by CSSM. These are errors, which are expected to be common to all CL modules implementing a given function. They are described in this document as part of the function definitions in Sections 2.3, 2.4, and 2.5. They are defined in the header file cssmerr.h, which is distributed as part of CSSM. The second range of error numbers is used to define module-specific error codes. These module-specific error codes should be in the range of CSSM_CL_PRIVATE_ERROR to CSSM_CL_END_ERROR. CSSM_CL_PRIVATE_ERROR and CSSM_CL_END_ERROR are also defined in the header file cssmerr.h. The CL module developer is responsible for making the definition and interpretation of their module-specific error codes available to applications.

When no error has occurred, but the appropriate return value from a function is CSSM_FALSE, that function should call *CSSM_ClearError* before returning. When the application receives a CSSM_FALSE return value, it is responsible for checking whether an error has occurred by calling *CSSM_GetError*. If the function in the CL module has called *CSSM_ClearError*, the calling application will receive CSSM_OK response from the *CSSM_GetError* function, indicating that no error has occurred.

3.6 Certificate Utility Libraries

Certificate utility libraries are software components that may be provided by a certificate library developer for use by other certificate library developers. They are expected to contain functions that may be useful to several certificate library modules, such as BER and DER encoding and decoding. The certificate utility library developer is responsible for making the definition, interpretation, and usage of their library available to other CL module developers.

3.7 Attach/Detach Example

The certificate library module is responsible for performing certain operations when CSSM attaches to and detaches from it. CL modules that have been developed for Windows-based systems will use the DllMain routine to perform those operations, as shown in the example below.

3.7.1 DllMain

```
CSSM_GUID intel_clm_guid =
{ 0x83bafc39, 0xfac1, 0x11cf, { 0x81, 0x72, 0x0, 0xaa, 0x0, 0xb1, 0x99, 0xdd }
};

BOOL WINAPI DllMain ( HANDLE hInstance, DWORD dwReason, LPVOID lpReserved)
{
   switch (dwReason)
   {
      case DLL_PROCESS_ATTACH:
      {
         CSSM_SPI_CL_FUNCS FunctionTable;
         CSSM_SPI_FUNC_TBL_PTR UpcallTable;

         /* Fill in FunctionTable with function pointers */
         FunctionTable.CertSign                  = CL_CertSign;
         FunctionTable.CertUnsign                = CL_CertUnsign;
         FunctionTable.CertVerify                = CL_CertVerify;
         FunctionTable.CertCreate                = CL_CertCreate;
         FunctionTable.CertView                  = CL_CertView;
         FunctionTable.CertGetFirstFieldValue    = CL_CertGetFirstFieldValue;
         FunctionTable.CertGetNextFieldValue     = CL_CertGetNextFieldValue;
         FunctionTable.CertAbortQuery            = CL_CertAbortQuery;
         FunctionTable.CertGetKeyInfo            = CL_CertGetKeyInfo;
         FunctionTable.CertGetAllFields          = CL_CertGetAllFields;
         FunctionTable.CertImport                = NULL;
         FunctionTable.CertExport                = NULL;
         FunctionTable.CertDescribeFormat        = CL_CertDescribeFormat;

         FunctionTable.CrlCreate                 = CL_CrlCreate;
         FunctionTable.CrlAddCert                = CL_CrlAddCert;
         FunctionTable.CrlRemoveCert             = CL_CrlRemoveCert;
         FunctionTable.CrlSign                   = CL_CrlSign;
         FunctionTable.CrlVerify                 = CL_CrlVerify;
         FunctionTable.IsCertInCrl               = CL_IsCertInCrl;
         FunctionTable.CrlGetFirstFieldValue     = CL_CrlGetFirstFieldValue;
         FunctionTable.CrlGetNextFieldValue      = CL_CrlGetNextFieldValue;
         FunctionTable.CrlAbortQuery             = CL_CrlAbortQuery;
         FunctionTable.CrlDescribeFormat         = CL_CrlDescribeFormat;

         FunctionTable.PassThrough               = CL_PassThrough;
         FunctionTable.Initialize                = CL_Initialize;
         FunctionTable.Uninitialize              = CL_Uninitialize;

         /* Call CSSM_CL_RegisterServices to register the FunctionTable */
         /* with CSSM and to receive the application's memory upcall table */
         if (CSSM_CL_RegisterServices (&intel_clm_guid, FunctionTable,
         &UpcallTable) != CSSM_OK)
            return FALSE;
```

```
            /* Make the upcall table available to all functions in this library
            */

            break;
         }
      case DLL_THREAD_ATTACH:
         break;
      case DLL_THREAD_DETACH:
         break;
      case DLL_PROCESS_DETACH:
         if (CSSM_CL_DeregisterServices (&intel_clm_guid) != CSSM_OK)
            return FALSE;
         break;
      }
return TRUE;
}
```

3.8 Certificate Operations Examples

This section contains sample implementations of certificate functions in the certificate library.

3.8.1 CL_CertCreate

```
/*-------------------------------------------------------------------------
-
 * Name: CL_CertCreate
 *
 * Description:
 * This function allocates and initializes memory for a certificate
 * based on the input tag/values pairs. The returned certificate
 * must be signed using the CSSM_CL_CertSign function.
 *
 * Parameters:
 * CertTemplate (input)   : A pointer to an array of tag/value pairs
 *                          which identify the fields of the new certificate
 * NumberOfFields (input) : The length of the CertTemplate array
 *
 * Return value:
 * The new certificate
 *
 * Error Codes:
 * CSSM_CL_INVALID_CL_HANDLE
 * CSSM_CL_INVALID_FIELD_POINTER
 * CSSM_CL_INVALID_TEMPLATE
 * CSSM_CL_MEMORY_ERROR
 * CSSM_CL_UNSUPPORTED_OPERATION
 * CSSM_CL_CERT_CREATE_FAIL
*-------------------------------------------------------------------------*/
CSSM_DATA_PTR   CSSMCLI CL_CertCreate   (CSSM_CL_HANDLE CLHandle,
                                         const CSSM_FIELD_PTR CertTemplate,
                                         uint32 NumberOfFields)
{
    /* Initializations */
    CSSM_CERTIFICATE_PTR cert_ptr = NULL;
    CSSM_DATA_PTR  packed_cert_ptr = NULL;
    CSSM_ERROR_PTR err_ptr = NULL;
    uint32 i=0;

    /* Check inputs */
       /* Check that this is a valid CLHandle */
    if (CLHandle == 0)
    {
        CSSM_SetError(&intel_clm_guid, CSSM_CL_INVALID_CL_HANDLE);
        return NULL;
    }
        /* Check that the NumberOfFields is greater than 0
           and that the CertTemplate pointer is not NULL */
    if ( !NumberOfFields || !CertTemplate)
    {
        CSSM_SetError(&intel_clm_guid, CSSM_CL_INVALID_TEMPLATE);
        return NULL;
    }
        /* Check that CertTemplate is a valid pointer */
    if (cssm_IsBadReadPtr (CertTemplate, NumberOfFields*sizeof(CSSM_FIELD)) ||
        cssm_IsBadReadPtr(CertTemplate[NumberOfFields-1].FieldValue.Data,
            CertTemplate[NumberOfFields-1].FieldValue.Length) ||
```

```
        cssm_IsBadReadPtr(CertTemplate[NumberOfFields-1].FieldOid.Data,
            CertTemplate[NumberOfFields-1].FieldOid.Length) )
    {
        CSSM_SetError(&intel_clm_guid, CSSM_CL_INVALID_TEMPLATE);
        return NULL;
    }

    /* Allocate a new certificate structure */
    cert_ptr = UpcallTable.malloc_func(CLHandle, sizeof(CSSM_CERTIFICATE));
    if (cert_ptr == NULL)
    {
        CSSM_SetError(&intel_cl_guid, CSSM_CL_MEMORY_ERROR);
        return NULL;
    }
    memset(cert_ptr, 0, sizeof(CSSM_CERTIFICATE));

    /* Loop through the CertTemplate array */
    for( i=0; i < NumberOfFields; i++ )
    {
        /* Check that this field contains a valid data pointer */
        if ( !cl_IsBadReadPtr (CertTemplate[i].FieldValue.Data,
                            CertTemplate[i].FieldValue.Length))
        {
            /* If so, copy the data into the certificate structure */
                    /* Add CL module-specific code here */
        }
        else
        {
            CSSM_SetError(&intel_cl_guid, CSSM_CL_INVALID_FIELD_POINTER);
            /* Free the certificate structure */
            return NULL;
        }
    }

    /* Add internal, CL-generated certificate information */
        /* Add CL module-specific code here */

    /* If there are signatures on this cert, delete them */
    /* A newly created cert is assumed to be unsigned */
        /* Add CL module-specific code here */

    /* Pack the new certificate */
    /* The pack routine will allocate memory for the new cert using the
       application's memory allocation routines */
    packed_cert_ptr = cl_PackCertificate(cert_ptr);

    /* Cleanup */
        /* Free the certificate structure */

    /* Return the packed certificate */
    return packed_cert_ptr;
};
```

3.9 CRL Operations Examples

This section contains sample implementations of certificate revocation list functions in the certificate library.

3.9.1 CL_CrlAddCert

```
/*-----------------------------------------------------------------------------
* Name: CL_CrlAddCert
 *
 * Description:
 * This function revokes the input certificate by adding a record representing
 * the certificate to the CRL. It uses the revoker's certificate to sign the
new
 * record in the CRL.  The reason for revoking the certificate may also be
stored
 * in the revocation record.
 *
 * Parameters:
 * Cert (input)           : A pointer to the CSSM_DATA structure containing
the
 *                          certificate to be revoked
 * RevokerCert (input)    : A pointer to the CSSM_DATA structure containing
the
 *                          revoker's certificate
 * RevokeReason (input)   : The reason for revoking the certificate
 * OldCrl (input)         : A pointer to the CSSM_DATA structure containing
the
 *                          CRL to which the newly revoked certificate will be
 *                          added
 *
 * Return value:
 * The updated CRL
 *
 * Error Codes:
 * CSSM_CL_INVALID_CL_HANDLE
 * CSSM_CL_INVALID_CC_HANDLE
 * CSSM_CL_INVALID_CERTIFICATE_PTR
 * CSSM_CL_INVALID_CRL
 * CSSM_CL_MEMORY_ERROR
 * CSSM_CL_CRL_ADD_CERT_FAIL
*-----------------------------------------------------------------------------*/
CSSM_DATA_PTR   CSSMCLI CL_CrlAddCert    (CSSM_CL_HANDLE CLHandle,
                                          CSSM_CC_HANDLE CCHandle,
                                          const CSSM_DATA_PTR Cert,
                                          const CSSM_DATA_PTR RevokerCert,
                                         CSSM_REVOKE_REASON RevokeReason,
                                          const CSSM_DATA_PTR OldCrl)
{
       CSSM_REVOCATION_LIST_PTR new_crl_ptr = NULL;
       CSSM_DATA_PTR new_crl_data_ptr = NULL;
       CSSM_DATA_PTR sign_data_ptr = NULL;
       CSSM_REVOKED_CERT_PTR new_revoked_cert_ptr = NULL;
       CSSM_REVOKED_CERT_PTR temp_revoked_cert_ptr = NULL;
       CSSM_REVOKED_CERT_PTR prev_revoked_cert_ptr = NULL;

       CSSM_CERTIFICATE_PTR revoker_cert_ptr = NULL;
       CSSM_CERTIFICATE_PTR cert_ptr = NULL;
```

```
	uint32 signature_size;
	CSSM_DATA_PTR signature_data_ptr = NULL;
	CSSM_CONTEXT_PTR context_ptr = NULL;
	CSSM_RETURN ret;

	/* Check inputs */
	if(CLHandle == 0)
	{
		CSSM_SetError(&intel_clm_guid,CSSM_CL_INVALID_CL_HANDLE);
		return NULL;
	}
	if(CCHandle == 0)
	{
		CSSM_SetError(&intel_clm_guid,CSSM_CL_INVALID_CC_HANDLE);
		return NULL;
	}
	if(Cert == NULL)
	{
		CSSM_SetError(&intel_clm_guid,CSSM_CL_INVALID_CERT_POINTER);
		return NULL;
	}
	if(Cert != NULL && cssm_IsBadReadPtr(Cert, sizeof(CSSM_DATA)) )
	{
		CSSM_SetError(&intel_clm_guid, CSSM_CL_INVALID_DATA_POINTER);
		return NULL;
	}
	if(Cert->Length != 0 && cssm_IsBadReadPtr(Cert->Data,Cert->Length))
	{
		CSSM_SetError(&intel_clm_guid, CSSM_CL_INVALID_CERT_POINTER);
		return NULL;
	}

	if(RevokerCert == NULL)
	{
		CSSM_SetError(&intel_clm_guid,CSSM_CL_INVALID_REVOKER_CERT_PTR);
		return NULL;
	}
	if(RevokerCert->Length != 0 && cssm_IsBadReadPtr(RevokerCert->Data,RevokerCert-
>Length))
	{
		CSSM_SetError(&intel_clm_guid, CSSM_CL_INVALID_REVOKER_CERT_PTR);
		return NULL;
	}
	if(OldCrl == NULL)
	{
		CSSM_SetError(&intel_clm_guid, CSSM_CL_INVALID_CRL_PTR);
		return NULL;
	}
	if(cssm_IsBadReadPtr(OldCrl, sizeof(CSSM_DATA)))
	{
		CSSM_SetError(&intel_clm_guid, CSSM_CL_INVALID_CRL_PTR);
		return NULL;
	}
	if(OldCrl->Length != 0 && !cssm_IsBadReadPtr(OldCrl->Data, OldCrl-
>Length))
	{
		/* Unpack the CRL */
		new_crl_ptr = cl_UnPackCrl(CLHandle,&MemoryFunctions,OldCrl);
		if(new_crl_ptr == NULL)
```

```
                {
                        CSSM_SetError(&intel_clm_guid, CSSM_CL_MEMORY_ERROR);
                        return NULL;
                }

                /* remove the crl signature, if necessary */
                /* unpack the revoker's certificate */
                revoker_cert_ptr =
cl_UnpackCertificate(CLHandle,&MemoryFunctions,RevokerCert);
                if(revoker_cert_ptr == NULL)
                {
                        /* Cleanup */
                        CSSM_SetError(&intel_clm_guid, CSSM_CL_MEMORY_ERROR);
                        return NULL;
                }
                /* unpack the certificate to be revoked */
                cert_ptr = cl_UnpackCertificate(CLHandle,&MemoryFunctions,Cert);;
                if(cert_ptr == NULL)
                {
                        /* Cleanup */
                        CSSM_SetError(&intel_clm_guid, CSSM_CL_MEMORY_ERROR);
                        return NULL;
                }

                /* Create the revoked certificate structure to be placed in the CRL
        */
                /* Add any revocation record specific information,
                    such as the time of revocation and the revocation reason */
                /* Sign the revoked certificate structure using the revoker's certificate */
        }

        /* Add the new revocation record to the CRL */

        /* Pack the new CRL */
        new_crl_data_ptr = cl_PackCrl(CLHandle,&MemoryFunctions,new_crl_ptr);

        /* Cleanup & Return */
        return new_crl_data_ptr;
}
```

3.10 Extensibility Functions Examples

This section contains a sample implementation of the pass-through function in the certificate library.

3.10.1 CL_PassThrough

In this example, the pack and unpack routines that are used internally to the CL module are exposed for use by applications via the pass-through mechanism.

```
typedef enum cl_custom_function_id {
    CL_CUSTOMID_PACK_CERTIFICATE = 0,
    CL_CUSTOMID_UNPACK_CERTIFICATE     = 1,
} CL_CUSTOM_FUNCTION_ID;

/*---------------------------------------------------------------------------
-
 * Name: CL_PassThrough
 *
 * Description:
 * This function allows applications to call CSSM CL module-specific
operations.
 * The CSSM CL module-specific operations include:
 *    cl_PackCertificate
 *    cl_UnpackCertificate
 *
 * Parameters:
 * CCHandle (input)       : Handle identifying a Cryptographic Context which
 *                          may be used by the pass-through function
 * PassThroughId (input)  : An identifier assigned by the CSSM CL module
 *                          to indicate the exported function to perform.
 * InputParams (input)    : Parameters to be interpreted in a
 *                          function-specific manner by the CSSM CL module.
 *
 * Return value:
 * Output from the pass-through function.
 * The output data must be interpreted by the calling application
 * based on externally available information.
 *
 * Error Codes:
 * CSSM_CL_INVALID_CL_HANDLE
 * CSSM_CL_INVALID_CC_HANDLE
 * CSSM_CL_INVALID_DATA_POINTER
 * CSSM_CL_UNSUPPORTED_OPERATION
 * CSSM_CL_PASS_THROUGH_FAIL
 *---------------------------------------------------------------------------
*/
CSSM_DATA_PTR CSSMCLI CL_PassThrough   (CSSM_CL_HANDLE CLHandle,
                                        CSSM_CC_HANDLE CCHandle,
                                        uint32 PassThroughId,
                                        const CSSM_DATA_PTR InputParams)
{
    /* Initializations */
    /* Check inputs */
        /* Check that this is a recognized PassThroughId */

    /* Call the requested function */
    switch ( PassThroughId ) {
    case CL_CUSTOMID_PACK_CERTIFICATE:
```

```
      return cl_PackCertificate( InputParams );
    case CL_CUSTOMID_UNPACK_CERTIFICATE:
      return cl_UnpackCertificate( InputParams );
    default:
       CSSM_SetError(&intel_cl_guid, CSSM_CL_UNSUPPORTED_OPERATION);
       return NULL;
    }
};
```

4. Appendix A. Relevant CSSM API functions

4.1 Overview

There are several API functions that will be particularly relevant to certificate library developers, either because they are used by the application to access the CL module or because they are used by the CL module to access CSSM services, such as the CSSM registry or the error-handling routines. They have been included in this appendix for quick-reference by CL module developers. For additional information, the CL module developer is encouraged to reference the *CSSM Application Programming Interface*.

4.2 Data Structures

4.2.1 CSSM_CERT_TYPE

This variable specifies the type of certificate supported by a certificate library and the types of certificate types understood for import and export. They are expected to define such well-known certificate formats as X.509 Version 3 and SDSI as well as custom certificate formats.

```
typedef uint32 CSSM_CERT_TYPE, *CSSM_CERT_TYPE_PTR
```

4.2.2 CSSM_DATA

The CSSM_DATA structure is used to associate a length, in bytes, with an arbitrary block of contiguous memory. This memory must be allocated and freed using the memory management routines provided by the calling application via CSSM.

```
typedef struct cssm_data {
    uint32 Length;
    uint8* Data;
} CSSM_DATA, *CSSM_DATA_PTR
```

Definition:

Length - The length, in bytes, of the memory block pointed to by *Data*

Data - A pointer to a contiguous block of memory.

4.2.3 CSSM_OID

The object identifier (OID) is used to identify the data types and data structures of a certificate or CRL.

```
typedef CSSM_DATA CSSM_OID
```

4.2.4 CSSM_GUID

A GUID is a globally unique identifier, which is used to uniquely identify a CL.

```
typedef struct cssm_guid {
    uint32 Data1;
    uint16 Data2;
    uint16 Data3;
    uint8 Data4[8];
} CSSM_GUID, *CSSM_GUID_PTR;
```

4.2.5 CSSM_CLINFO

Certificate Libraries have certain common characteristics, which should be exposed to applications. These characteristics are given by the CSSM_CLINFO structure, which is registered with CSSM during installation so that they may be queried by applications.

```
typedef struct cssm_clinfo{
    CSSM_CERT_TYPE CertType;
    uint32 NumberOfFields;
    CSSM_OID_PTR CertTemplate;
    uint32 VerMajor;
    uint32 VerMinor;
    uint32 NumberOfTypes;
    CSSM_CERT_TYPE_PTR CertTranslationType;
}CSSM_CLINFO, *CSSM_CLINFO_PTR
```

Definition:

CertType - An identifier for the type of certificate format supported by the CL.

NumberOfFields - The number of certificate object identifiers. This number also indicates the length of the *CertTemplate* array.

CertTemplate - A pointer to an array of object identifiers (OIDs) which identify the tags of the supported certificate format.

VerMajor - The major version number of the add-in module.

VerMinor - The minor version number of the add-in module.

NumberOfTypes - The number of certificate types that this certificate library add-in module can import and export. This number also indicates the length of the *CertTranslationType* array.

CertTranslationType - A pointer to an array of certificate types. This array indicates the certificate types that can be imported into and exported from this certificate library module's native certificate type.

4.2.6 CSSM_SPI_FUNC_TBL

This data structure contains function pointers to the calling application's memory management routines. These routines will be used by the CL module to allocate and free any memory that belongs to or will belong to the application.

```
typedef struct cssm_spi_func_tbl {
    void *(*malloc_func) (CSSM_HANDLE, uint32);
    void (*free_func) (CSSM_HANDLE, void *);
    void  *(*realloc_func) (CSSM_HANDLE, void *, uint32);
    void  *(*calloc_func) (CSSM_HANDLE, uint32 num, uint32 size );
} CSSM_SPI_MEMORY_FUNCS, *CSSM_SPI_MEMORY_FUNCS_PTR;
```

4.3 Function Definitions

4.3.1 CSSM_CL_Install

CSSM_BOOL CSSMAPI CSSM_CL_Install (const char *CLName,
const char *CLFileName,
const char *CLPathName,
const CSSM_GUID_PTR GUID,
const CSSM_CLINFO_PTR CLInfo,
const void * Reserved1,
const CSSM_DATA_PTR Reserved2)

This function updates the CSSM persistent internal information about the CL module.

Parameters

CLName (input)
The name of the certificate library module.

CLFileName (input)
The name of file that implements the certificate library.

CLPathName (input)
The path to the file that implements the certificate library.

GUID (input)
A pointer to the CSSM_GUID structure containing the global unique identifier for the CL module.

CLInfo (input)
A pointer to the CSSM_CLINFO structure containing information about the CL module.

Reserved1 (input)
Reserve data for the function.

Reserved2 (input)
Reserve data for the function.

Return Value

A CSSM_TRUE return value signifies that information has been updated. When CSSM_FALSE is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_DATA_POINTER	Invalid pointer
CSSM_INVALID_CLINFO_POINTER	Invalid pointer
CSSM_INVALID_POINTER	Invalid pointer
CSSM_INSTALL_FAIL	Unable to update internal information

See Also

CSSM_CL_Uninstall

4.3.2 CSSM_CL_Uninstall

CSSM_BOOL CSSMAPI CSSM_CL_Uninstall (const CSSM_GUID_PTR GUID)

This function deletes the persistent CSSM internal information about the CL module.

Parameters

GUID (input)
A pointer to the CSSM_GUID structure containing the global unique identifier for the CL module.

Return Value

A CSSM_TRUE return value signifies that information has been deleted. When CSSM_FALSE is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_DATA_POINTER	Invalid pointer
CSSM_INVALID_ GUID	Certificate Library was not installed
CSSM_UNINSTALL_FAIL	Unable to delete information

See Also

CSSM_CL_Install

4.3.3 CSSM_CL_Attach

CSSM_CL_HANDLE CSSMAPI CSSM_CL_Attach (const CSSM_GUID_PTR GUID,
uint32 CheckCompatibleVerMajor,
uint32 CheckCompatibleVerMinor,
const CSSM_API_FUNC_TBL_PTR MemoryFuncs,
const void * Reserved)

This function attaches the application with the CL module. The CL module tests for compatibility with the version specified.

Parameters

GUID (input)
A pointer to the CSSM_GUID structure containing the global unique identifier for the CL module.

CheckCompatibleVerMajor (input)
The major version number of the CL module that the application is compatible with.

CheckCompatibleVerMinor (input)
The minor version number of the CL module that the application is compatible with.

MemoryFuncs (input)
A pointer to the structure containing the application's memory management function pointers.

Reserved (input)
A reserved input.

Return Value

A handle is returned for the CL module. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_DATA_POINTER	Invalid pointer
CSSM_INVALID_ GUID	Invalid CL GUID
CSSM_INCOMPATIBLE_VERSION	Incompatible version
CSSM_ATTACH_FAIL	Unable to attach to CL module

See Also

CSSM_CL_Detach

4.3.4 CSSM_CL_Detach

CSSM_BOOL CSSMAPI CSSM_CL_Detach (CSSM_CL_HANDLE CLHandle)

This function detaches the application from the CL module.

Parameters

CLHandle (input)
The handle that describes the CL module.

Return Value

A CSSM_TRUE return value signifies that the application has been detached from the CL module. When CSSM_FALSE is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_CL_HANDLE	Invalid handle
CSSM_DETACH_FAIL	Unable to detach from CL module

See Also

CSSM_CL_Attach

4.3.5 CSSM_CL_RegisterServices

CSSM_RETURN CSSMAPI CSSM_CL_RegisterServices (const CSSM_GUID_PTR GUID,
CSSM_SPI_CL_FUNCS_PTR FunctionTable,
CSSM_SPI_MEMORY_FUNCS_PTR *UpcallTable,
void *Reserved)

This function is used by a certificate library module to register its function table with CSSM and to receive a memory management upcall table from CSSM.

Parameters

GUID (input)
A pointer to the CSSM_GUID structure containing the global unique identifier for the CL module.

FunctionTable (input)
A structure containing pointers to the Certificate Library Interface functions implemented by the CL module.

UpcallTable (output)
A structure containing pointers to the memory routines to be used by the CL module to allocate and free memory owned by the calling application.

Reserved (input)
A reserved input.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error condition occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_GUID	Invalid GUID
CSSM_INVALID_FUNCTION_TABLE	Invalid function table
CSSM_REGISTER_SERVICES_FAIL	Unable to register services

See Also

CSSM_CL_DeregisterServices

4.3.6 CSSM_CL_DeregisterServices

CSSM_RETURN CSSMAPI CSSM_TP_DeregisterServices (const CSSM_GUID_PTR GUID)

This function is used by a certificate library module to deregister from the CSSM registry.

Parameters

GUID (input)
A pointer to the CSSM_GUID structure containing the global unique identifier for the CL module.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error condition occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_GUID	Invalid GUID
CSSM_DEREGISTER_SERVICES_FAIL	Unable to deregister services

See Also

CSSM_CL_RegisterServices

4.3.7 CSSM_GetError

CSSM_ERROR_PTR CSSMAPI CSSM_GetError (void)

This function returns the current error information.

Parameters

None

Return Value

Returns the current error information. If there is currently no valid error, the error number will be CSSM_OK. A NULL pointer indicates that the CSSM_InitError was not called by the CSSM Core or that a call to CSSM_DestroyError has been made by the CSSM Core. No error information is available.

See Also

CSSM_ClearError, CSSM_SetError

4.3.8 CSSM_SetError

CSSM_RETURN CSSMAPI CSSM_SetError (CSSM_GUID_PTR guid,
uint32 error_number)

This function sets the current error information to *error_number* and *guid*.

Parameters

guid (input)
Pointer to the GUID (global unique ID) of the add-in module.

error_number (input)
An error number. It should fall within one of the valid CSSM, CL, TP, DL, or CSP error ranges.

Return Value

CSSM_OK if error was successfully set. A return value of CSSM_FAIL indicates that the error number passed is not within a valid range, the GUID passed is invalid, CSSM_InitError was not called by the CSSM Core, or CSSM_DestroyError has been called by the CSSM Core. No error information is available.

See Also

CSSM_ClearError, CSSM_GetError

4.3.9 CSSM_ClearError

void CSSMAPI CSSM_ClearError (void)

This function sets the current error value to CSSM_OK. This can be called if the current error value has been handled and therefore is no longer a valid error.

Parameters

None

See Also

CSSM_SetError, CSSM_GetError