Common Security Services Manager

Application Programming Interface (API)

Draft for Release 1.2
March 1997

Subject to Change Without Notice

Specification Disclaimer and Limited Use License

This specification is for release version 1.2, February 1997, updated March 1997.

You are licensed under Intel's copyrights in the CDSA Specifications to download the specifications and to develop, distribute and/or use a conformant software implementation of the specifications. A software implementation of the CDSA Specifications can be tested for conformance via use of the CDSA Conformance Test Suite that accompanies the specifications, and you are licensed to use the conformance test suite for that purpose.

ALL INFORMATION AND OTHER MATERIALS TO BE PROVIDED BY INTEL HEREUNDER ARE PROVIDED "AS IS," AND INTEL MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AND EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS, AND FITNESS FOR A PARTICULAR PURPOSE.

Intel grants no other license under any of its intellectual property other than as expressly granted above. If you desire any broader rights under Intel intellectual property, please contact Intel directly.

.

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

Table of Contents

List of Tables

List of Figures

1. Introduction

This section provides:

- An overview of the Common Data Security Architecture
- An overview of the Common Security Services Manager Application Programming Interface document
- An overview of the Common Data Security Architecture documentation
- References

1.1 Common Data Security Architecture

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, which 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, which 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.

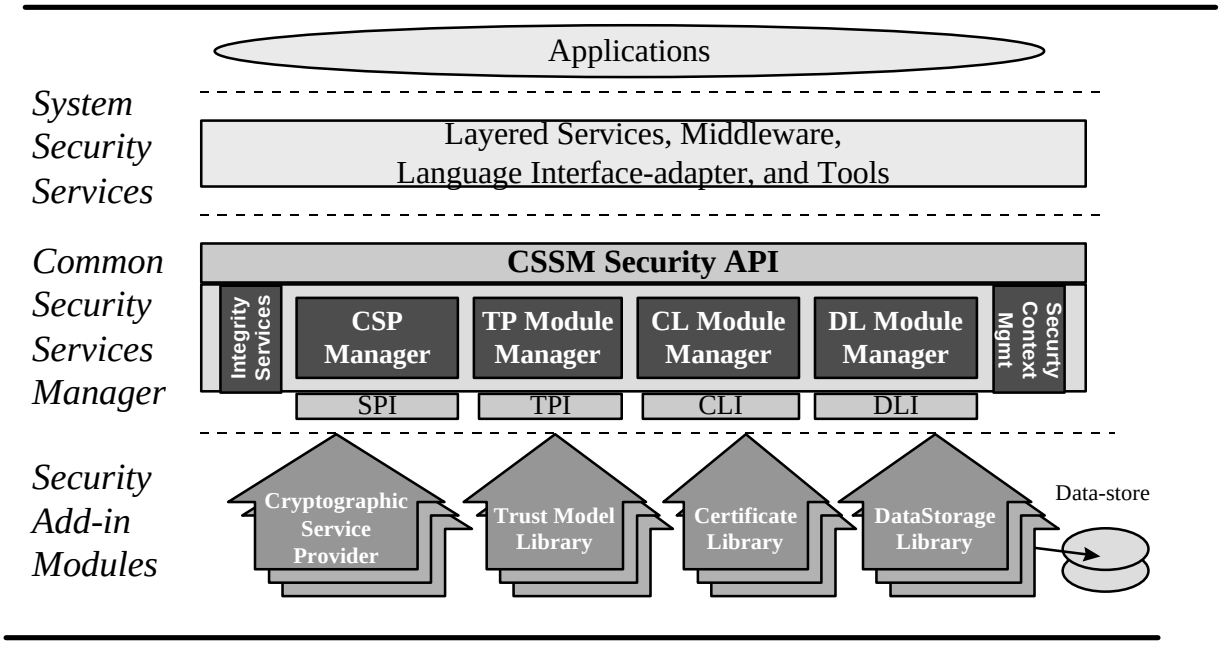

Figure 1. **The Common Data Security Architecture for all platforms.**

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 module installation, 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 CSSM API Document

1.2.1 Intended Audience

This document is intended for use by Independent Software Vendors (ISVs) who will develop their own application code to interact with CSSM services. These ISVs will be highly experienced software and security architects, advanced programmers, and sophisticated users. They are familiar with network operating systems and high-end cryptography. We assume that this audience is familiar with the basic capabilities and features of the protocols they are considering.

1.2.2 Document Organization

This document is divided into the following sections:

Section 2, Core Services API, describes functions that relate to the CSSM core. It also describes data structures and functions that are common to all types of add-in modules.

Section 3, Cryptographic Services API, describes functions that perform encryption, digital signature digests, signature generation and validation. These functions access the cryptographic service providers (tokens) within the context of a cryptographic session. This section also describes functions that are used to manage CSP modules and which provide access to module-specific functionality.

Section 4, Trust Policy Services API, describes functions that can be used for determining a level of trust before performing a syntactic operation on a certificate. For example, the trust policy may determine whether or not a given certificate is authorized to sign other certificates. This section also describes functions which are used to manage TP modules and which provide access to module-specific functionality, such as verification of a certificate's authority to perform module-specific operations.

Section 5, Certificate Library Services API, describes functions that perform syntactic, format-specific operations on certificates and certificate revocation lists (CRLs). The certificate library module performs data format-specific operations, such as creating a new certificate from a list of tag-value pairs. This section also describes functions which are used to manage CL modules, and which provide access to module-specific functionality.

Section 6, Data Storage Library Services API, describes functions that allow access to databases within data storage modules which are used for the persistent storage of certificates and certificate revocation lists. The mechanism used for persistence is assumed to be transparent to the calling application. This section also describes functions which are used to manage DL modules and which provide access to module-specific functionality.

Appendix A, Error-Handling describes the error handling functions and the error return codes used by CSSM.

Appendix B, Application Memory Functions, describes memory management in CSSM as it relates to applications.

Appendix C, Acronyms, a list of acronyms and their definitions. For a more complete glossary, see the *CDSA Specification*.

Sections 2 through 6 are each organized into the following sub-sections:

1. A section overview which describes important implementation details and which highlights each API call.
2. A description of the C data structures used by the functions in that section.
3. A description of each function's purpose, input parameters, output parameters, return value, and applicable error codes.

1.3 CDSA Documentation

A set of documents describing CDSA and CSSM are envisioned. The CDSA Specification and CSSM API Specification are completed and available to the industry for feedback. The other documents are under development. The list of envisioned documents includes:

- *Common Data Security Architecture Specification* (or *CDSA Specification*). This presents the overall CDSA architecture, including CSSM.
- *CSSM Application Programming Interface* (this document, the *CSSM API*). This defines the interface that applications developers use to access CSSM and add-in module services.
- *CSSM Cryptographic Service Provider Interface Specification* (or *CSSM SPI*). This defines the interface that cryptographic service providers must conform to in order to be accessible via CSSM. Individuals interested in making cryptographic services available under the CSSM interface will need to be familiar with the CSSMSPI. This document also provides key information regarding the expected behavior of a cryptographic service provider as well as detailed implementation examples, which may be of use to the cryptographic service provider developer.
- *CSSM Trust Policy Interface Specification* (or *CSSM TPI*). This defines the interface that trust policy modules must conform to in order to be accessible via CSSM. Individuals interested in making trust policy features available under the CSSM interface will need to be familiar with the CSSMTPI. This document also provides key information regarding the expected behavior of a trust policy module as well as detailed implementation examples which may be of use to the trust policy module developer.
- *CSSM Certificate Library Interface Specification* (or *CSSM CLI*). This defines the interface that certificate libraries must conform to in order to be accessible via CSSM. Individuals interested in making certificate library features available under the CSSM interface will need to be familiar with the CSSM CLI. This document also provides key information regarding the expected behavior of a certificate library module, as well as detailed implementation examples which may be of use to the certificate library module developer.

- *CSSM Data Storage Library Interface Specification* (or *CSSM DLI*). This defines the interface that a data storage library must conform to in order to be accessible via CSSM. Individuals interested in making data storage library features available under the CSSM interface will need to be familiar with the CSSM DLI. This document also provides key information regarding the expected behavior of a data storage library module, as well as, detailed implementation examples which may be of use to the data storage library module developer.
- *CSSM-Java* Application Programming Interface* (or *CSSM-Java).* This defines a Java package of classes and methods that Java applets and Java applications must use to access CSSM managed security services.

1.4 References

BSAFE*	*BSAFE Cryptographic Toolkit*, RSA Data Security, Inc., Redwood City, CA: RSA Laboratories
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)
CDSA Specification	*Common Data Security Architecture Specification*, 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. Core Services API

2.1 Overview

The CSSM provides a set of core services for version management, component verification and memory management. These services are supplied by the CSSM and are not handled by add-in modules.

The CSSM management functions allow applications to query for information about the CSSM and to verify components associated with CSSM. A query of CSSM will return information about the version of the CSSM that is running. A function is also provided to verify whether the application's expected CSSM version is compatible with the currently-running CSSM version.

The components verification function allows applications to check the integrity of the system components listed in the signed bill-of-materials file. *All applications should call this routine once, at start-up.* Applications can (and should) use this to verify system integrity before performing a vital operation; a failure return code indicates that system integrity may have been compromised.

To protect against assaults on CSSM and its components, any system binary or data file can be authenticated. There are two levels of inclusion when securing CSSM and its components. The first level consists of signing CSSM itself. During the creation of CSSM, a digital signature and a public key are embedded into the binaries of CSSM. CSSM provides the CSSM_VerifyComponents function to authenticate this signature.

The second level consists of CSSM signing additional components, such as add-in modules. For example, as part of installation, the user generates keys which are used to sign the default encryption module and its adaptation layer.

The CSSM memory management functions are a class of routines for reclaiming memory allocated for the base CSSM objects. The CSSM and the add-in modules are responsible for allocating and freeing these memory objects. However, because add-in modules cannot determine when memory space can be reclaimed, these API calls have been provided for the application to indicate when the memory objects are no longer needed.

2.1.1 CSSM Management Functions

CSSM_RETURN CSSMAPI CSSM_Init accepts as input the CSSM's major and minor version numbers required for compatibility with the calling application.

CSSM_INFO_PTR CSSMAPI CSSM_GetInfo CSSM returns its major and minor version numbers.

CSSM_RETURN CSSMAPI CSSM_FreeInfo accepts as input the pointer to the data structure returned in the CSSM_GetInfo function. The memory allocated by the CSSM is reclaimed by the operating system.

CSSM_RETURN CSSMAPI CSSM_VerifyComponents no input is needed for this function. CSSM verifies the components it has signed. Changes in those components will be detected by the verification process.

2.1.2 CSSM Memory Management Functions

CSSM_RETURN CSSMAPI CSSM_FreeList accepts as input a pointer to a CSSM_LIST memory object allocated by the CSSM. This function reclaims memory by the operating system.

void CSSMAPI CSSM_Free accepts as input a pointer to generic memory allocated by the add-in. This function reclaims memory by the operating system.

CSSM_API_MEMORY_FUNCS_PTR CSSMAPI CSSM_GETAPIMemoryFunctions This function returns the pointer to the memory function table associated with the add-in handle.

2.2 Data Structures

2.2.1 CSSM_INFO

This data structure represents the information associated with an installation of CSSM.

```
typedef struct cssm_info{
    uint32 VerMajor;
    uint32 VerMinor;
}CSSM_INFO, *CSSM_INFO_PTR
```

Definition:

VerMajor - major version number

VerMinor - minor version number

2.2.2 CSSM_BOOL

This data type is used to indicate conditional responses to a function.

```
typedef enum cssm_bool {
       CSSM_TRUE = 1,
       CSSM_FALSE = 0
} CSSM_BOOL
```

Definition:

CSSM_TRUE - indicates operation was successful

CSSM_FALSE - indicates operation was unsuccessful

2.2.3 CSSM_RETURN

This data type is used to indicate whether a function was successful.

```
typedef enum cssm_return {
       CSSM_OK = 0,
       CSSM_FAIL = -1
} CSSM_RETURN
```

Definition:

CSSM_OK - indicates operation was successful

CSSM_FAIL - indicates operation was unsuccessful

2.2.4 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;  /* in bytes */
    uint8 *Data;
} CSSM_DATA, *CSSM_DATA_PTR
```

Definition:

Length - length of the data buffer in bytes

Data - points to the start of an arbitrary length data buffer

2.2.5 CSSM_GUID

This structure designates a global unique identifier (GUID) that distinguishes one add-in module from another. All GUID values should be computer-generated to guarantee uniqueness (the GUID generator in Microsoft Developer Studio* and the RPC UUIDGEN/uuid_gen program on a number of UNIX* platforms can be used).

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

Definition:

Data1 - Specifies the first eight hexadecimal digits of the GUID.

Data2 - Specifies the first group of four hexadecimal digits of the GUID.

Data3 - Specifies the second group of four hexadecimal digits of the GUID.

Data4 - Specifies an array of eight elements that contains the third and final group of eight hexadecimal digits of the GUID in elements 0 and 1, and the final 12 hexadecimal digits of the GUID in elements 2 through 7.

2.2.6 CSSM_LIST_ITEM

This structure is used to encapsulate the name and GUID of an add-in module.

```
typedef struct cssm_list_item{
    CSSM_GUID GUID;
    char *Name;
} CSSM_LIST_ITEM, *CSSM_LIST_ITEM_PTR
```

Definition:

GUID - the global unique identifier of the module

Name - the name of the module

2.2.7 CSSM_LIST

This structure is used to encapsulate an array of CSSM_LIST_ITEMs, where the array length is given by the Length variable.

```
typedef struct cssm_list{
    uint32 NumberItems;
    CSSM_LIST_ITEM_PTR Items;
} CSSM_LIST, *CSSM_LIST_PTR
```

Definition:

Data - an array of name and GUID pairs

Length - the number of entries in the *Data* array

2.2.8 CSSM_API_MEMORY_FUNCS

This structure is used by applications to supply memory functions for the CSSM and the add-in modules. The functions are used when memory needs to be allocated by the CSSM or add-ins for returning data structures to the applications.

```
typedef struct cssm_api_memory_funcs {
    void * (*malloc_func) (uint32 size, void *allocRef);
    void (*free_func) (void *memblock, void *allocRef);
    void * (*realloc_func) (void *memblock, uint32 size, void *allocRef);
    void * (*calloc_func) (uint32 num, uint32 size, void *allocRef);
} CSSM_API_MEMORY_FUNCS, *CSSM_API_MEMORY_FUNCS_PTR
```

Definition:

malloc_func - pointer to function that returns a void pointer to the allocated memory block of at least *size* bytes from heap allocRef

free_func - pointer to function that deallocates a previously-allocated memory block (*memblock*) from heap allocRef

realloc_func - pointer to function that returns a void pointer to the reallocated memory block (*memblock*) of at least *size* bytes from heap allocRef

calloc_func - pointer to function that returns a void pointer to an array of *num* elements of length *size* initialized to zero from heap allocRef

See Appendix B for details about the application memory functions.

2.3 Core Functions

2.3.1 CSSM_Init

CSSM_RETURN CSSMAPI CSSM_Init (
uint32 CheckCompatibleVerMajor,
uint32 CheckCompatibleVerMinor,
const CSSM_API_MEMORY_FUNCS_PTR MemoryFuncs,
const void * Reserved)

This function initializes CSSM and verifies that the version of CSSM expected by the application is compatible with the version of CSSM on the system. This function should be called once by each application.

Parameters

CheckCompatibleVerMajor(input)
The major version number of the CSSM release the application is compatible with.

CheckCompatibleVerMinor(input)
The minor version number of the CSSM release the application is compatible with.

MemoryFuncs (input)
Memory functions for the CSSM when allocating data structures for the application.

Reserved (input)
A reserved input.

Return Value

A CSSM_OK return value signifies the initialization operation was successful. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_INCOMPATIBLE_VERSION	Incompatible version

2.3.2 CSSM_GetInfo

CSSM_INFO_PTR CSSMAPI CSSM_GetInfo (void)

This function returns the version information of the CSSM Core.

Parameters

None

Return Value

A pointer to the CSSM_INFO structure. If the pointer is NULL, an error occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_MEMORY_ERROR	Error in allocating memory
CSSM_NOT_INITIALIZE	CSSM has not been initialized

See Also

CSSM_FreeInfo

2.3.3 CSSM_FreeInfo

CSSM_RETURN CSSMAPI CSSM_FreeInfo (CSSM_INFO_PTR CssmInfo)

This function frees the memory allocated for the CSSM_INFO structure in the CSSM_GetInfo function.

Parameters

CssmInfo (input/output)
A pointer to the CSSM_INFO structure to be freed.

Return Value

A CSSM_OK return value signifies the memory has been freed. When CSSM_FAIL is returned, an error occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_NOT_INITIALIZE	CSSM has not been initialized

See Also

CSSM_GetInfo

2.3.4 CSSM_VerifyComponents

CSSM_RETURN CSSMAPI CSSM_VerifyComponents (void)

This function performs an integrity check on all the components of CSSM to insure no tampering has occurred since installation.

Parameters

None

Return Value

A CSSM_TRUE return value signifies that all components verified successfully. When CSSM_FALSE is returned, either the verification failed or an error occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_VERIFY_COMPONENTS_FAILED	Unable to verify components
CSSM_INTEGRITY_COMPROMISED	Integrity check failed

2.4 Common Functions

2.4.1 CSSM_FreeList

CSSM_RETURN CSSMAPI CSSM_FreeList (CSSM_LIST_PTR CSSMList)

This function frees the memory allocated to hold a list of strings.

Parameters

CSSMList (input)
A pointer to the CSSM_LIST structure containing the GUID, name pair of add-ins.

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_POINTER	Invalid pointer input

2.4.2 CSSM_Free

void CSSMAPI CSSM_Free (void *MemPtr, CSSM_HANDLE AddInHandle)

This function frees the memory allocated by add-in.

Parameters

MemPtr (input)
A pointer to the memory to be freed.

AddInHandle (input)
The handle to add-in module that needs to free memory

Return Value

None

Error Codes

None

2.4.3 CSSM_GetAPIMemoryFunctions

CSSM_API_MEMORY_FUNCS_PTR CSSMAPI CSSM_GetAPIMemoryFunctions(
CSSM_HANDLE AddInHandle)

This function retrieves the memory function table associated with the add-in module.

Parameters

AddInHandle (input)
The handle to add-in module that is associated to memory function table.

Return Value

Non NULL if the function was successful. NULL if an error condition occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_ADDIN_HANDLE	Invalid add-in handle
CSSM_MEMORY_ERROR	Internal memory error

3. Cryptographic Services API

3.1 Overview

Cryptographic Service Providers (CSPs) are add-in modules which perform cryptographic operations including encryption, decryption, digital signaturing, key and key pair generation, random number generation, message digest, key wrapping, key unwrapping, and key exchange. Cryptographic services can be implemented by a hardware-software combination or by software only. Besides the traditional cryptographic functions, CSPs may provide other vendor-specific services. The set of services provided can be dynamic even after the CSP has been attached for service by a caller. This means the capabilities registered when the CSP was installed can change during execution based on changes internal or external to the system.

The CSP is always responsible for the secure storage of private keys. Optionally the CSP may assume responsibility for the secure storage of other object types, such as symmetric keys and certificates. The implementation of secured persistent storage for keys can use the services of a Data Storage Library module within the CSSM framework or some approach internal to the CSP. Accessing persistent objects managed by the CSP, other than keys, is performed using CSSM's Data Storage Library APIs.

CSPs optionally support a password-based login sequence. When login is supported, the caller is allowed to change passwords as deemed necessary. This is part of a standard user-initiated maintenance procedure. Some CSPs support operations for privileged, CSP administrators. The model for CSP administration varies widely among CSP implementations. For this reason, CSSM does not define APIs for vendor-specific CSP administration operations. CSP vendors can makes these services available to CSP administration tools using the CSSM_Passthrough function.

The range and types of cryptographic services a CSP supports is at the discretion of the vendor. A registry and query mechanism is available through the CSSM for CSPs to disclose the services and details about the services. As an example, a CSP may register with the CSSM: Encryption is supported, the algorithms present are DES with cipher block chaining for key sizes 40 and 56 bits, triple DES with 3 keys for key size 168 bits.

All cryptographic services requested by applications will be channeled to one of the CSPs via the CSSM. CSP vendors only need target their modules to CSSM for all security-conscious applications to have access to their product.

Calls made to a Cryptographic Service Provider (CSP) to perform cryptographic operations occur within a framework called a *session*, which is established and terminated by the application. The *session context* (simply referred to as the *context*) is created prior to starting CSP operations and is deleted as soon as possible upon completion of the operation. Context information is not persistent; it is not saved permanently in a file or database.

Before an application calls a CSP to perform a cryptographic operation, the application uses the query services function to determine what CSPs are installed, and what services they provide. Based on this information, the application then can determine which CSP to use for subsequent operations; the application creates a session with this CSP and performs the operation.

Depending on the class of cryptographic operations, individualized attributes are available for the cryptographic context. Besides specifying an algorithm when creating the context, the application may also initialize a session key, pass an initialization vector and/or pass padding information to complete the

description of the session. A successful return value from the create function indicates the desired CSP is available. Functions are also provided to manage the created context.

When a context is no longer required, the application calls CSSMDeleteContext. Resources that were allocated for that context can be reclaimed by the operating system.

Cryptographic operations come in two types — a single call to perform an operation and a staged method of performing the operation. For the single call method, only one call is needed to obtain the result. For the staged method, there is an initialization call followed by one or more update calls, and ending with a completion (final) call. The result is available after the final function completes its execution for most crypto operations — staged encryption/decryption are an exception in that each update call generates a portion of the result.

3.1.1 Cryptographic Context Operations

CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateKeyExchContext
CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateSignatureContext
CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateSymmetricContext
CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateDigestContext
CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateMacContext
CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateRandomGenContext
CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateUniqueIdContext
CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateAsymmetricContext
CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateDeriveKeyContext
CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateKeyGenContext
CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreatePassThroughContext—accepts as input a handle to the CSP that provides the cryptographic services and the necessary data to complete description of the cryptographic context. When the context is successfully created, a handle to a cryptographic context is returned to the calling application.

CSSM_CONTEXT_PTR CSSMAPI CSSM_CSP_GetContext—accepts as input the handle of a cryptographic context. The function returns a pointer to the context data structure that describes the handle.

3.1.2 Cryptographic Sessions and Logon

CSSM_RETURN CSSMAPI CSSM_CSP_Login—accepts as input a login password and a flag indicating the persistent or non-persistent status of keys and other objects created during the login session. CSPs are not required to support a login model. If a login model is supported, the CSP may request additional passwords at any time during the period of service.

CSSM_RETURN CSSMAPI CSSM_CSP_Logout—the caller is logged out of the current login session with the designated CSP.

CSSM_RETURN CSSMAPI CSSM_CSP_ChangeLoginPassword—accepts as input a handle to a CSP, the caller's old login password for that CSP, and the caller's new login password. The old password is replaced with the new password. The caller's current login is terminated and another login session is created using the new password.

3.1.3 Cryptographic Operations

CSSM_RETURN CSSMAPI CSSM_QuerySize accepts as input a handle to a cryptographic context describing the sign, digest, message authentication code, encryption, or decryption operation. This function returns pointers to variables indicating the block size (encryption and decryption only) and output size for the specified algorithm.

CSSM_RETURN CSSMAPI CSSM_SignData

CSSM_RETURN CSSMAPI CSSM_SignDataInit

CSSM_RETURN CSSMAPI CSSM_SignDataUpdate

CSSM_RETURN CSSMAPI CSSM_SignDataFinal accepts as input a handle to a cryptographic context describing the sign operation and the data to operate on. The result of the completed sign operation is returned in a CSSM_DATA structure.

CSSM_BOOL CSSMAPI CSSM_VerifyData

CSSM_RETURN CSSMAPI CSSM_VerifyDataInit

CSSM_RETURN CSSMAPI CSSM_VerifyDataUpdate

CSSM_BOOL CSSMAPI CSSM_VerifyDataFinal accepts as input a handle to a cryptographic context describing the verify operation and the data to operate on. The result of the completed verify operation is a CSSM_TRUE or CSSM_FALSE.

CSSM_RETURN CSSMAPI CSSM_DigestData

CSSM_RETURN CSSMAPI CSSM_DigestDataInit

CSSM_RETURN CSSMAPI CSSM_DigestDataUpdate

CSSM_RETURN CSSMAPI CSSM_DigestDataFinal accepts as input a handle to a cryptographic context describing the digest operation and the data to operate on. The result of the completed digest operation is returned in a CSSM_DATA structure.

CSSM_CC_HANDLE CSSMAPI CSSM_DigestDataClone accepts as input a handle to a cryptographic context describing the digest operation. A new handle to another cryptographic context is created with similar information and intermediate result as described by the first context.

CSSM_RETURN CSSMAPI CSSM_GenerateMac

CSSM_RETURN CSSMAPI CSSM_GenerateMacInit

CSSM_RETURN CSSMAPI CSSM_GenerateMacUpdate

CSSM_RETURN CSSMAPI CSSM_GenerateMacFinal accepts as input a handle to a cryptographic context describing the MAC operation and the data to operate on. The result of the completed MAC operation is returned in a CSSM_DATA structure.

CSSM_RETURN CSSMAPI CSSM_EncryptData

CSSM_RETURN CSSMAPI CSSM_EncryptDataInit

CSSM_RETURN CSSMAPI CSSM_EncryptDataUpdate

CSSM_RETURN CSSMAPI CSSM_EncryptDataFinal accepts as input a handle to a cryptographic context describing the encryption operation and the data to operate on. The encrypted data is returned in CSSM_DATA structures.

CSSM_RETURN CSSMAPI CSSM_DecryptData
CSSM_RETURN CSSMAPI CSSM_DecryptDataInit
CSSM_RETURN CSSMAPI CSSM_DecryptDataUpdate
CSSM_RETURN CSSMAPI CSSM_DecryptDataFinal accepts as input a handle to a cryptographic context describing the decryption operation and the data to operate on. The decrypted data is returned in CSSM_DATA structures.

CSSM_RETURN CSSMAPI CSSM_GenerateKey accepts as input a handle to a cryptographic context describing the generate key operation. The key is returned in a CSSM_KEY structure.

CSSM_RETURN CSSMAPI CSSM_GenerateKeyPair accepts as input a handle to a cryptographic context describing the generate key operation. The keys are returned in CSSM_KEY structures.

CSSM_RETURN CSSMAPI CSSM_GenerateRandom accepts as input a handle to a cryptographic context describing the generate random operation. The random data is returned in a CSSM_DATA structure.

CSSM_RETURN CSSMAPI CSSM_GenerateUniqueId accepts as input a handle to a cryptographic context describing the generate unique identifier operation. The unique identifier is returned in a CSSM_DATA structure.

CSSM_RETURN CSSMAPI CSSM_WrapKey accepts as input a handle to a symmetric/asymmetric cryptographic context describing the wrap key operation and the wrapping key to be used in the operation, the key to be wrapped, and a passphrase (if required by the CSP) that permits access to the private key to be wrapped.

CSSM_RETURN CSSMAPI CSSM_UnwrapKey accepts as input a handle to a cryptographic context describing the key unwrap operation, the wrapped key to be unwrapped, and a passphrase (if required by the CSP), that will be used to control access to the private key that will be unwrapped.

CSSM_RETURN CSSMAPI CSSM_DeriveKey accepts as input a handle to a cryptographic context describing the derive key operation and the base key that will be used to derive new keys.

CSSM_RETURN CSSMAPI CSSM_KeyExchGenParam
CSSM_RETURN CSSMAPI CSSM_KeyExchPhase1
CSSM_RETURN CSSMAPI CSSM_KeyExchPhase2 accepts as input a handle to a cryptographic context describing the key exchange operation. The intermediate results are returned in a CSSM_DATA structure. For the exchange to be successful, it has to complete phase 2 of the sequence.

3.1.4 Module Management Functions

CSSM_RETURN CSSMAPI CSSM_CSP_Install ( ) accepts as input the GUID of the CSP module, selected attributes describing the module, and information required by CSSM to dynamically load the module if its use is requested by some application. CSSM adds the CSP module name and attributes to the registry of CSP modules.

CSSM_RETURN CSSMAPI CSSM_CSP_Uninstall ( ) CSSM removes a specified CSP module from the CSP module registry.

CSSM_LIST_PTR CSSMAPI CSSM_CSP_ListModules (-)CSSM returns a list of all currently-registered CSP modules.

CSSM_CSP_HANDLECSSMAPI CSSM_CSP_Attach ( )- accepts as input the GUID of a CSP module, a major and minor version of the caller, a physical slot identifier (if appropriate for the selected CSP), a bitmask of session flags defining preferences using the CSP, and an optional notification callback function to be used by a CSP supporting detached operation. The application is requesting a dynamic load of the specified CSP module, if the available version of the CSP module is compatible with the version level specified by the caller.

CSSM_RETURN CSSMAPI CSSM_CSP_Detach ( ) the application is requesting the dynamic unload of a specified CSP module.

CSSM_CSPINFO_PTR CSSMAPI CSSM_CSP_GetInfo (-)CSSM returns one or more information structures describing the capabilities of a specified CSP module as it is recorded in the CSP module registry. One information structure is returned for each logical or physical slot managed by the CSP. The caller can select to receive all cpability structures or only those containing a complete and current description of the CSP's dynamic capabilities.

CSSM_RETURN CSSMAPI CSSM_CSP_FreeInfo ( ) accepts as input the pointer to the CSP information structures allocated by the CSSM. This function reclaims memory for use by the operating system.

CSSM_HANDLEINFO_PTR CSSMAPI CSSM_GetHandleInfo ( )accepts as input the handle to a CSP. CSSM returns the basic meta-information describing the identity and state of the CSP. This function does not return the capability descriptions for this CSP.

3.1.5 Extensibility Functions

CSSM_RETURN CSSMAPI CSSM_CSP_PassThroughaccepts as input an operation ID and a set of arbitrary input parameters. The operation ID may specify any type of operation the CSP wishes to export for use by an application. Such operations may include queries or services that are specific to the CSP.

3.2 Data Structures

```
typedef uint32 CSSM_CC_HANDLE    /* Cryptographic Context Handle */
typedef uint32 CSSM_CSP_HANDLE  /* Cryptographic Service Provider Handle */
typedef CSSM_CONTEXT CSSM_CONTEXTINFO
```

3.2.1 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;  /* in bytes */
    uint8 *Data;
} CSSM_DATA, *CSSM_DATA_PTR
```

Definition:

Length - length of the data buffer in bytes

Data - points to the start of an arbitrary length data buffer

3.2.2 CSSM_KEYHEADER

```
typedef struct CSSM_KeyHeader{
    uint32 HeaderFormatVersion;
    CSSM_GUID CspId;
    uint32 BlobDescription;
    uint32 DataFormatVersion;
    uint32 AlgorithmId;
    uint32 KeyUsage;
    uint32 SizeInBits;  /* in bits */
    uint32 WrapMethod;
    uint32 Reserved;
} CSSM_KEYHEADER, *CSSM_KEYHEADER_PTR
```

Definition:

HeaderFormatVersion- Version number of the KeyHeader format. Current value is CSSM_KEYHEADER_VERSION (0x01).

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

BlobDescription- KeyBlob Description Mask. When creating a BlobDescription Mask, use one from each of the following groups :

```
/* Wrap state */
#define CSSM_BLOBDESC_WRAP_MASK          (0x80000000) /* Use to mask wrap flag
*/
#define CSSM_BLOBDESC_UNWRAPPED          (0x00000000) /* Key is cleartext, can
be
                                                      parsed */
#define CSSM_BLOBDESC_WRAPPED            (0x80000000) /* Key is encrypted, might
                                                      not be parseable */

/* Transient State */
#define CSSM_BLOBDESC_TRANS_MASK         (0x40000000) /* Use to mask transient
                                                      flag */
#define CSSM_BLOBDESC_PERMANENT          (0x00000000) /* Data constant across
                                                      attaches */
#define CSSM_BLOBDESC_TRANSIENT          (0x40000000) /* Data not constant
across
                                                      attaches */

/* Data Type */
#define CSSM_BLOBDESC_TYPE_MASK          (0x30000000) /* Use to mask type */
#define CSSM_BLOBDESC_DATA               (0x00000000) /* Actual k  ey data */
#define CSSM_BLOBDESC_LABEL                    (0x10000000) /* Label ref to key
*/
#define CSSM_BLOBDESC_HANDLE             (0x20000000) /* Handle ref to key */

/* Contents */
#define CSSM_BLOBDESC_CONTENTS_MASK      (0x0F000000) /* Mask contents value */
#define CSSM_BLOBDESC_PUBLIC_KEY         (0x00000000)
#define CSSM_BLOBDESC_PRIVATE_KEY              (0x01000000)
#define CSSM_BLOBDESC_SESSION_KEY              (0x02000000)
#define CSSM_BLOBDESC_SECRET_PART              (0x03000000) /* Part of shared
secret */

/* Data Format */
#define CSSM_BLOBDESC_FORMAT_MASK              (0x0 0FFFF00) /* Mask format
value */
#define CSSM_BLOBDESC_RAW                (0x00000000) /* Single part key data,
no encoding */
#define CSSM_BLOBDESC_BER                (0x00000100)
#define CSSM_BLOBDESC_PKCS1                    (0x00000200) /* RSA Inc PKCS#1 -
RSA*/
#define CSSM_BLOBDESC_PKCS3                    (0x00000300) /* RSA Inc PKCS#3 -
Diffie-
                                                      Hellman */
#define CSSM_BLOBDESC_MSCAPI             (0x00000400) /* Microsoft CAPI */
#define CSSM_BLOBDESC_PGP                (0x00000500)
#define CSSM_BLOBDESC_FIPS186            (0x00000600) /* FIPS Pub 186 - DSS */
```

DataFormatVersion- Version number of the KeyData format. Current value is CSSM_DATAFORMAT_VERSION (0x01).

AlgorithmId - Algorithm identifier for the key contained by the key blob. Valid identifier values are indicated in Table 3 below.

KeyUsage - Mask describing authorized key usage modes. The identified list of key usage masks is shown below:

```
/* Key usage masks */
#define CSSM_KEYUSE_ENCRYPT                     0x0001
#define CSSM_KEYUSE_DECRYPT                     0x0002
#define CSSM_KEYUSE_SIGN                0x0004
#define CSSM_KEYUSE_VERIFY              0x0008
#define CSSM_KEYUSE_WRAP                0x0010
#define CSSM_KEYUSE_UNWRAP              0x0020
#define CSSM_KEYUSE_DERIVE              0x0040
```

SizeInBits - Size of the key in bits. This is the logical length of the key in bits, which translates to be the actual length of a key for symmetric algorithms or the length of the modulus for asymmetric algorithms.

WrapMethod- Key wrapping scheme. The key wrapping methods currently defined are the symmetric and asymmetric encryption algorithms listed in Table 3 below.

Reserved - Reserved for future use.

3.2.3 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.

3.2.4 CSSM_KEY

```
typedef CSSM_DATA    CSSM_KEY, *CSSM_KEY_PTR
typedef CSSM_KEY     CSSM_WRAP_KEY, *CSSM_WRAP_KEY_PTR
```

3.2.5 CSSM_CALLBACK

```
typedef CSSM_DATA_PTR (CALLBACK *CSSM_CALLBACK) (void *allocRef, uint32 ID);
```

Definition:

allocRef - Memory heap reference specifying which heap to use for memory allocation.

ID - Input data to identify the callback.

3.2.6 CSSM_CRYPTO_DATA

```
typedef struct cssm_crypto_data {
       CSSM_DATA_PTR Param;
       CSSM_CALLBACK Callback;
       uint32 ID;
}CSSM_CRYPTO_DATA, *CSSM_CRYPTO_DATA_PTR
```

Definition:

Param - A pointer to the parameter data and its size in bytes.

Callback - An optional callback routine for the add-in modules to obtain the parameter.

ID - A tag that identifies the callback.

3.2.7 CSSM_CSP_TYPE

```
typedef enum cssm_csp_type {
       CSSM_CSPTYPE_HW      = 0,
       CSSM_CSPTYPE_SW      = CSSM_CSPTYPE_HW+1,
       CSSM_CSPTYPE_HYBRID  = CSSM_CSPTYPE_HW+2
}CSSM_CSP_TYPE;
```

3.2.8 CSSM_CSP_SESSION_TYPE

```
#define       CSSM_CSP_SESSION_EXCLUSIVE          0x0001
#define       CSSM_CSP_SESSION_READWRITE          0x0002
#define       CSSM_CSP_SESSION_SERIAL             0x0004
```

3.2.9 CSSM_NOTIFY_CALLBACK

```
typedef CSSM_RETURN (*CSSM_NOTIFY_CALLBACK)(CSSM_CSP_HANDLE hCSP,
                                            uint32 Application,
                                            uint32 Reason,
                                            uint32 Param)
```

Definition:

hCSP - Handle of the add-in to which the notification applies.

Application - Application specific context indicator. This value is specified when an add-in module is attached.

Reason - One of the values specified below.

```
#define       CSSM_NOTIFY_SURRENDER              0
#define       CSSM_NOTIFY_COMPLETE                      1
#define       CSSM_NOTIFY_DEVICE_REMOVED         2
#define       CSSM_NOTIFY_DEVICE_INSERTED               3
```

Param - Used by the add-in that triggers the notification to pass relevant information about the notification to the application. This parameter will contain the cryptographic context handle for the CSSM_NOTIFY_SURRENDER/COMPLETE types and zero for the CSSM_NOTIFY_DEVICE_REMOVED/INSERTED types.

3.2.10 CSSM_HANDLEINFO

```
typedef struct cssm_handleinfo {
       uint32 SlotID;
       uint32 SessionFlags;
       CSSM_NOTIFY_CALLBACK Callback;
       uint32 ApplicationContext;
} CSSM_HANDLEINFO, *CSSM_HANDLEINFO_PTR;
```

3.2.11 CSSM_CSPPININFO

```
typedef struct cssm_csppininfo {
       uint32 MaxLength;
       uint32 MinLength;
} CSSM_CSPPININFO, *CSSM_CSPPININFO_PTR;
```

3.2.12 CSSM_CSPMEMINFO

```
typedef struct cssm_cspmeminfo {
       uint32 PublicMem;
       uint32 FreePublicMem;
       uint32 PrivateMem;
       uint32 FreePrivateMem;
} CSSM_CSPMEMINFO, *CSSM_CSPMEMINFO_PTR;
```

3.2.13 CSSM_CSPSESSIONINFO

```
typedef struct cssm_cspsessioninfo {
       uint32 MaxSessions;
       uint32 OpenedSessions;
       uint32 MaxRWSessions;
       uint32 OpenedRWSessions;
} CSSM_CSPSESSIONINFO, CSSM_CSPSESSIONINFO_PTR;
```

3.2.14 CSSM_CSPINFO

```
typedef struct cssm_cspinfo {
       uint32 VerMajor;
       uint32 VerMinor;
       CSSM_BOOL ExportFlag;
       CSSM_BOOL MultiTasking;
       CSSM_BOOL SerialRequ  ired;
       CSSM_CSP_TYPE CSPType;
       CSSM_BOOL LoginRequired;
       uint32 SlotID;
       char *SlotDescription;
       char *SlotVendor;
       CSSM_BOOL SlotIsHardware;
       CSSM_DATA ExclusiveCSPCertificate;
       char *Vendor;
       char *Description;
       char *Label;
       CSSM_DATA SerialNumber;
       CSSM_BOOL Removable;
       CSSM_BOOL CapabilitiesInitialized;
       uint32 NumberOfCapabilities;
       CSSM_CONTEXT_PTR Capabilities;
       CSSM_CSPPININFO PinInfo;           /* CSP Pin information */
       CSSM_CSPMEMINFO MemInfo;           /* CSP memory information */
       CSSM_CSPSESSIONINFO SessionInf   o; /* CSP multitasking information */
}CSSM_CSPINFO, *CSSM_CSPINFO_PTR;
```

Definition:

VerMajor - Major version number.

VerMinor - Minor version number.

ExportFlag - Exportable flag.

MultiTasking - Flag to indicate if CSP handles multitasking.

SerialRequired - Flag to indicate if CSP is only capable of executing operations in serial mode. If this flag is set, the CSSM_CSP_SESSION_SERIAL flag should be used during attach.

CSPType - Enumerated value indicating CSP type.

LoginRequired - True or false, indicating whether the CSP requires caller login and logout.

SlotID - Identifier for a slot in a hardware token/CSP.

SlotDescription - Description of the token slot (whether physical or virtual).

SlotVendor - Manufacturer of the slot device.

SlotIsHardware- True or False, indicating whether the CSP is hardware or software.

ExclusiveCSPCertificate - The certificate used to sign certificates issued to exclusive users of this CSP.

Vendor - CSP Vendor name.

Description - Detailed description field for the CSP.

Label - CSP Label.

SerialNumber - Serial number of the CSP.

Removable - True or false, indicating whether the CSP can be removed from the slot.

CapabilitiesInitialized- True or false, indicating whether complete capabilities are currently specified in this CSPinfo structure.

NumberOfCapabilities- Number of contexts.

Capabilities - Pointer to a CSSM_CONTEXT structure describing CSP capabilities and attributes.

PinInfo - Optional information on the minimum and maximum PIN lengths allowed by the CSSM_CSP_Login/Logout APIs.

MemInfo - Optional information on the amount of free memory (both public and private) available in the CSP for storing keys and other security objects.

SessionInfo - Optional information on the maximum number and the current number of cryptographic sessions with this CSP.

3.2.15 CSSMContextAttributes

```
typedef struct cssm_context_attribute{
    uint32 AttributeType;    /* attribute type */
    uint32 AttributeLength;  /* length of attribute */
    union {
        uint8 *Description;
        uint32 *Length;
        void *Pointer;
        CSSM_CRYPTO_DATA_PTR SeedPassPhrase;
        CSSM_KEY_PTR Key;
        CSSM_DATA_PTR Data;
    }Attribute;       /* data that describes attribute */
}CSSM_CONTEXT_ATTRIBUTE, *CSSM_CONTEXT_ATTRIBUTE_PTR
```

Definition:

AttributeType - An identifier describing the type of attribute.

Table 1. Attribute types

Value	Description
CSSM_ATTRIBUTE_NONE	No attribute
CSSM_ATTRIBUTE_CUSTOM	Custom data
CSSM_ATTRIBUTE_DESCRIPTION	Description of attribute
CSSM_ATTRIBUTE_KEY	Key Data
CSSM_ATTRIBUTE_INIT_VECTOR	Initialization vector
CSSM_ATTRIBUTE_SALT	Salt
CSSM_ATTRIBUTE_PADDING	Padding information
CSSM_ATTRIBUTE_RANDOM	Random data
CSSM_ATTRIBUTE_SEED	Seed
CSSM_ATTRIBUTE_PASSPHRASE	Pass phrase
CSSM_ATTRIBUTE_KEY_LENGTH	Key length (specified in bits)
CSSM_ATTRIBUTE_MODULUS_LEN	Modulus length (specified in bits)
CSSM_ATTRIBUTE_INPUT_SIZE	Input size
CSSM_ATTRIBUTE_OUTPUT_SIZE	Output size
CSSM_ATTRIBUTE_ROUNDS	Number of runs (or rounds)

AttributeLength - Length of the attribute data.

Attribute - Attribute data. Depending on the*AttributeType*, the attribute data represents different information.

3.2.16 CSSMContext

```
typedef uint32 CSSM_CC_HANDLE    /* Cryptographic Context Handle */
typedef CSSM_CONTEXT CSSM_CONTEXTINFO

typedef struct cssm_context {
    uint32 ContextType;       /* context type */
    uint32 AlgorithmType;      /* algorithm type of context */
    uint32 Mode;               /* for encryption only */
    uint32 Reserve;            /* reserved for future use */
    uint32 NumberOfAttributes; /* number of attributes associated with context
*/
    CSSM_CONTEXT_ATTRIBUTE_PTR ContextAttributes;    /* pointer to attributes
*/
} CSSM_CONTEXT, *CSSM_CONTEXT_PTR
```

Definitions:

ContextType - An identifier describing the type of services for this context.

Table 2. Context types

Value	Description
CSSM_ALGCLASS_NONE	Null Context type
CSSM_ALGCLASS_CUSTOM	Custom Algorithms
CSSM_ALGCLASS_KEYXCH	Key Exchange Algorithms
CSSM_ALGCLASS_SIGNATURE	Signature Algorithms
CSSM_ALGCLASS_SYMMETRIC	Symmetric Encryption Algorithms
CSSM_ALGCLASS_DIGEST	Message Digest Algorithms
CSSM_ALGCLASS_RANDOMGEN	Random Number Generation Algorithms
CSSM_ALGCLASS_UNIQUEGEN	Unique ID Generation Algorithms
CSSM_ALGCLASS_MAC	Message Authentication Code Algorithms
CSSM_ALGCLASS_ASYMMETRIC	Asymmetric Encryption Algorithms
CSSM_ALGCLASS_KEYGEN	Key Generation Algorithms
CSSM_ALGCLASS_DERIVEKEY	Key Derivation Algorithms

AlgorithmType - An ID number describing the algorithm to be used.

Table 3. Algorithms for a session context.

Value	Description
CSSM_ALGID_NONE	Null algorithm
CSSM_ALGID_CUSTOM	Custom algorithm
CSSM_ ALGID_DH	Diffie Hellman key exchange algorithm
CSSM_ ALGID_PH	Pohlig Hellman key exchange algorithm
CSSM_ ALGID_KEA	Key Exchange Algorithm
CSSM_ ALGID_MD2	MD2 hash algorithm
CSSM_ALGID_MD4	MD4 hash algorithm
CSSM_ ALGID_MD5	MD5 hash algorithm
CSSM_ ALGID_SHA1	Secure Hash Algorithm
CSSM_ALGID_NHASH	N-Hash algorithm

CSSM_ALGID_HAVAL	HAVAL hash algorithm (MD5 variant)
CSSM_ALGID_RIPEMD	RIPE-MD hash algorithm (MD4 variant - developed for the European Community's RIPE project)
CSSM_ALGID_IBCHASH	IBC-Hash (keyed hash algorithm or MAC)
CSSM_ALGID_RIPEMAC	RIPE-MAC
CSSM_ALGID_DES	Data Encryption Standard block cipher
CSSM_ALGID_DESX	DESX block cipher (DES variant from RSA)
CSSM_ALGID_RDES	RDES block cipher (DES variant)
CSSM_ALGID_3DES_3KEY	Triple-DES block cipher (with 3 keys)
CSSM_ALGID_3DES_2KEY	Triple-DES block cipher (with 2 keys)
CSSM_ALGID_3DES_1KEY	Triple-DES block cipher (with 1 key)
CSSM_ALGID_IDEA	IDEA block cipher
CSSM_ALGID_RC2	RC2 block cipher
CSSM_ALGID_RC5	RC5 block cipher
CSSM_ ALGID_RC4	RC4 stream cipher
CSSM_ ALGID_SEAL	SEAL stream cipher
CSSM_ALGID_CAST	CAST block cipher
CSSM_ALGID_BLOWFISH	BLOWFISH block cipher
CSSM_ALGID_SKIPJACK	Skipjack block cipher
CSSM_ALGID_LUCIFER	Lucifer block cipher
CSSM_ALGID_MADRYGA	Madryga block cipher
CSSM_ALGID_FEAL	FEAL block cipher
CSSM_ALGID_REDOC	REDOC 2 block cipher
CSSM_ALGID_REDOC3	REDOC 3 block cipher
CSSM_ALGID_LOKI	LOKI block cipher
CSSM_ALGID_KHUFU	KHUFU block cipher
CSSM_ALGID_KHAFRE	KHAFRE block cipher
CSSM_ALGID_MMB	MMB block cipher (IDEA variant)
CSSM_ALGID_GOST	GOST block cipher
CSSM_ALGID_SAFER	SAFER K-64 block cipher
CSSM_ALGID_CRAB	CRAB block cipher
CSSM_ALGID_RSA	RSA public key cipher
CSSM_ALGID_DSA	Digital Signature Algorithm
CSSM_ ALGID_MD5WithRSA	MD5/RSA signature algorithm
CSSM_ALGID_MD2WithRSA	MD2/RSA signature algorithm
CSSM_ALGID_ElGamal	ElGamal signature algorithm
CSSM_ALGID_MD2Random	MD2-based random numbers
CSSM_ALGID_MD5Random	MD5-based random numbers
CSSM_ALGID_SHARandom	SHA-based random numbers
CSSM_ALGID_DESRandom	DES-based random numbers
CSSM_ALGID_SHA1WithRSA	SHA-1/RSA signature algorithm
CSSM_ALGID_RSA_PKCS	RSA as specified in PKCS #1
CSSM_ALGID_RSA_ISO9796	RSA as specified in ISO 9796
CSSM_ALGID_RSA_RAW	Raw RSA as assumed in X.509
CSSM_ALGID_CDMF	CDMF block cipher
CSSM_ALGID_CAST3	Entrust's CAST3 block cipher
CSSM_ALGID_CAST5	Entrust's CAST5 block cipher
CSSM_ALGID_GenericSecret	Generic secret operations
CSSM_ALGID_ConcatBaseAndKey	Concatenate two keys, base key first
CSSM_ALGID_ConcatKeyAndBase	Concatenate two keys, base key last
CSSM_ALGID_ConcatBaseAndData	Concatenate base key and random data, key first
CSSM_ALGID_ConcatDataAndBase	Concatenate base key and data, data first

CSSM_ALGID_XORBaseAndData	XOR a byte string with the base key
CSSM_ALGID_ExtractFromKey	Extract a key from base key, starting at arbitrary bit position
CSSM_ALGID_SSL3PreMasterGen	Generate a 48 byte SSL 3 pre-master key
CSSM_ALGID_SSL3MasterDerive	Derive an SSL 3 key from a pre-master key
CSSM_ALGID_SSL3KeyAndMacDerive	Derive the keys and MACing keys for the SSL cipher suite
CSSM_ALGID_SSL3MD5_MAC	Performs SSL 3 MD5 MACing
CSSM_ALGID_SSL3SHA1_MAC	Performs SSL 3 SHA-1 MACing
CSSM_ALGID_MD5Derive	Generate key by MD5 hashing a base key
CSSM_ALGID_MD2Derive	Generate key by MD2 hashing a base key
CSSM_ALGID_SHA1Derive	Generate key by SHA-1 hashing a base key
CSSM_ALGID_WrapLynks	Spyrus LYNKS DES based wrapping scheme w/checksum
CSSM_ALGID_WrapSET_OAEP	SET key wrapping
CSSM_ALGID_BATON	Fortezza BATON cipher
CSSM_ALGID_ECDSA	Elliptic Curve DSA
CSSM_ALGID_MAYFLY	Fortezza MAYFLY cipher
CSSM_ALGID_JUNIPER	Fortezza JUNIPER cipher
CSSM_ALGID_FASTHASH	Fortezza FASTHASH

Mode - An algorithm mode — values identified in table below apply only to symmetric algorithms.

Table 4. Modes of algorithms.

Value	Description
CSSM_ALGMODE_NONE	Null Algorithm mode
CSSM_ALGMODE_CUSTOM	Custom mode
CSSM_ALGMODE_ECB	Electronic Code Book
CSSM_ALGMODE_ECBPad	ECB with padding
CSSM_ALGMODE_CBC	Cipher Block Chaining
CSSM_ALGMODE_CBC_IV8	CBC with Initialization Vector of 8 bytes
CSSM_ALGMODE_CBCPadIV8	CBC with padding and Initialization Vector of 8 bytes
CSSM_ALGMODE_CFB	Cipher FeedBack
CSSM_ALGMODE_CFB_IV8	CFB with Initialization Vector of 8 bytes
CSSM_ALGMODE_OFB	Output FeedBack
CSSM_ALGMODE_OFB_IV8	OFB with Initialization Vector of 8 bytes
CSSM_ALGMODE_COUNTER	Counter
CSSM_ALGMODE_BC	Block Chaining
CSSM_ALGMODE_PCBC	Propagating CBC
CSSM_ALGMODE_CBCC	CBC with Checksum
CSSM_ALGMODE_OFBNLF	OFB with NonLinear Function
CSSM_ALGMODE_PBC	Plaintext Block Chaining
CSSM_ALGMODE_PFB	Plaintext FeedBack
CSSM_ALGMODE_CBCPD	CBC of Plaintext Difference
CSSM_ALGMODE_PUBLIC_KEY	Use the public key
CSSM_ALGMODE_PRIVATE_KEY	Use the private key
CSSM_ALGMODE_SHUFFLE	Fortezza shuffle mode

NumberOfAttributes- Number of attributes associated with this service.

ContextAttributes - Pointer to data that describes the attributes. To retrieve the next attribute, advance the attribute pointer.

3.3 Cryptographic Context Operations

3.3.1 CSSM_CSP_CreateKeyExchContext

CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateKeyExchContext
(CSSM_CSP_HANDLE CSPHandle,
uint32 AlgorithmID)

This function creates a key exchange context given a handle of a CSP, an algorithm identification number, a key, and the length of the key in bits. The cryptographic context handle is returned. The cryptographic context handle can be used to call key exchange functions.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform this function. If a NULL handle is specified, CSSM returns error.

AlgorithmID (input)
The algorithm identification number for the algorithm used to do the key exchange.

Return Value

Returns a cryptographic context handle. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_CSP_HANDLE	Invalid provider handle
CSSM_MEMORY_ERROR	Internal memory error

See Also

CSSM_KeyExchPhase1, CSSM_KeyExchPhase2, CSSM_KeyExchGenParam, CSSM_GetContext, CSSM_SetContext, CSSM_DeleteContext, CSSM_GetContextAttribute, CSSM_UpdateContextAttributes

3.3.2 CSSM_CSP_CreateSignatureContext

CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateSignatureContext
(CSSM_CSP_HANDLE CSPHandle,
uint32 AlgorithmID,
const CSSM_CRYPTO_DATA_PTR PassPhrase,
const CSSM _KEY_PTR Key)

This function creates a signature cryptographic context for sign and verify given a handle of a CSP, an algorithm identification number, a key, and the length of the key in bits. The cryptographic context handle is returned. The cryptographic context handle can be used to call sign and verify cryptographic functions.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform this function. If a NULL handle is specified, CSSM returns error.

AlgorithmID (input)
The algorithm identification number for a signature/verification algorithm.

PassPhrase (input)
The passphrase used to unlock the private key. Optionally, the application can provide a pointer to a callback function. In which case, the CSP will invoke the callback to obtain the passphrase. The passphrase is needed only for signature operations, not verify operations.

Key (input)
The key used to sign. The caller passes in a pointer to a CSSM_KEY structure containing the key.

Return Value

Returns a cryptographic context handle. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_CSP_HANDLE	Invalid provider handle
CSSM_MEMORY_ERROR	Internal memory error

See Also

CSSM_SignData, CSSM_SignDataInit, CSSM_SignDataUpdate, CSSM_SignDataFinal, CSSM_VerifyData, CSSM_VerifyDataInit, CSSM_VerifyDataUpdate, CSSM_VerifyDataFinal, CSSM_GetContext, CSSM_SetContext, CSSM_DeleteContext, CSSM_GetContextAttribute, CSSM_UpdateContextAttributes

3.3.3 CSSM_CSP_CreateSymmetricContext

CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateSymmetricContext
(CSSM_CSP_HANDLE CSPHandle,
uint32 AlgorithmID,
uint32 Mode,
const CSSM _KEY_PTR Key,
const CSSM_DATA_PTR InitVector,
uint32 Padding,
uint32 Rounds)

This function creates a symmetric encryption cryptographic context given a handle of a CSP, an algorithm identification number, a key, an initial vector, padding, and the number of encryption rounds. The cryptographic context handle is returned. The cryptographic context handle can be used to call symmetric encryption functions and the cryptographic wrap/unwrap functions.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform this function. If a NULL handle is specified, CSSM returns error.

AlgorithmID (input)
The algorithm identification number for symmetric encryption.

Mode (input)
The mode of the specified algorithm ID.

Key (input)
The key used for symmetric encryption. The caller passes in a pointer to a CSSM_KEY structure containing the key.

InitVector (input/optional)
The initial vector for symmetric encryption; typically specified for block ciphers.

Padding (input/optional)
The method for padding; typically specified for ciphers that pad.

Rounds (input/optional)
Specifies the number of rounds of encryption; used for ciphers with variable number of rounds.

Return Value

Returns a cryptographic context handle. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_CSP_HANDLE	Invalid provider handle
CSSM_MEMORY_ERROR	Internal memory error

See Also

CSSM_EncryptData, CSSM_QuerySize, CSSM_EncryptDataInit, CSSM_EncryptDataUpdate, CSSM_EncryptDataFinal, CSSM_DecryptData, CSSM_DecryptDataInit, CSSM_DecryptDataUpdate, CSSM_DecryptDataFinal, CSSM_GetContext, CSSM_SetContext, CSSM_DeleteContext, CSSM_GetContextAttribute, CSSM_UpdateContextAttributes

3.3.4 CSSM_CSP_CreateDigestContext

CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateDigestContext
(CSSM_CSP_HANDLE CSPHandle,
uint32 AlgorithmID)

This function creates a digest cryptographic context, given a handle of a CSP and an algorithm identification number. The cryptographic context handle is returned. The cryptographic context handle can be used to call digest cryptographic functions.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform this function. If a NULL handle is specified, CSSM returns error.

AlgorithmID (input)
The algorithm identification number for message digests.

Return Value

Returns a cryptographic context handle. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_CSP_HANDLE	Invalid crypto services provider handle
CSSM_MEMORY_ERROR	Internal memory error

See Also

CSSM_DigestData, CSSM_DigestDataInit, CSSM_DigestDataUpdate, CSSM_DigestDataFinal, CSSM_GetContext, CSSM_SetContext, CSSM_DeleteContext, CSSM_GetContextAttribute, CSSM_UpdateContextAttributes

3.3.5 CSSM_CSP_CreateMacContext

CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateMacContext
(CSSM_CSP_HANDLE CSPHandle,
uint32 AlgorithmID,
const CSSM _KEY_PTR Key)

This function creates a message authentication code cryptographic context, given a handle of a CSP, algorithm identification number, key, and the length of the key in bits. The cryptographic context handle is returned. The cryptographic context handle can be used to call message authentication code functions.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform this function. If a NULL handle is specified, CSSM returns error.

AlgorithmID (input)
The algorithm identification number for the MAC algorithm.

Key (input)
The key used to generate a message authentication code. Caller passes in a pointer to a CSSM_KEY structure containing the key.

Return Value

Returns a cryptographic context handle. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_CSP_HANDLE	Invalid crypto services provider handle
CSSM_MEMORY_ERROR	Internal memory error

See Also

CSSM_GenerateMac, CSSM_GenerateMacInit, CSSM_GenerateMacUpdate, CSSM_GenerateMacFinal, CSSM_GetContext, CSSM_SetContext, CSSM_DeleteContext, CSSM_GetContextAttribute, CSSM_UpdateContextAttributes

3.3.6 CSSM_CSP_CreateRandomGenContext

CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateRandomGenContext
(CSSM_CSP_HANDLE CSPHandle,
uint32 AlgorithmID,
const CSSM_CRYPTO_DATA_PTR Seed,
uint32 Length)

This function creates a random number generation cryptographic context, given a handle of a CSP, an algorithm identification number, a seed, and the length of the random number in bytes. The cryptographic context handle is returned, and can be used for the random number generation function.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform this function. If a NULL handle is specified, CSSM returns error.

AlgorithmID (input)
The algorithm identification number for random number generation.

Seed (input/optional)
A seed used to generate random number. The caller can either pass a seed and seed length in bytes or pass in a callback function. If NULL is passed, the cryptographic service provider will use its default seed handling mechanism.

Length (input)
The length of the random number to be generated.

Return Value

Returns a cryptographic context handle. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_CSP_HANDLE	Invalid provider handle
CSSM_MEMORY_ERROR	Internal memory error

See Also

CSSM_GenerateRandom, CSSM_GetContext, CSSM_SetContext, CSSM_DeleteContext, CSSM_GetContextAttribute, CSSM_UpdateContextAttributes

3.3.7 CSSM_CSP_CreateUniqueIdContext

CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateUniqueIdContext
(CSSM_CSP_HANDLE CSPHandle,
uint32 AlgorithmID,
const CSSM_CRYPTO_DATA_PTR Seed,
uint32 Length)

This function creates a unique identification number generation cryptographic context, given a handle of a CSP, an algorithm identification number, a seed, and the length of the unique ID in bytes. The cryptographic context handle is returned. The cryptographic context handle can be used to call unique ID generation function.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform this function. If a NULL handle is specified, CSSM returns error.

AlgorithmID (input)
The algorithm identification number for unique identification generation.

Seed (input/optional)
A seed used to generate unique ID. The caller can either pass a seed and seed length in bytes or pass in a callback function. If NULL is passed, the cryptographic service provider will use its default seed handling mechanism.

Length (input)
The length of the unique ID to be generated.

Return Value

Returns a cryptographic context handle. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_CSP_HANDLE	Invalid provider handle
CSSM_MEMORY_ERROR	Internal memory error

See Also

CSSM_GenerateUniqueId, CSSM_GetContext, CSSM_SetContext, CSSM_DeleteContext, CSSM_GetContextAttribute, CSSM_UpdateContextAttributes

3.3.8 CSSM_CSP_CreateAsymmetricContext

CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateAsymmetricContext
(CSSM_CSP_HANDLE CSPHandle,
uint32 AlgorithmID,
const CSSM_CRYPTO_DATA_PTR PassPhrase,
const CSSM _KEY_PTR Key,
uint32 Padding,
uint32 Mode)

This function creates an asymmetric encryption cryptographic context , given a handle of a CSP, an algorithm identification number, a key, padding, and the key mode (CSSM_ALGMODE_PRIVATE_KEY or CSSM_ ALGMODE_PUBLIC_KEY). The cryptographic context handle is returned. The cryptographic context handle can be used to call asymmetric encryption functions and cryptographic wrap/unwrap functions.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform this function. If a NULL handle is specified, CSSM returns error.

AlgorithmID (input)
The algorithm identification number for the algorithm used for asymmetric encryption

PassPhrase (input)
The passphrase used to unlock the private key. Optionally, the application can provide a pointer to a callback function. In which case, the CSP will invoke the callback to obtain the passphrase. The passphrase is needed only for private key operations, not public key operations.

Key (input)
The key used for asymmetric encryption. The caller passes a pointer to a CSSM_KEY structure containing the key.

Padding (input/optional)
The method for padding. Typically specified for ciphers that pad.

Mode (input)
The mode indicates whether to use the private or public key (CSSM_ALGMODE_PRIVATE_KEY or CSSM_ALGMODE_PUBLIC_KEY).

Return Value

Returns a cryptographic context handle. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_CSP_HANDLE	Invalid provider handle
CSSM_MEMORY_ERROR	Internal memory error

See Also

CSSM_EncryptData, CSSM_QuerySize, CSSM_EncryptDataInit, CSSM_EncryptDataUpdate, CSSM_EncryptDataFinal, CSSM_DecryptData, CSSM_DecryptDataInit, CSSM_DecryptDataUpdate, CSSM_DecryptDataFinal, CSSM_GetContext, CSSM_SetContext, CSSM_DeleteContext, CSSM_GetContextAttribute, CSSM_UpdateContextAttributes

3.3.9 CSSM_CSP_CreateDeriveKeyContext

CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateDeriveKeyContext
(CSSM_CSP_HANDLE CSPHandle,
uint32 AlgorithmID,
CSSM_KEY_TYPE DeriveKeyType,
uint32 DeriveKeyLength,
uint32 IterationCount,
const CSSM_CRYPTO_DATA_PTR Seed,
const CSSM_CRYPTO_DATA_PTR PassPhrase)

This function creates a cryptographic context to derive a symmetric key given a handle of a CSP, an algorithm, the type of symmetric key to derive, the length of the derived key, and an optional seed or an optional passphrase from which to derive a new key. The cryptographic context handle is returned. The cryptographic context handle can be used for calling the cryptographic derive key function.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform this function. If a NULL handle is specified, CSSM returns an error.

AlgorithmID (input)
The algorithm identification number for a derived key algorithm.

DeriveKeyType (input)
The type of symmetric key to derive.

DeriveKeyLength (input)
The length of key to derive.

InterationCount (input/optional)
The number of iterations to be performed during the derivation process. Used heavily by password based derivation methods.

Seed (input/optional)
A seed used to generate a random number. The caller can either pass a seed and seed length in bytes or pass in a callback function. If NULL is passed, the cryptographic service provider will use its default seed handling mechanism.

PassPhrase (input/optional)
The passphrase or a callback function to be used to obtain the passphrase to be used in deriving a key.

Return Value

Returns a cryptographic context handle. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_CSP_HANDLE	Invalid provider handle
CSSM_MEMORY_ERROR	Internal memory error

See Also

CSSM_DeriveKey

3.3.10 CSSM_CSP_CreateKeyGenContext

CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateKeyGenContext
(CSSM_CSP_HANDLE CSPHandle,
uint32 AlgorithmID,
const CSSM_CRYPTO_DATA_PTR PassPhrase,
uint32 KeySizeInBits,
const CSSM_CRYPTO_DATA_PTR Seed,
const CSSM_DATA_PTR Salt,
const CSSM_DATA_PTR KeyLabel)

This function creates a key generation cryptographic context, given a handle of a CSP, an algorithm identification number, a pass phrase, a modulus size (for public/private keypair generation), a key size (for symmetric key generation), a seed, salt, and a label. The cryptographic context handle is returned. The cryptographic context handle can be used to call key/keypair generation functions.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform this function. If a NULL handle is specified, CSSM returns error.

AlgorithmID (input)
The algorithm identification number of the algorithm used for key generation.

PassPhrase (input)
The passphrase is used to wrap the private key upon generating a key pair or as a nickname persistently associated with the private key or the symmetric key that will be generated by the GenerateKey function. Optionally, the application can provide a pointer to a callback function, in which case, the CSP will invoke the callback to obtain the passphrase. Once the new key is created, the passphrase or nickname must be provided in all future references to access the private or symmetric key.

KeySizeInBits (input)
The logical size of the key (specified in bits). This refers to either the actual key size (for symmetric key generation) or the modulus size (for asymmetric key pair generation).

Seed (input/optional)
A seed used to generate the key. The caller can either pass a seed and seed length in bytes or pass in a callback function. If NULL is passed, the cryptographic service provider will use its default seed handling mechanism.

Salt (input/optional)
A Salt used to generate the key.

KeyLabel(input/optional)
A label that can be used to reference the key or key pair being generated.

Return Value

Returns a cryptographic context handle. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_CSP_HANDLE	Invalid provider handle
CSSM_MEMORY_ERROR	Internal memory error

See Also

CSSM_GenerateKey, CSSM_GetContext, CSSM_SetContext, CSSM_DeleteContext, CSSM_GetContextAttribute, CSSM_UpdateContextAttributes

3.3.11 CSSM_CSP_CreatePassThroughContext

CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreatePassThroughContext
(CSSM_CSP_HANDLE CSPHandle,
const CSSM_KEY_PTR Key,
const CSSM_DATA_PTR ParamBufs,
uint32 ParamBufCount)

This function creates a custom cryptographic context, given a handle of a CSP and pointer to a custom input data structure. The cryptographic context handle is returned. The cryptographic context handle can be used to call the CSSM pass-through function for the CSP.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform this function. If a NULL handle is specified, CSSM returns error.

Key (input)
The key to be used for the context. The caller passes in a pointer to a CSSM_KEY structure containing the key.

ParamBufs (input)
Array of input buffers to the pass-through call.

ParamBufCount (input)
The number of input buffers pointed to by *ParamBufs*.

Return Value

Returns a cryptographic context handle. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_CSP_HANDLE	Invalid provider handle
CSSM_MEMORY_ERROR	Internal memory error

Comments

A CSP can create its own set of custom functions. The context information can be passed through its own data structure. The CSSM_CSP_PassThrough function should be used along with the function ID to call the desired custom function.

See Also

CSSM_CSP_PassThrough, CSSM_GetContext, CSSM_SetContext, CSSM_DeleteContext, CSSM_GetContextAttribute, CSSM_UpdateContextAttributes

3.3.12 CSSM_GetContext

CSSM_CONTEXT_PTR CSSMAPI CSSM_GetContext (CSSM_CC_HANDLE CCHandle)

This function retrieves the context information when provided with a context handle.

Parameters

CCHandle (input)
The handle to the context information.

Return Value

The pointer to the CSSM_CONTEXT structure that describes the context associated with the handle CCHandle. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code. Call the CSSM_DeleteContext to free the memory allocated by the CSSM.

Error Codes

Value	Description
CSSM_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_MEMORY_ERROR	Unable to allocate memory

See Also

CSSM_SetContext, CSSM_DeleteContext, CSSM_FreeContext

3.3.13 CSSM_FreeContext

CSSM_RETURN CSSMAPI CSSM_FreeContext(CSSM_CONTEXT_PTR Context)

This function frees the memory associated with the context structure.

Parameters

Context (input)
The pointer to the memory that describes the context structure.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_INVALID_CONTEXT_POINTER	Invalid context pointer

See Also

CSSM_GetContext

3.3.14 CSSM_SetContext

CSSM_RETURN CSSMAPI CSSM_SetContext (CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context)

This function replaces the context information associated with an existing context handle with the new context information supplied in *Context*. Before replacing the context, this function queries the provider associated with the context, to make sure the services requested from it are available in the provider.

Parameters

CCHandle (input)
The handle to the context.

Context (input)
The context data describing the service to replace the current service associated with context handle CCHandle

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_INVALID_CONTEXT_POINTER	Invalid context pointer

See Also

CSSMGetContext

3.3.15 CSSM_DeleteContext

CSSM_RETURN CSSMAPI CSSM_DeleteContext (CSSM_CC_HANDLE CCHandle)

This function frees the context structure allocated by the CSSM_GetContext.

Parameters

CCHandle (input)
The handle that describes a context to be deleted.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_INVALID_CONTEXT_HANDLE	Invalid context handle

See Also

CSSM_GetContext

3.3.16 CSSM_GetContextAttributes

CSSM_CONTEXT_ATTRIBUTE_PTR CSSMAPI CSSM_GetContextAttributes
(CSSM_CC_HANDLE CCHandle,
uint32 AttributeType)

This function retrieves the context attributes information for the given context handle and attribute type.

Parameters

CCHandle (input)
The handle to the context.

AttributeType (input)
The attribute type of the given CCHandle.

Return Value

The pointer to the CSSM_ATTRIBUTE structure that describes the context attributes associated with the handle CCHandle and the attribute type. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code. Call the CSSM_DeleteContextAttributes to free memory allocated by the CSSM.

Error Codes

Value	Description
CSSM_INVALID_CONTEXT_HANDLE	Invalid context handle

See Also

CSSM_DeleteContextAttributes, CSSMGetContext

3.3.17 CSSM_UpdateContextAttributes

CSSM_RETURN CSSMAPI CSSM_UpdateContextAttributes
(CSSM_CC_HANDLE CCHandle,
uint32 NumberAttributes,
const CSSM_CONTEXT_ATTRIBUTE_PTR ContextAttributes)

This function updates the security context. When an attribute is already present in the context, this update operation replaces the previously-defined attribute with the current attribute.

Parameters

CCHandle (input)
The handle to the context.

NumberAttributes (input)
The number of CSSM_CONTEXT_ATTRIBUTE structures to allocate.

ContextAttributes (input)
Pointer to data that describes the attributes to be associated with this context.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_INVALID_POINTER	Invalid pointer to attributes

See Also

CSSM_GetContextAttribute, CSSM_DeleteContextAttributes

3.3.18 CSSM_DeleteContextAttributes

CSSM_RETURN CSSMAPI CSSM_DeleteContextAttributes
(CSSM_CC_HANDLE CCHandle,
CSSM_CONTEXT_ATTRIBUTE_PTR ContextAttributes)

This function deletes internal data associated with given attribute type of the context handle.

Parameters

hContext (input)
The handle that describes a context that is to be deleted.

AttributeType (input)
The attribute to be deleted from the context.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_INVALID_POINTER	Invalid pointer to attributes

See Also

CSSM_GetContextAttributes, CSSM_UpdateContextAttributes

3.4 Cryptographic Sessions and Logon

3.4.1 CSSM_CSP_Login

CSSM_RETURN CSSMAPI CSSM_CSP_Login (CSSM_CSP_HANDLE CSPHandle,
const CSSM_CRYPTO_DATA_PTR Password,
const CSSM_DATA_PTR pReserved)

Logs the user into the CSP, allowing for multiple login types and parallel operation notification.

Parameters

CSPHandle (input)
Handle of the CSP to log into.

Password (input)
Password used to log into the token.

PReserved(input)
This field is reserved for future use. The value NULL should always be given. (May be used for multiple user support in the future.)

Return Value

CSSM_OK if login is successful, CSSM_FAIL is login fails. Use CSSM_GetError to determine the exact error.

Error Codes

Value	Description
CSSM_CSP_INVALID_CSP_HANDLE	Invalid CSP handle
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_INVALID_PASSWORD	Invalid password
CSSM_CSP_ALREADY_LOGGED_IN	User attempted to log in more than once

See Also

CSSM_CSP_ChangeLoginPassword, CSSM_CSP_Logout

3.4.2 CSSM_CSP_Logout

CSSM_RETURN CSSMAPI CSSM_CSP_Logout(CSSM_CSP_HANDLE CSPHandle)

Terminates the login session associated with the specified CSP Handle.

Parameters

CSPHandle (input)
Handle for the target CSP.

Return Value

CSSM_OK if successful, CSSM_FAIL if an error occured. Use CSSM_GetError to determine the exact error.

Error Codes

Value	Description
CSSM_CSP_INVALID_CSP	Invalid CSP handle
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_NOT_LOGGED_IN	No login session existed

See Also

CSSM_CSP_Login, CSSM_CSP_ChangeLoginPassword

3.4.3 CSSM_CSP_ChangeLoginPassword

CSSM_RETURN CSSMAPI CSSM_CSP_ChangeLoginPassword
(CSSM_CSP_HANDLE CSPHandle,
const CSSM_CRYPTO_DATA_PTR OldPassword,
const CSSM_CRYPTO_DATA_PTR NewPassword)

Changes the login password of the current login session from the old password to the new password. The requesting user must have a login session in process.

Parameters

CSPHandle (input)
Handle of the CSP supporting the current login session.

OldPassword (input)
Current password used to log into the token.

NewPassword(input)
New password to be used for future logins by this user to this token.

Return Value

CSSM_OK if login is successful, CSSM_FAIL is login fails. Use CSSM_GetError to determine the exact error.

Error Codes

Value	Description
CSSM_CSP_INVALID_CSP_HANDLE	Invalid CSP handle
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_INVALID_PASSWORD	Old password is invalid

See Also

CSSM_CSP_Login, CSSM_CSP_Logout

3.5 Cryptographic Operations

3.5.1 CSSM_QuerySize

CSSM_RETURN CSSMAPI CSSM_QuerySize (CSSM_CC_HANDLE CCHandle,
uint32 SizeOfInput,
uint32 * ReqSizeOutBlock)

This function queries for the size of the output data for Signature, Message Digest, and Message Authentication Code context types and queries for the algorithm block size or the size of the output data for encryption and decryption context types. This function can also be used to query the output size requirements for the intermediate steps of a staged cryptographic operation (for example, **CSSM_EncryptDataUpdate** and **CSSM_DecryptDataUpdate**). There may be algorithm-specific and token-specific rules restricting the lengths of data following data update calls.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

SizeOfInput (input)
This parameter currently applies only to encrypt and decrypt context types. If this parameter is 0, the function returns the algorithm block size. Otherwise, the size of the output data is returned.

ReqSizeOutBlock (output)
Pointer to a uint32 variable where the function returns the size of the output in bytes.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_NO_METHOD	Service not provided
CSSM_CSP_QUERY_SIZE_FAILED	Unable to query size

See Also

CSSM_EncryptData, CSSM_EncryptDataUpdate, CSSM_DecryptData, CSSM_DecryptDataUpdate, CSSM_SignData, CSSM_VerifyData, CSSM_DigestData, CSSM_GenerateMac

3.5.2 CSSM_SignData

CSSM_RETURN CSSMAPI CSSM_SignData (CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR DataBufs,
uint32 DataBufCount,
CSSM_DATA_PTR Signature)

This function signs data using the private key associated with the public key specified in the context.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

DataBufs (input)
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

DataBufCount (input)
The number of *DataBufs* to be signed.

Signature (output)
A pointer to the CSSM_DATA structure for the signature.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer
CSSM_CSP_INVALID_DATA_COUNT	Invalid data count
CSSM_CSP_SIGN_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_SIGN_NO_METHOD	Service not provided
CSSM_CSP_SIGN_FAILED	Sign failed
CSSM_CSP_PRIKEY_NOT_FOUND	Cannot find the corresponding private key
CSSM_CSP_PASSWORD_INCORRECT	Password incorrect
CSSM_CSP_UNWRAP_FAILED	Unwrapped the private key failed
CSSM_CSP_NOT_ENOUGH_BUFFER	The output buffer is not big enough
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_VECTOROFBUFS_UNSUPPORTED	Supports only a single buffer of input

Comments

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space; application has to free the memory in this case. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

See Also

CSSM_VerifyData, CSSM_SignDataInit, CSSM_SignDataUpdate, CSSM_SignDataFinal

3.5.3 CSSM_SignDataInit

CSSM_RETURN CSSMAPI CSSM_SignDataInit (CSSM_CC_HANDLE CCHandle)

This function initializes the staged sign data function.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_SIGN_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_SIGN_NO_METHOD	Service not provided
CSSM_CSP_SIGN_INIT_FAILED	Staged sign initialize function failed
CSSM_CSP_STAGED_OPERATION_UNSUPPORTED	Supports only single stage operations

See Also

CSSM_SignData, CSSM_SignDataUpdate, CSSM_SignDataFinal

3.5.4 CSSM_SignDataUpdate

CSSM_RETURN CSSMAPI CSSM_SignDataUpdate (CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR DataBufs,
uint32 DataBufCount)

This function updates the data for the staged sign data function.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

DataBufs (input)
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

DataBufCount (input)
The number of *DataBufs* to be signed.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer
CSSM_CSP_INVALID_DATA_COUNT	Invalid data count
CSSM_CSP_SIGN_UPDATE_FAILED	Staged sign update function failed
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_STAGED_OPERATION_UNSUPPORTED	Supports only single stage operations

See Also

CSSM_SignData, CSSM_SignDataInit, CSSM_SignDataFinal

3.5.5 CSSM_SignDataFinal

CSSM_RETURN CSSMAPI CSSM_SignDataFinal (CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR Signature)

This function completes the final stage of the sign data function.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Signature (output)
A pointer to the CSSM_DATA structure for the signature.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_SIGN_FINAL_FAILED	Staged sign final function failed
CSSM_NOT_ENOUGH_BUFFER	The output buffer is not big enough
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_STAGED_OPERATION_UNSUPPORTED	Supports only single stage operations

Comments

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space; application has to free the memory in this case. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

See Also

CSSM_SignData, CSSM_SignDataInit, CSSM_SignDataUpdate

3.5.6 CSSM_VerifyData

CSSM_BOOL CSSMAPI CSSM_VerifyData (CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR DataBufs,
uint32 DataBufCount,
const CSSM_DATA_PTR Signature)

This function verifies the input data against the provided signature.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

DataBufs (input)
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

DataBufCount (input)
The number of *DataBufs* to be verified.

Signature (input)
A pointer to a CSSM_DATA structure which contains the signature and the size of the signature.

Return Value

A CSSM_TRUE return value signifies the signature was successfully verified. When CSSM_FALSE is returned, either the signature was not successfully verified or an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer
CSSM_CSP_INVALID_DATA_COUNT	Invalid data count
CSSM_CSP_VERIFY_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_VERIFY_NO_METHOD	Service not provided
CSSM_CSP_VERIFY_FAILED	Unable to perform verification on data
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_VECTOROFBUFS_UNSUPPORTED	Supports only a single buffer of input

See Also

CSSM_SignData, CSSM_VerifyDataInit, CSSM_VerifyDataUpdate, CSSM_VerifyDataFinal

3.5.7 CSSM_VerifyDataInit

CSSM_RETURN CSSMAPI CSSM_VerifyDataInit (CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR Signature)

This function initializes the staged verify data function.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Signature (input)
A pointer to a CSSM_DATA structure which contains the starting address for the signature to verify against and the length of the signature in bytes.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer
CSSM_CSP_VERIFY_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_VERIFY_NO_METHOD	Service not provided
CSSM_CSP_VERIFY_INIT_FAILED	Staged verify initialize function failed
CSSM_CSP_STAGED_OPERATION_UNSUPPORTED	Supports only single stage operations

See Also

CSSM_VerifyDataUpdate, CSSM_VerifyDataFinal, CSSM_VerifyData

3.5.8 CSSM_VerifyDataUpdate

CSSM_RETURN CSSMAPI CSSM_VerifyDataUpdate (CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR DataBufs,
uint32 DataBufCount)

This function updates the data to the staged verify data function.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

DataBufs (input)
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

DataBufCount (input)
The number of *DataBufs* to be verified.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer
CSSM_CSP_INVALID_DATA_COUNT	Invalid data count
CSSM_CSP_VERIFY_UPDATE_FAILED	Staged verify update function failed
CSSM_CSP_STAGED_OPERATION_UNSUPPORTED	Supports only single stage operations

See Also

CSSM_VerifyData, CSSM_VerifyDataInit, CSSM_VerifyDataFinal

3.5.9 CSSM_VerifyDataFinal

CSSM_BOOL CSSMAPI CSSM_VerifyDataFinal (CSSM_CC_HANDLE CCHandle)

This function finalizes the staged verify data function.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Return Value

A CSSM_TRUE return value signifies the signature successfully verified. When CSSM_FALSE is returned, either the signature was not successfully verified or an error has occurred; use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_POINTER	Invalid context pointer
CSSM_CSP_VERIFY_FINAL_FAILED	Staged verify final function failed
CSSM_CSP_STAGED_OPERATION_UNSUPPORTED	Supports only single stage operations

See Also

CSSM_VerifyData, CSSM_VerifyDataInit, CSSM_VerifyDataUpdate

3.5.10 CSSM_DigestData

CSSM_RETURN CSSMAPI CSSM_DigestData (CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR DataBufs,
uint32 DataBufCount,
CSSM_DATA_PTR Digest)

This function computes a message digest for the supplied data.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

DataBufs (input)
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

DataBufCount (input)
The number of *DataBufs*.

Digest (output)
A pointer to the CSSM_DATA structure for the message digest.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer
CSSM_CSP_INVALID_DATA_COUNT	Invalid data count
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_DIGEST_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_DIGEST_NO_METHOD	Service not provided
CSSM_CSP_DIGEST_FAILED	Unable to perform digest on data
CSSM_CSP_VECTOROFBUFS_UNSUPPORTED	Supports only a single buffer of input

Comments

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space; application has to free the memory in this case. If the output buffer pointer this is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

See Also

CSSM_DigestDataInit, CSSM_DigestDataUpdate, CSSM_DigestDataFinal

3.5.11 CSSM_DigestDataInit

CSSM_RETURN CSSMAPI CSSM_DigestDataInit (CSSM_CC_HANDLE CCHandle)

This function initializes the staged message digest function.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_DIGEST_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_DIGEST_NO_METHOD	Service not provided
CSSM_CSP_DIGEST_INIT_FAILED	Unable to perform digest initialization
CSSM_CSP_STAGED_OPERATION_UNSUPPORTED	Supports only single stage operations

See Also

CSSM_DigestData, CSSM_DigestDataUpdate, CSSM_DigestDataClone, CSSM_DigestDataFinal

3.5.12 CSSM_DigestDataUpdate

CSSM_RETURN CSSMAPI CSSM_DigestDataUpdate (CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR DataBufs,
uint32 DataBufCount)

This function updates the staged message digest function.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

DataBufs (input)
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

DataBufCount (input)
The number of *DataBufs*.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer
CSSM_CSP_INVALID_DATA_COUNT	Invalid data count
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_DIGEST_UPDATE_FAILED	Unable to perform digest on data
CSSM_CSP_STAGED_OPERATION_UNSUPPORTED	Supports only single stage operations

See Also

CSSM_DigestData, CSSM_DigestDataInit, CSSM_DigestDataClone, CSSM_DigestDataFinal

3.5.13 CSSM_DigestDataClone

CSSM_CC_HANDLE CSSMAPI CSSM_DigestDataClone (CSSM_CC_HANDLE CCHandle)

This function clones a given staged message digest context with its cryptographic attributes and intermediate result.

Parameters

CCHandle (input)
The handle that describes the context of a staged message digest operation.

Return Value

The pointer to a user-allocated CSSM_CC_HANDLE for holding the cloned context handle return from CSSM. If the pointer is NULL, an error has occured; use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_DIGEST_CLONE_FAILED	Unable to clone the digest context

Comments

When a digest context is cloned, a new context is created with data associated with the parent context. Changes made to the parent context after calling this function will not be reflected in the cloned context. The cloned context could be used with the CSSM_DigestDataUpdate and CSSM_DigestDataFinal functions.

See Also

CSSM_DigestData, CSSM_DigestDataInit, CSSM_DigestDataUpdate, CSSM_DigestDataFinal

3.5.14 CSSM_DigestDataFinal

CSSM_RETURN CSSMAPI CSSM_DigestDataFinal (CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR Digest)

This function finalizes the staged message digest function.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Digest (output)
A pointer to the CSSM_DATA structure for the message digest.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_DIGEST_FINAL_FAILED	Staged digest final failed
CSSM_CSP_STAGED_OPERATION_UNSUPPORTED	Supports only single stage operations

Comments

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space; application has to free the memory in this case. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

See Also

CSSM_DigestData, CSSM_DigestDataInit, CSSM_DigestDataUpdate, CSSM_DigestDataClone

3.5.15 CSSM_GenerateMac

CSSM_RETURN CSSMAPI CSSM_GenerateMac (CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR DataBufs,
uint32 DataBufCount,
CSSM_DATA_PTR Mac)

This function generates a message authentication code for the supplied data.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

DataBufs (input)
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

DataBufCount (input)
The number of *DataBufs*.

Mac (output)
A pointer to the CSSM_DATA structure for the message authentication code.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer
CSSM_CSP_INVALID_DATA_COUNT	Invalid data count
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_MAC_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_MAC_NO_METHOD	Service not provided
CSSM_CSP_MAC_FAILED	Unable to perform mac on data
CSSM_CSP_VECTOROFBUFS_UNSUPPORTED	Supports only a single buffer of input

Comments

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space; application has to free the memory in this case. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

See Also

CSSM_GenerateMacInit, CSSM_GenerateMacUpdate, CSSM_GenerateMacFinal

3.5.16 CSSM_GenerateMacInit

CSSM_RETURN CSSMAPI CSSM_GenerateMacInit (CSSM_CC_HANDLE CCHandle)

This function initializes the staged message authentication code function.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_MAC_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_MAC_NO_METHOD	Service not provided
CSSM_CSP_MAC_INIT_FAILED	Unable to perform staged mac init
CSSM_CSP_STAGED_OPERATION_UNSUPPORTED	Supports only single stage operations

See Also

CSSM_GenerateMac, CSSM_GenerateMacUpdate, CSSM_GenerateMacFinal

3.5.17 CSSM_GenerateMacUpdate

CSSM_RETURN CSSMAPI CSSM_GenerateMacUpdate (CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR DataBufs,
uint32 DataBufCount)

This function updates the staged message authentication code function.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

DataBufs (input)
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

DataBufCount (input)
The number of *DataBufs*.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer
CSSM_CSP_INVALID_DATA_COUNT	Invalid data count
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_MAC_UPDATE_FAILED	Unable to perform staged mac update
CSSM_CSP_STAGED_OPERATION_UNSUPPORTED	Supports only single stage operations

See Also

CSSM_GenerateMac, CSSM_GenerateMacInit, CSSM_GenerateMacFinal

3.5.18 CSSM_GenerateMacFinal

CSSM_RETURN CSSMAPI CSSM_GenerateMacFinal (CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR Mac)

This function finalizes the staged message authentication code function.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Mac (output)
A pointer to the CSSM_DATA structure for the message authentication code.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CSP_HANDLE	Invalid CSP handle
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_MAC_FINAL_FAILED	Unable to perform staged mac final
CSSM_CSP_STAGED_OPERATION_UNSUPPORTED	Supports only single stage operations

Comments

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space; application has to free the memory in this case. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

See Also

CSSM_GenerateMac, CSSM_GenerateMacInit, CSSM_GenerateMacUpdate

3.5.19 CSSM_EncryptData

CSSM_RETURN CSSMAPI CSSM_EncryptData (CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR ClearBufs,
uint32 ClearBufCount,
CSSM_DATA_PTR CipherBufs,
uint32 CipherBufCount,
uint32 *bytesEncrypted,
CSSM_DATA_PTR RemData)

This function encrypts the supplied data using information in the context. The **CSSM_QuerySize** function can be used to estimate the output buffer size required.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

ClearBufs (input)
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

ClearBufCount (input)
The number of *ClearBufs*.

CipherBufs (output)
A pointer to a vector of CSSM_DATA structures that contain the results of the operation on the data.

CipherBufCount (input)
The number of *CipherBufs*.

bytesEncrypted (output)
A pointer to uint32 for the size of the encrypted data in bytes.

RemData (output)
A pointer to the CSSM_DATA structure for the last encrypted block containing padded data.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer
CSSM_CSP_INVALID_DATA_COUNT	Invalid data count
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_ENC_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_ENC_NO_METHOD	Service not provided
CSSM_CSP_ENC _FAILED	Unable to encrypt data
CSSM_CSP_ENC_BAD_IV_LENGTH	Length of IV unsupported
CSSM_CSP_ENC_BAD_KEY_LENGTH	Length of key unsupported

Comments

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space; application has to free the memory in this case. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned. In-place encryption can be done by supplying the same input and output buffers.

See Also

CSSM_QuerySize, CSSM_DecryptData, CSSM_EncryptDataInit, CSSM_EncryptDataUpdate, CSSM_EncryptDataFinal

3.5.20 CSSM_EncryptDataInit

CSSM_RETURN CSSMAPI CSSM_EncryptDataInit (CSSM_CC_HANDLE CCHandle)

This function initializes the staged encrypt function. There may be algorithm-specific and token-specific rules restricting the lengths of data following data update calls making use of these parameters.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_ENC_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_ENC_NO_METHOD	Service not provided
CSSM_CSP_ENC_INIT_FAILED	Unable to perform encrypt initialization
CSSM_CSP_ENC_BAD_IV_LENGTH	Length of IV unsupported
CSSM_CSP_ENC_BAD_KEY_LENGTH	Length of key unsupported
CSSM_CSP_STAGED_OPERATION_UNSUPPORTED	Supports only single stage operations

See Also

CSSM_EncryptData, CSSM_EncryptDataUpdate, CSSM_EncryptDataFinal

3.5.21 CSSM_EncryptDataUpdate

CSSM_RETURN CSSMAPI CSSM_EncryptDataUpdate
(CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR ClearBufs,
uint32 ClearBufCount,
CSSM_DATA_PTR CipherBufs,
uint32 CipherBufCount,
uint32 *bytesEncrypted)

This function updates the staged encrypt function. The **CSSM_QuerySize** function can be used to estimate the output buffer size required for each update call. There may be algorithm-specific and token-specific rules restricting the lengths of data in **CSSM_EncryptUpdate** calls.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

ClearBufs (input)
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

ClearBufCount (input)
The number of *ClearBufs*.

CipherBufs (output)
A pointer to a vector of CSSM_DATA structures that contain the encrypted data resulting from the encryption operation.

CipherBufCount (input)
The number of *CipherBufs*.

bytesEncrypted (output)
A pointer to uint32 for the size of the encrypted data in bytes.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_DATA_COUNT	Invalid data count
CSSM_CSP_ENC _FAILED	Unable to encrypt data
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_STAGED_OPERATION_UNSUPPORTED	Supports only single stage operations

Comments

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space; application has to free the memory in this case. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned. In-place encryption can be done by supplying the same input and output buffer.

See Also

CSSM_EncryptData, CSSM_EncryptDataInit, CSSM_EncryptDataFinal, CSSM_QuerySize

3.5.22 CSSM_EncryptDataFinal

CSSM_RETURN CSSMAPI CSSM_EncryptDataFinal (CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR RemData)

This function finalizes the staged encrypt function.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

RemData (output)
A pointer to the CSSM_DATA structure for the last encrypted block containing padded data.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_ENC_FINAL_FAILED	Unable to encrypt data
CSSM_CSP_STAGED_OPERATION_UNSUPPORTED	Supports only single stage operations

Comments

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space; application has to free the memory in this case. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned. In-place encryption can be done by supplying the same input and output buffers.

See Also

CSSM_EncryptData, CSSM_EncryptDataInit, CSSM_EncryptDataUpdate

3.5.23 CSSM_DecryptData

CSSM_RETURN CSSMAPI CSSM_DecryptData (CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR CipherBufs,
uint32 CipherBufCount,
CSSM_DATA_PTR ClearBufs,
uint32 ClearBufCount,
uint32 *bytesDecrypted,
CSSM_DATA_PTR RemData)

This function decrypts the supplied encrypted data. The **CSSM_QuerySize** function can be used to estimate the output buffer size required.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

CipherBufs (input)
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

CipherBufCount (input)
The number of *CipherBufs*.

ClearBufs (output)
A pointer to a vector of CSSM_DATA structures that contain the decrypted data resulting from the decryption operation.

ClearBufCount (input)
The number of *ClearBufs*.

BytesDecrypted (output)
A pointer to uint32 for the size of the decrypted data in bytes.

RemData (output)
A pointer to the CSSM_DATA structure for the last decrypted block.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer
CSSM_CSP_INVALID_DATA_COUNT	Invalid data count
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_DEC_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_DEC_NO_METHOD	Service not provided
CSSM_CSP_DEC _FAILED	Unable to encrypt data
CSSM_CSP_DEC _BAD_IV_LENGTH	Length of IV unsupported
CSSM_CSP_DEC _BAD_KEY_LENGTH	Length of key unsupported

Comments

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space; application has to free the memory in this case. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned. In-place decryption can be done by supplying the same input and output buffer.

See Also

CSSM_QuerySize, CSSM_EncryptData, CSSM_DecryptDataInit, CSSM_DecryptDataUpdate, CSSM_DecryptDataFinal

3.5.24 CSSM_DecryptDataInit

CSSM_RETURN CSSMAPI CSSM_CSSM_DecryptDataInit (CSSM_CC_HANDLE CCHandle)

This function initializes the staged decrypt function.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_DEC_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_DEC_NO_METHOD	Service not provided
CSSM_CSP_DEC_INIT_FAILED	Unable to perform decrypt initialization
CSSM_CSP_DEC_BAD_IV_LENGTH	Length of IV unsupported
CSSM_CSP_DEC_BAD_KEY_LENGTH	Length of key unsupported
CSSM_CSP_STAGED_OPERATION_UNSUPPORTED	Supports only single stage operations

See Also

CSSM_DecryptData, CSSM_DecryptDataUpdate, CSSM_DecryptDataFinal

3.5.25 CSSM_DecryptDataUpdate

CSSM_RETURN CSSMAPI CSSM_DecryptDataUpdate
(CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR CipherBufs,
uint32 CipherBufCount,
CSSM_DATA_PTR ClearBufs,
uint32 ClearBufCount,
uint32 *bytesDecrypted)

This function updates the staged decrypt function. The **CSSM_QuerySize** function can be used to estimate the output buffer size required for each update call. There may be algorithm-specific and token-specific rules restricting the lengths of data in **CSSM_DecryptUpdate** calls.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

CipherBufs (input)
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

CipherBufCount (input)
The number of *CipherBufs*.

ClearBufs (output)
A pointer to a vector of CSSM_DATA structures that contain the decrypted data resulting from the decryption operation.

ClearBufCount (input)
The number of *ClearBufs*.

bytesDecrypted (output)
A pointer to uint32 for the size of the decrypted data in bytes.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer
CSSM_CSP_INVALID_DATA_COUNT	Invalid data count
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_DEC_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_DEC_NO_METHOD	Service not provided
CSSM_CSP_DEC _UPDATE_FAILED	Staged encryption update failed
CSSM_CSP_STAGED_OPERATION_UNSUPPORTED	Supports only single stage operations

Comments

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space; application has to free the memory in this case. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned. In-place decryption can be done by supplying the same input and output buffers.

See Also

CSSM_DecryptData, CSSM_DecryptDataInit, CSSM_DecryptDataFinal, CSSM_QuerySize

3.5.26 CSSM_DecryptDataFinal

CSSM_RETURN CSSMAPI CSSM_DecryptDataFinal (CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR RemData)

This function finalizes the staged decrypt function.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

RemData (output)
A pointer to the CSSM_DATA structure for the last decrypted block.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_DEC_FINAL_FAILED	Stages encrypt final failed
CSSM_CSP_STAGED_OPERATION_UNSUPPORTED	Supports only single stage operations

Comments

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space; application has to free the memory in this case. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned. In-place decryption can be done by supplying the same input and output buffers.

See Also

CSSM_DecryptData, CSSM_DecryptDataInit, CSSM_DecryptDataUpdate

3.5.27 CSSM_GenerateKey

CSSM_RETURN CSSMAPI CSSM_GenerateKey (CSSM_CC_HANDLE CCHandle,
CSSM_BOOL StoreKey,
CSSM_KEY_PTR Key)

This function generates a symmetric key.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

StoreKey (input)
Boolean flag that indicates whether the symmetric key should be stored in the CSP — this is possible if the CSP allows storage of symmetric keys.

Key (output)
Pointer to CSSM_ KEY structure used to obtain the key.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_KEYGEN_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_KEYGEN_NO_METHOD	Service not provided
CSSM_CSP_KEYGEN_FAILED	Unable to generate key pair

Comments

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space; application has to free the memory in this case. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

See Also

CSSM_GenerateRandom, CSSM_GenerateKeyPair

3.5.28 CSSM_GenerateKeyPair

CSSM_RETURN CSSMAPI CSSM_GenerateKeyPair (CSSM_CC_HANDLE CCHandle,
CSSM_BOOL StorePublicKey,
CSSM_KEY_PTR PublicKey,
CSSM_KEY_PTR PrivateKey)

This function generates an asymmetric key pair.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

StorePublicKey (input)
Boolean flag that indicates whether the public key should be stored in the CSP — this is possible if the CSP allows storage of public keys. It is recommended that CSPs always have the facility for storage of private keys.

PublicKey (output)
Pointer to CSSM_ KEY structure used to obtain the public key.

PrivateKey (output)
Pointer to CSSM_ KEY structure used to obtain the private key.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_KEYGEN_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_KEYGEN_NO_METHOD	Service not provided
CSSM_CSP_KEYGEN_FAILED	Unable to generate key pair

Comments

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space; application has to free the memory in this case. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

See Also

CSSM_GenerateRandom

3.5.29 CSSM_GenerateRandom

CSSM_RETURN CSSMAPI CSSM_GenerateRandom (CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR RandomNumber)

This function generates random data.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

RandomNumber (output)
Pointer to CSSM_DATA structure used to obtain the random number and the size of the random number in bytes.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_RNG_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_RNG_NO_METHOD	Service not provided
CSSM_CSP_RNG_FAILED	Unable to generate keys

Comments

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space; application has to free the memory in this case. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

3.5.30 CSSM_GenerateUniqueId

CSSM_RETURN CSSMAPI CSSM_GenerateUniqueId (CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR UniqueID)

This function generates unique identification code.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

UniqueId (output)
Pointer to CSSM_DATA structure used to obtain the unique ID and the size of the unique ID in bytes.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_UIDG_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_UIDG_NO_METHOD	Service not provided
CSSM_CSP_UIDG_FAILED	Unable to generate unique ID

Comments

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space; application has to free the memory in this case. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

3.5.31 CSSM_WrapKey

CSSM_RETURN CSSMAPI CSSM_WrapKey
(CSSM_CC_HANDLE CCHandle,
const CSSM_CRYPTO_DATA_PTR PassPhrase,
CSSM_KEY_PTR Key,
CSSM_WRAP_KEY_PTR WrappedKey)

This function wraps the supplied key using the context. The key may be a symmetric key or the public key of a public/private key pair. If a symmetric key is specified it is wrapped. If a public key is specified, the passphrase is used to unlock the corresponding private key, which is then wrapped.

Parameters

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

PassPhrase (input)
The passphrase or a callback function to be used to obtain the passphrase that can be used by the CSP to unlock the private key before it is wrapped. This input is ignored when wrapping a symmetric, secret key.

Key (input)
A pointer to the target key to be wrapped. If a private key is to be wrapped, the target key is the public key associated with the private key. If a symmetric key is to be wrapped, the target key is that symmetric key.

WrappedKey (output)
A pointer to a CSSM_KEY structure that returns the wrapped key.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_INVALID_KEY	Invalid wrapping key
CSSM_CSP_PRIKEY_NOT_FOUND	Cannot find the corresponding private key
CSSM_CSP_PASSWORD_INCORRECT	Password incorrect
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_ENC_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_ENC_NO_METHOD	Service not provided
CSSM_INVALID_SUBJECT_KEY	Invalid key to be wrapped
CSSM_CSP_ENC _FAILED	Unable to encrypt data

See Also

CSSM_CSP_CreateWrapContext, CSSM_UnwrapKey

3.5.32 CSSM_UnwrapKey

CSSM_RETURN CSSMAPI CSSM_UnwrapKey
(CSSM_CC_HANDLE CCHandle,
const CSSM_CRYPTO_DATA_PTR NewPassPhrase,
const CSSM_WRAP_KEY_PTR WrappedKey,
CSSM_BOOL StoreKey,
CSSM_KEY_PTR UnwrappedKey)

This function unwraps the data using the context. Depending on the *PersistentObject* mode of the CSP and the *StoreKey* parameter, the unwrapped key can be securely stored by the CSP and locked by the new passphrase.

Parameters

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

PassPhrase (input)
The passphrase or a callback function to be used to obtain the passphrase. If the unwrapped key is a private key and the persistent object mode is true, then the private key is unwrapped and securely stored by the CSP. The *PassPhrase* is used to secure the private key after it is unwrapped. It is assumed that a known public key is associated with the private key.

WrappedKey (input)
A pointer to the wrapped key. The wrapped key may be a symmetric key or the private key of a public/private keypair. The unwrapping method is specified as meta data within the wrapped key and is not specified outside of the wrapped key.

StoreKey (input)
Boolean flag that indicates whether the unwrapped key should be stored in the CSP — this is possible if the CSP allows storage of the particular key type.

UnwrappedKey (output)
A pointer to a CSSM_KEY structure that returns the unwrapped key.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_INVALID_KEY	Invalid unwrapping key
CSSM_INVALID_PASSPHRASE	Invalid passphrase for the unwrapping key or invalid passphrase for securing the unwrapped key in persistent storage
CSSM_INVALID_WRAPPED_KEY	Invalid wrapped key
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_ENC_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_ENC_NO_METHOD	Service not provided
CSSM_CSP_ENC _FAILED	Unable to encrypt data

See Also

CSSM_CSP_CreateUnwrapContext, CSSM_WrapKey

3.5.33 CSSM_DeriveKey

CSSM_RETURN CSSMAPI CSSM_DeriveKey (CSSM_CC_HANDLE CCHandle,
const CSSM_KEY_PTR BaseKey,
CSSM_DATA_PTR Param,
CSSM_BOOL StoreKey,
CSSM_KEY_PTR DerivedKey)

This function derives a new symmetric key using the context and information from the base key.

Parameters

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

BaseKey (input)
The base key used to derive the new key. The base key may be a public key, a private key, or a symmetric key.

Param (input/output)
This parameter varies depending on the derivation mechanism. Password based derivation algorithms use this parameter to return a cipher block chaining initilazation vector. Concatenation algorithms will use this parameter to get the second item to concatenate.

StoreKey (input)
Boolean flag that indicates whether the unwrapped key should be stored in the CSP - this is possible if the CSP allows storage of the particular key type.

DerivedKey (output)
A pointer to a CSSM_KEY structure that returns the derived key.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_NO_METHOD	Service not provided
CSSM_INVALID_KEY	Invalid base key
CSSM_CSP_DERIVE _FAILED	Unable to derive key

See Also

CSSM_CSP_CreateDeriveKeyContext

3.5.34 CSSM_KeyExchGenParam

CSSM_RETURN CSSMAPI CSSM_KeyExchGenParam
(CSSM_CC_HANDLE CCHandle,
uint32 ParamBits,
CSSM_DATA_PTR Param)

This function generates key exchange parameter data for CSSM_KeyExchPhase1.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

ParamBits (input)
Used to generate parameters for the key exchange algorithm (for example, Diffie-Hellman).

Param (output)
Pointer to CSSM_DATA structure used to obtain the key exchange parameter and the size of the key exchange parameter in bytes.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_KEYEXCH_GENPARAM_FAILED	Unable to generate exchange param data

Comments

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space; application has to free the memory in this case. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

See Also

CSSM_KeyExchPhase1, CSSM_KeyExchPhase2

3.5.35 CSSM_KeyExchPhase1

CSSM_RETURN CSSMAPI CSSM_KeyExchPhase1 (CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR Param,
CSSM_DATA_PTR Param1)

Phase 1 of the key exchange operation — generates data for CSSM_KeyExchPhase2.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Param (input)
Param is the return value from the CSSM_KeyExchGenParam function.

Param1 (output)
Pointer to CSSM_DATA structure used to obtain the Phase 1 output.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer
CSSM_CSP_KEYEXCH_PHASE1_FAILED	Unable to generate to stage key exchange
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate

Comments

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space; application has to free the memory in this case. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

See Also

CSSM_KeyExchGenParam, CSSM_KeyExchPhase2

3.5.36 CSSM_KeyExchPhase2

CSSM_RETURN CSSMAPI CSSM_KeyExchPhase2 (CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR Param1,
CSSM_KEY_PTR ExchangedKey)

Phase 2 of the key exchange operation.

Parameters

CCHandle (input)
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Param1 (input)
Param is the return value from the CSSM_KeyExchPhase1 function.

ExchangedKey (output)
Pointer to CSSM_KEY structure used to obtain the exchanged key blob.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_KEYEXCH_PHASE2_FAILED	Unable to stage key exchange

Comments

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space; application has to free the memory in this case. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

See Also

CSSM_KeyExchPhase1, CSSM_KeyExchGenParam

3.6 Module Management Functions

3.6.1 CSSM_CSP_Install

CSSM_RETURN CSSMAPI CSSM_CSP_Install (const char *CSPName,
const char *CSPFileName,
const char *CSPPathName,
const CSSM_GUID_PTR GUID,
const CSSM_CSPINFO_PTR CSPInfo,
const void * Reserved1,
const CSSM_DATA_PTR Reserved2)

This function updates the CSSM-persistent internal information about the CSP module.

Parameters

CSPName (input)
The name of the CSP module.

CSPFileName (input)
The name of the file that implements the CSP.

CSPPathName (input)
The path to the file that implements the CSP.

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

CSPInfo (input)
A pointer to the CSSM_CSPINFO structure containing information about the CSP module.

Reserved1 (input)
Reserve data for the function.

Reserved2 (input)
Reserve data for the function.

Return Value

A CSSM_OK return value signifies that information has been updated. If CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_REGISTRY_ERROR	Error in the registry

See Also

CSSM_CSP_Uninstall

3.6.2 CSSM_CSP_Uninstall

CSSM_RETURN CSSMAPI CSSM_CSP_Uninstall (const CSSM_GUID_PTR GUID)

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

Parameters

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

Return Value

A CSSM_OK return value means the CSP has been successfully uninstalled. If CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_INVALID_ GUID	CSP module was not installed
CSSM_REGISTRY_ERROR	Unable to delete information

See Also

CSSM_CSP_Install

3.6.3 CSSM_CSP_Attach

CSSM_CSP_HANDLE CSSMAPI CSSM_CSP_Attach
(const CSSM_GUID_PTR GUID,
uint32 CheckCompatibleVerMajor,
uint32 CheckCompatibleVerMinor,
const CSSM_API_MEMORY_FUNCS_PTR MemoryFuncs,
uint32 SlotID,
uint32 SessionFlags,
uint32 Application,
const CSSM_NOTIFY_CALLBACK Notification,
const void * Reserved)

This function attaches the CSP module and verifies that the version of the module expected by the application is compatible with the version on the system.

Parameters

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

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

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

MemoryFuncs (input)
A structure containing pointers to the memory routines.

SlotID (input)
Slot ID number of the target hardware token. This value should always be taken from the CSSM_CSPINFO structure to insure that a compatible slot is used. (Software-only implementations can always use zero.)

SessionFlags(input)
Bitmask of default “session” modes. Legal values are defined in the table below.

Application(input/optional)
Nonce passed to the application when its callback is invoked allowing the application to determine the proper context of operation.

Notification (input/optional)
Callback provided by the application that is called by the CSP when one of three things takes place: a parallel operation completes, a token running in serial mode surrenders control to the application or the token is removed (hardware specific).

Reserved (input)
A reserved input.

Valid *SessionFlags* Values

Value	Description
CSSM_CSP_SESSION_SERIAL	Sessions created should be in serial mode
CSSM_CSP_SESSION_EXCLUSIVE	Sessions created should be exclusive
CSSM_CSP_SESSION_READWRITE	Sessions created should be read/write

Return Value

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

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_MEMORY_ERROR	Internal memory error
CSSM_INCOMPATIBLE_VERSION	Incompatible version
CSSM_EXPIRE	Add-in has expired
CSSM_INVALID_ARGS	Invalid argument pointer
CSSM_ATTACH_FAIL	Unable to load CSP module

See Also

CSSM_CSP_Detach

3.6.4 CSSM_CSP_Detach

CSSM_RETURN CSSMAPI CSSM_CSP_Detach (CSSM_CSP_HANDLE CSPHandle)

This function detaches the application from the CSP module.

Parameters

CSPHandle (input)
The handle that describes the CSP module.

Return Value

A CSSM_OK return value signifies that the application has been detached from the CSP module. If CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_ADDIN_HANDLE	Invalid CSP handle

See Also

CSSM_CSP_Attach

3.6.5 CSSM_CSP_ListModules

CSSM_LIST_PTR CSSMAPI CSSM_CSP_ListModules (void)

This function returns a list containing the GUID/name pair for each of the currently-installed CSP modules.

Parameters

None

Return Value

A pointer to the CSSM_LIST structure containing the GUID/name pair for each of the CSP modules. If the pointer is NULL, an error has occurred; use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_NO_ADDIN	No add-ins found
CSSM_MEMORY_ERROR	Error in memory allocation

See Also

CSSM_CSP_GetInfo, CSSM_FreeList

3.6.6 CSSM_CSP_GetInfo

CSSM_CSPINFO_PTR CSSMAPI CSSM_CSP_GetInfo
(const CSSM_GUID_PTR GUID,
CSSM_BOOL CompleteCapabilitiesOnly,
uint32 *NumberOfInfos)

This function returns the information about the CSP module.

Parameters

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

CompleteCapabilitiesOnly (input)
Boolean value indicating which capabilities should be returned. If set to TRUE only completely sepcified capabilities should be returned. If set to false, all capability structures registered for the specified CSP should be returned whether or not those capabilities are completely specified.

NumberOfInfos (output)
The number of CSPinfo structures returned by this execution of this function.

Return Value

A CSSM_CSPINFO_PTR to an array of one or more CSP info structures. There is one structure per logical slot managed by the CSP. Hardware tokens may have multiple physical slots. The CSP info structure provides information on the current state of each occupied slot. A software CSP may define an analogous logical slot concept and provide realtime descriptions of each logical slot. If the specified CSP does not support the slot concept, then a single CSP info structure will be returned and the number of structures reported will be one.

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_MEMORY_ERROR	Internal memory error
CSSM_INVALID_GUID	Unknown GUID

See Also

CSSM_CSP_FreeInfo

3.6.7 CSSM_CSP_FreeInfo

CSSM_RETURN CSSMAPI CSSM_CSP_FreeInfo (CSSM_CSPINFO_PTR CSPInfos,
uint32 NumberOfInfos)

This function frees the memory allocated to hold all of the CSP info structures returned by CSSM_CSP_GetInfo.

Parameters

CSPInfso (input)
A pointer to the array of CSSM_CSPINFO structures to be freed.

numberOfInfos (input)
The number of CSP Info structures to be freed.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_INVALID_CSPINFO_POINTER	Invalid pointer

See Also

CSSM_CSP_GetInfo

3.6.8 CSSM_GetHandleInfo

CSSM_HANDLEINFO_PTR CSSMAPI CSSM_GetHandleInfo(CSSM_HANDLE hModule)

Requests meta-information associated with the specified add-in module. Returned information includes slot ID, event notification pointer, and the application-defined identifier for the calling context used during an event callback.

Parameters

hModule (input)
Handle of the module for which information should be returned.

Return Value

A CSSM_HANDLEINFO_PTR to an info structure containing information about the module referenced by the handle.

Error Codes

Value	Description
CSSM_CSP_INVALID_HANDLE	Invalid add-in handle
CSSM_INVALID_POINTER	Invalid pointer to a handle info structure

See Also

CSSM_NOTIFY_CALLBACK

3.7 Extensibility Functions

The CSSM_CSP_PassThrough function is provided to allow CSP developers to extend the crypto functionality of the CSSM API. Because it is only exposed to CSSM as a function pointer, its name internal to the CSP can be assigned at the discretion of the CSP 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 CSPs to describe common error conditions.

3.7.1 CSSM_CSP_PassThrough

CSSM_RETURN CSSMAPI CSSM_CSP_PassThrough (CSSM_CC_HANDLE CCHandle,
uint32 PassThroughId,
const CSSM_DATA_PTR InData,
CSSM_DATA_PTR OutData)

Parameters

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

PassThroughId (input)
An identifier specifying the custom function to be performed.

InData (input)
A pointer to CSSM_DATA structure containing the input data.

OutData (output)
A pointer to CSSM_DATA structure for the output data.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns an error code if an error has occurred.

Error Codes

Value	Description
CSSM_CSP_INVALID_CSP_HANDLE	Invalid CSP handle
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_CONTEXT_POINTER	Invalid context pointer
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer for input data
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_UNSUPPORTED_OPERATION	Add-in does not support this function
CSSM_CSP_PASS_THROUGH_FAIL	Unable to perform custom function

4. Trust Policy Services API

4.1 Overview

The primary purpose of a Trust Policy (TP) module is to answer the question, *Is this certificate authorized for this action?* Different trust policies define different actions that may be requested by an application. There are also a few basic actions that should be common to every trust policy. These actions are operations on the basic objects used by all trust models. The basic objects common to all trust models are certificates and certificate revocation lists. The basic operations on these objects are sign, verify, and revoke.

A registry and query mechanism is available through the CSSM for TP module descriptions. This information is captured during install and load time. Applications can query against this information to find out more about the add-in trust policy module.

CSSM provides two ways for trust policy module developers to extend CSSM's trust policy API. The first way is for the trust policy module to enforce the use of CSSM_TP_CertVerifyForAction, rather than CSSM_TP_CertVerify. This allows the trust policy module to define a module-specific set of actions that certificates can be authorized to perform. A trust policy module may also choose to implement additional API calls. Applications gain access to those functions using the provided CSSM_TP_PassThrough function.

4.1.1 Trust Policy Operations

CSSM_BOOL CSSMAPI CSSM_TP_CertVerify ( ) accepts as input a certificate. The TP module must determine whether the certificate is trusted.

CSSM_DATA_PTR CSSMAPI CSSM_TP_CertSign ( ) accepts as input a signer's certificate, a second certificate to be signed, and the *scope* of the signing process. The *scope* of a signature may be used to identify which field of the certificate should be signed. A simple example is the case of multiple signature on a certificate. Should signatures be applied to just the certificate, meaning they are signing at the same level, or to the certificate and all currently-existing signatures, as a notary public would do, the TP module is responsible for determining whether the signer's certificate is authorized to perform the signing operation and, if so, to carry out the signing operation.

CSSM_ DATA_PTR CSSMAPI CSSM_TP_CertRevoke ( ) accepts as input a revoker's certificate, a certificate revocation list (CRL), and an optional reason for revoking the certificate. The TP module must determine whether the revoker's certificate is trusted to perform/sign the revocation and if so, to carry out the operation by adding a new revocation record to the CRL.

CSSM_BOOL CSSMAPI CSSM_TP_CrlVerify ( ) accepts as input a certificate revocation list. The TP module determines whether the CRL is trusted. This test may include verifying the correctness of the signature associated with the CRL, determining that the CRL

has not been tampered with, and determining that the agent who signed the CRL was trusted to do so.

CSSM_ DATA_PTR CSSMAPI CSSM_TP_CrlSign ( ) accepts as input a CRL and a signer's certificate. The TP module must determine whether the certificate is trusted to sign the CRL. If so, the TP module should carry out the operation.

CSSM_RETURN CSSMAPI CSSM_TP_ApplyCrlToDb ( ) accepts as input a CRL and a data storage handle. The TP module must determine whether the memory-resident CRL is trusted and should be applied to a persistent database, which could result in designating certificates as revoked.

4.1.2 Extensibility Functions

CSSM_BOOL CSSMAPI CSSM_TP_CertVerifyForAction ( ) accepts as input a certificate and a domain-specific action. The TP module must determine whether or not the certificate is trusted to perform the domain-specific action.

CSSM_RETURN CSSMAPI CSSM_TP_PassThrough ( ) accepts as input an operation ID and an arbitrary set of input parameters. The operation ID may specify any type of operation the TP wishes to export. Such operations may include queries or services specific to the domain represented by the TP module.

4.1.3 CSSM TP Management Functions

CSSM_RETURN CSSMAPI CSSM_TP_Install ( ) accepts as input the name and GUID of the TP module, selected attributes describing the module, and information required by CSSM to dynamically load the module, if its use is requested by an application. CSSM adds the TP module name and attributes to the registry of TP modules.

CSSM_RETURN CSSMAPI CSSM_TP_Uninstall ( ) CSSM removes a specified TP module from the TP module registry.

CSSM_LIST_PTR CSSMAPI CSSM_TP_ListModules ( ) CSSM returns a list of all currently-registered TP modules.

CSSM_TP_HANDLECSSMAPI CSSM_TP_Attach ( ) accepts as input the GUID of a TP module and a major and minor version of the caller. The application is requesting a dynamic load of the specified TP module, or of a TP module compatible with the version specified by the caller.

CSSM_RETURN CSSMAPI CSSM_TP_Detach ( ) the application is requesting the dynamic unload of a specified TP module.

CSSM_TPINFO_PTR CSSMAPI CSSM_TP_GetInfo ( ) CSSM returns the major and minor version number of a specified TP module as it is recorded in the TP module registry.

CSSM_RETURN CSSMAPI CSSM_TP_FreeInfo ( ) accepts as input the pointer to the TP information structure allocated by the CSSM. This function reclaims the memory for use by the operating system.

4.2 Data Structures

```
typedef uint32 CSSM_TP_HANDLE    /* Trust Policy Handle */
typedef uint32 CSSM_TP_ACTION
```

4.2.1 CSSM_TPINFO

This data structure represents the information associated with a TP module.

```
typedef struct cssm_tpinfo{
    uint32 VerMajor;
    uint32 VerMinor;
}CSSM_TPINFO, *CSSM_TPINFO_PTR
```

Definition:

VerMajor - Major version number.

VerMinor - Minor version number.

4.2.2 CSSM_REVOKE_REASON

This data structure represents the reason a certificate is being 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
```

4.3 Trust Policy Operations

4.3.1 CSSM_TP_CertVerify

CSSM_BOOL CSSMAPI CSSM_TP_CertVerify (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR SubjectCert,
const CSSM_DATA_PTR SignerCert,
const CSSM_FIELD_PTR VerifyScope,
uint32 ScopeSize)

This functions calls in to the TP module to determine whether certificate is trusted.

Parameters

TPHandle (input)
The handle that describes the add-in trust policy module used to perform this function.

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

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

DBHandle (input)
The handle that describes the database used to perform this function.

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

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

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

VerifyScope (input)
A pointer to the CSSM_FIELD array containing the tags 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

A CSSM_TRUE return value signifies that the certificate can be trusted. When CSSM_FALSE is returned, either the certificate cannot be trusted or an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_TP_INVALID_TP_HANDLE	Invalid handle
CSSM_TP_INVALID_CL_HANDLE	Invalid handle
CSSM_TP_INVALID_DL_HANDLE	Invalid handle
CSSM_TP_INVALID_DB_HANDLE	Invalid handle
CSSM_TP_INVALID_CC_HANDLE	Invalid handle
CSSM_TP_INVALID_CERTIFICATE	Invalid certificate
CSSM_TP_NOT_SIGNER	Signer certificate is not signer of subject
CSSM_TP_NOT_TRUSTED	Signature can’t be trusted
CSSM_TP_CERT_VERIFY_FAIL	Unable to verify certificate
CSSM_FUNCTION_NOT_IMPLEMENTED	Function not implemented

See Also

CSSM_TP_CertSign

4.3.2 CSSM_TP_CertSign

CSSM_DATA_PTR CSSMAPI CSSM_TP_CertSign (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR SubjectCert,
const CSSM_DATA_PTR SignerCert,
const CSSM_FIELD_PTR SignScope,
uint32 ScopeSize)

This function signs a certificate when given a signer's certificate and the *scope* of the signing process. The TP module must decide whether the signer certificate is trusted to sign the subject certificate.

Parameters

TPHandle (input)
The handle that describes the add-in trust policy module used to perform this function.

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

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

DBHandle (input)
The handle that describes the database used to perform this function.

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

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

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

SignScope (input)
A pointer to the CSSM_FIELD array containing the tags 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_TP_INVALID_CERTIFICATE	Invalid certificate
CSSM_TP_CERTIFICATE_CANT_OPERATE	Signer certificate can't sign subject
CSSM_TP_MEMORY_ERROR	Error in allocating memory
CSSM_TP_CERT_SIGN_FAIL	Unable to sign certificate
CSSM_TP_INVALID_TP_HANDLE	Invalid handle
CSSM_TP_INVALID_CL_HANDLE	Invalid handle
CSSM_TP_INVALID_DL_HANDLE	Invalid handle
CSSM_TP_INVALID_DB_HANDLE	Invalid handle
CSSM_TP_INVALID_CC_HANDLE	Invalid handle
CSSM_FUNCTION_NOT_IMPLEMENTED	Function not implemented

See Also

CSSM_TP_CertVerify

4.3.3 CSSM_TP_CertRevoke

CSSM_DATA_PTR CSSMAPI CSSM_TP_CertRevoke (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR OldCrl,
const CSSM_DATA_PTR SubjectCert,
const CSSM_DATA_PTR RevokerCert,
CSSM_REVOKE_REASON Reason)

This function updates a certificate revocation list. The TP module determines whether the revoking certificate can revoke the subject certificate.

Parameters

TPHandle (input)
The handle that describes the add-in trust policy module used to perform this function.

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

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

DBHandle (input)
The handle that describes the database used to perform this function.

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

OldCrl (input)
A pointer to the CSSM_DATA structure containing an existing certificate revocation list. If this input is NULL, a new list is created.

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

RevokerCert (input)
A pointer to the CSSM_DATA structure containing the certificate used to revoke the subject certificate.

Reason (input)
The reason for revoking the subject certificate.

Return Value

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

Error Codes

Value	Description
CSSM_TP_INVALID_CRL	Invalid CRL
CSSM_TP_INVALID_CERTIFICATE	Invalid certificate
CSSM_TP_CERTIFICATE_CANT_OPERATE	Revoker certificate can’t revoke subject
CSSM_TP_MEMORY_ERROR	Error in allocating memory
CSSM_TP_CERT_REVOKE_FAIL	Unable to revoke certificate
CSSM_TP_INVALID_TP_HANDLE	Invalid handle
CSSM_TP_INVALID_CL_HANDLE	Invalid handle
CSSM_TP_INVALID_DL_HANDLE	Invalid handle
CSSM_TP_INVALID_DB_HANDLE	Invalid handle
CSSM_TP_INVALID_CC_HANDLE	Invalid handle
CSSM_FUNCTION_NOT_IMPLEMENTED	Function not implemented

4.3.4 CSSM_TP_CrlVerify

CSSM_BOOL CSSMAPI CSSM_TP_CrlVerify (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR SubjectCrl,
const CSSM_DATA_PTR SignerCert,
const CSSM_FIELD_PTR VerifyScope,
uint32 ScopeSize)

This functions calls into the TP module to determine whether the certificate revocation list is trusted.

Parameters

TPHandle (input)
The handle that describes the add-in trust policy module used to perform this function.

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

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

DBHandle (input)
The handle that describes the database used to perform this function.

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

SubjectCrl (input)
A pointer to the CSSM_DATA structure containing the certificate revocation list.

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

VerifyScope (input)
A pointer to the CSSM_FIELD array containing the tags of the fields to be verified. A null input verifies all the fields in the certificate revocation list.

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

Return Value

A CSSM_TRUE return value signifies that the certificate revocation list can be trusted. When CSSM_FALSE is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_TP_INVALID_CERTIFICATE	Invalid certificate
CSSM_TP_NOT_SIGNER	Signer certificate is not signer of CRL
CSSM_TP_NOT_TRUSTED	Certificate revocation list can’t be trusted
CSSM_TP_CRL_VERIFY_FAIL	Unable to verify certificate
CSSM_TP_INVALID_TP_HANDLE	Invalid handle
CSSM_TP_INVALID_CL_HANDLE	Invalid handle
CSSM_TP_INVALID_DL_HANDLE	Invalid handle
CSSM_TP_INVALID_DB_HANDLE	Invalid handle
CSSM_TP_INVALID_CC_HANDLE	Invalid handle
CSSM_FUNCTION_NOT_IMPLEMENTED	Function not implemented

4.3.5 CSSM_TP_CrlSign

CSSM_DATA_PTR CSSMAPI CSSM_TP_CrlSign (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR SubjectCrl,
const CSSM_DATA_PTR SignerCert,
const CSSM_FIELD_PTR SignScope,
uint32 ScopeSize)

This function signs a certificate revocation list. The TP module must decide whether the signer certificate is trusted to sign the subject certificate revocation list.

Parameters

TPHandle (input)
The handle that describes the add-in trust policy module used to perform this function.

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

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

DBHandle (input)
The handle that describes the database used to perform this function.

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

SubjectCrl (input)
A pointer to the CSSM_DATA structure containing the certificate revocation list.

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

SignScope (input)
A pointer to the CSSM_FIELD array containing the tags of the fields to be signed. A null input signs all the fields in the certificate revocation list.

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

Return Value

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

Error Codes

Value	Description
CSSM_TP_INVALID_CERTIFICATE	Invalid certificate
CSSM_TP_CERTIFICATE_CANT_OPERATE	Signer certificate can’t sign certificate revocation list
CSSM_TP_MEMORY_ERROR	Error in allocating memory
CSSM_TP_CRL_SIGN_FAIL	Unable to sign certificate revocation list
CSSM_TP_INVALID_TP_HANDLE	Invalid handle
CSSM_TP_INVALID_CL_HANDLE	Invalid handle
CSSM_TP_INVALID_DL_HANDLE	Invalid handle
CSSM_TP_INVALID_DB_HANDLE	Invalid handle
CSSM_TP_INVALID_CC_HANDLE	Invalid handle
CSSM_FUNCTION_NOT_IMPLEMENTED	Function not implemented

4.3.6 CSSM_TP_ApplyCrlToDb

CSSM_RETURN CSSMAPI CSSM_TP_ApplyCrlToDb (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
const CSSM_DATA_PTR Crl)

This function updates persistent storage to reflect entries in the certificate revocation list. The TP module determines whether the memory-resident CRL is trusted, and if it should be applied to a persistent database. This results in designating persistent certificates as revoked.

Parameters

TPHandle (input)
The handle that describes the add-in trust policy module used to perform this function.

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

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

DBHandle (input)
The handle that describes the database used to perform this function.

Crl (input)
A pointer to the CSSM_DATA structure containing the certificate revocation list.

Return Value

A CSSM_TRUE return value signifies that the certificate revocation list has been used to update the revocation status of certificates in the specified database. When CSSM_FALSE is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_TP_INVALID_CRL	Invalid certificate revocation list
CSSM_TP_NOT_TRUSTED	certificate revocation list can't be trusted
CSSM_TP_APPLY_CRL_TO_DB_FAIL	Unable to apply certificate revocation list on database
CSSM_TP_INVALID_TP_HANDLE	Invalid handle
CSSM_TP_INVALID_CL_HANDLE	Invalid handle
CSSM_TP_INVALID_DL_HANDLE	Invalid handle
CSSM_TP_INVALID_DB_HANDLE	Invalid handle
CSSM_FUNCTION_NOT_IMPLEMENTED	Function not implemented

See Also

CSSM_CL_CrlGetFirstItem, CSSM_CL_CrlGetNextItem, CSSM_DL_CertRevoke

4.4 Extensibility Functions

4.4.1 CSSM_TP_VerifyAction

CSSM_BOOL CSSMAPI CSSM_TP_VerifyAction (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_CC_HANDLE CCHandle,
CSSM_TP_ACTION Action,
const CSSM_DATA_PTR Data,
const CSSM_DATA_PTR Cert)

This function queries the TP module to determine whether the input certificate is trusted to perform the module-specific action.

Parameters

TPHandle (input)
The handle that describes the add-in trust policy module used to perform this function.

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

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

DBHandle (input)
The handle that describes the database used to perform this function.

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

Action (input)
An action to be performed using the input certificate.

Data (input)
A pointer to the CSSM_DATA structure containing the module-specific data to perform the requested action.

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

Return Value

A CSSM_TRUE return value signifies that certificate can be trusted. When CSSM_FALSE is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_TP_INVALID_TP_HANDLE	Invalid handle
CSSM_TP_INVALID_CL_HANDLE	Invalid handle
CSSM_TP_INVALID_DL_HANDLE	Invalid handle
CSSM_TP_INVALID_DB_HANDLE	Invalid handle
CSSM_TP_INVALID_CC_HANDLE	Invalid handle
CSSM_TP_INVALID_CERTIFICATE	Invalid certificate
CSSM_TP_INVALID_ACTION	Invalid action
CSSM_TP_NOT_TRUSTED	Certificate not trusted for action
CSSM_TP_VERIFY_ACTION_FAIL	Unable to determine trust for action
CSSM_FUNCTION_NOT_IMPLEMENTED	Function not implemented

4.4.2 CSSM_TP_PassThrough

CSSM_DATA_PTR CSSMAPI CSSM_TP_PassThrough (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_CC_HANDLE CCHandle,
uint32 PassThroughId,
const CSSM_DATA_PTR InputParams)

This function allows applications to call trust policy module-specific operations that have been exported. Such operations may include queries or services specific to the domain represented by the TP module.

Parameters

TPHandle (input)
The handle that describes the add-in trust policy module used to perform this function.

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

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

DBHandle (input)
The handle that describes the database used to perform this function.

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

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

InputParams (input)
A pointer to the CSSM_DATA structure containing parameters to be interpreted in a function-specific manner by the requested TP module. This parameter can be used as a pointer to an array of CSSM_DATA_PTRs.

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.

Error Codes

Value	Description
CSSM_TP_INVALID_TP_HANDLE	Invalid handle
CSSM_TP_INVALID_CL_HANDLE	Invalid handle
CSSM_TP_INVALID_DL_HANDLE	Invalid handle
CSSM_TP_INVALID_DB_HANDLE	Invalid handle
CSSM_TP_INVALID_CC_HANDLE	Invalid handle
CSSM_TP_INVALID_DATA_POINTER	Invalid pointer for input data
CSSM_TP_INVALID_ID	Invalid pass through ID
CSSM_TP_MEMORY_ERROR	Error in allocating memory
CSSM_TP_PASS_THROUGH_FAIL	Unable to perform pass through
CSSM_FUNCTION_NOT_IMPLEMENTED	Function not implemented

4.5 CSSM TP Management Functions

4.5.1 CSSM_TP_Install

CSSM_RETURN CSSMAPI CSSM_TP_Install (const char *TPName,
const char *TPFileName,
const char *TPPathName,
const CSSM_GUID_PTR GUID,
const CSSM_TPINFO_PTR TPInfo,
const void * Reserved1,
const CSSM_DATA_PTR Reserved2)

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

Parameters

TPName (input)
The name of the trust policy module.

TPFileName (input)
The name of file that implements the trust policy.

TPPathName (input)
The path to the file that implements the trust policy.

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

TPInfo (input)
A pointer to the CSSM_TPINFO structure containing information about the TP module.

Reserved1 (input)
Reserve data for the function.

Reserved2 (input)
Reserve data for the function.

Return Value

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

Error Codes

Value	Description
CSSM_TP_INVALID_POINTER	Invalid pointer
CSSM_TP_REGISTRY_ERROR	Error in writing registry

See Also

CSSM_TP_Uninstall

4.5.2 CSSM_TP_Uninstall

CSSM_RETURN CSSMAPI CSSM_TP_Uninstall (const CSSM_GUID_PTR GUID)

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

Parameters

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

Return Value

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

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_REGISTRY_ERROR	Error in writing registry

See Also

CSSM_TP_Install

4.5.3 CSSM_TP_ListModules

CSSM_LIST_PTR CSSMAPI CSSM_TP_ListModules (void)

This function returns a list containing the GUID/name pair for each of the currently-installed CL modules.

Parameters

None

Return Value

A pointer to the CSSM_LIST structure containing a GUID/name pair for each of the currently-installed TP modules. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_NO_ADDIN	No add-ins found
CSSM_MEMORY_ERROR	Error in memory allocation

4.5.4 CSSM_TP_Attach

CSSM_TP_HANDLE CSSMAPI CSSM_TP_Attach
(const CSSM_GUID_PTR GUID,
uint32 CheckCompatibleVerMajor,
uint32 CheckCompatibleVerMinor,
uint32 Application,
const CSSM_NOTIFY_CALLBACK Notification,
const void * Reserved)

This function attaches the application to the TP module, and verifies that the version of the TP module expected by the application is compatible with the version on the system.

Parameters

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

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

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

Application(input/optional)
Nonce passed to the application when its callback is invoked allowing the application to determine the proper context of operation.

Notification (input/optional)
Callback provided by the application that is called by the TP when one of two things takes place: a parallel operation completes or a token running in serial mode surrenders control to the application.

Reserved (input)
A reserved input.

Return Value

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

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_MEMORY_ERROR	Internal memory error
CSSM_INCOMPATIBLE_VERSION	Incompatible version
CSSM_EXPIRE	Add-in has expired
CSSM_ATTACH_FAIL	Unable to load TP module

See Also

CSSM_TP_Detach

4.5.5 CSSM_TP_Detach

CSSM_RETURN CSSMAPI CSSM_TP_Detach (CSSM_TP_HANDLE TPHandle)

This function detaches the application from the TP module.

Parameters

TPHandle (input)
The handle that describes the TP module.

Return Value

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

Error Codes

Value	Description
CSSM_INVALID_ADDIN_HANDLE	Invalid TP handle

See Also

CSSM_TP_Attach

4.5.6 CSSM_TP_GetInfo

CSSM_TPINFO_PTR CSSMAPI CSSM_TP_GetInfo (const CSSM_GUID_PTR GUID)

This function returns the CSSM registry information about the TP module.

Parameters

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

Return Value

A pointer to the CSSM_TPINFO structure containing information about the TP module.
If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_MEMORY_ERROR	Internal memory error
CSSM_INVALID_GUID	Unknown GUID

See Also

CSSM_TP_FreeInfo

4.5.7 CSSM_TP_FreeInfo

CSSM_RETURN CSSMAPI CSSM_TP_FreeInfo (CSSM_TPINFO_PTR TPInfo)

Frees the memory allocated by the TP module for the CSSM_TP_INFO structure.

Parameters

TPInfo (input/output)
A pointer to the CSSM_TPINFO structure to be freed.

Return Value

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

Error Codes

Value	Description
CSSM_INVALID_TPINFO_POINTER	Invalid pointer

See Also

CSSM_TP_GetInfo

5. Certificate Library Services API

5.1 Overview

The primary purpose of a Certificate Library (CL) module is to perform syntactic manipulations on a specific certificate format, and its associated certificate revocation list (CRL) format. The data format of CRLs used to track revoked certificates will be influenced, if not determined, by the data format of the certificates. For this reason, these objects should be manipulated by a single, cohesive library.

Certificate libraries manipulate memory-based objects only. The persistence of certificates and CRLs is an independent property of these objects. It is the responsibility of the application and/or the trust policy module to use data storage add-in modules to make these objects persistent (if appropriate). The particular storage mechanism used by a selected data storage module can often be selected, independent of the trust policy and the application.

The Certificate Library encapsulates 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.

Under current certificate models, such as X.509, SDSI, SPKI, etc., a single certificate represents the identity of some entity and possibly some authorizations assigned to that entity. When verifying trust in a certificate, the certificate is rarely considered as a stand-alone credential. Certificates are signed using one or more certificates. Root certificates are self-signed, but most certificates are signed using other certificates. The syntactic process of signing corresponds to a trust relationship between the entities identified by the certificates. Hence it is often necessary to manipulate groups of certificates. The CLM API define three operations on certificate groups:

- add certificates to a group
- remove certificates from a group
- verify the signatures of a group of certificates

Ten functions are defined to perform syntactic manipulation of individual certificates, and nine functions are defined to perform syntactic manipulation of CRLs. Additional operations are defined for certificate library module management and for certificate library API extensibility.

5.1.1 Application and Certificate Library 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 are registered with CSSM. An application uses registry information to find an appropriate CL and to request that CSSM attach to the CL. When CSSM attaches to the CL, it returns a CL handle to the application which uniquely identifies the pairing of the application thread to the CL module instance. This handle is used by the application to identify the CL in future function calls.

CSSM passes CL function calls from an application to the application-selected Certificate Library.

The application is responsible for the allocation and de-allocation of all memory which is passed into or out of the Certificate Library module. 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 at that time. The Certificate Library Module uses these functions to allocate and de-allocate memory which belongs to or will belong to the application.

5.1.2 Operations on Certificates

CSSM defines the general security API that all certificate libraries should provide to manipulate certificates and certificate revocation lists. The basic areas of functionality include:

- Certificate operations
- Certificate revocation list operations
- Extensibility functions
- Module management functions

Each certificate library may implement some or all of these functions. The available functions are registered with CSSM at attach time. Each certificate library should be accompanied with documentation specifying supported functions, non-supported functions, and module-specific passthrough functions. It is the responsibility of the application developer to obtain and use this information when developing applications using a selected certificate library.

The CSSM-defined API and the general semantics of those functions for all certificate libraries is described below.

CSSM_DATA_PTR CSSMAPI CSSM_CL_CertSign ( ) accepts as input a signer's certificate, a memory-resident certificate to be signed, and the *scope* of the signing process. The *scope* of a signature may be used to identify which fields of the certificate should be signed. In response, the CL module should perform the data format-specific process of generating a digital signature for the certificate, according to the scoping request. This function may be used to generate multiple signatures over a certificate. The newly-signed certificate and the associated signature are returned as memory-resident objects. If the certificate also resided in persistent storage prior to invoking this function, the newly-generated signature is not transparently written back to the data store.

CSSM_DATA_PTR CSSMAPI CSSM_CL_CertUnsign ( ) removes a signature from a signed, memory-resident certificate. The newly-unsigned certificate is returned to the calling application. If a persistent copy of the certificate also exists in some data store, the removal of a signature is not written back to the data store by this function.

CSSM_BOOL CSSMAPI CSSM_CL_CertVerify ( ) accepts as input a signer's certificate, a memory-resident, signed certificate, and the *scope* of the signing that was performed using the signer's certificate. In response, the CL module performs the data format-specific operation of checking the signature over the certificate. This determines whether or not the certificate has been altered,

and whether the signer's certificate was actually used to sign the certificate, according to the specified signing scope.

CSSM_DATA_PTR CSSMAPI CSSM_CL_CertCreate ( ) accepts as input a set of name-value pairs and a count of the number of fields presented. In response the CL should create and return a memory-resident certificate containing the values specified by the field-value pairs. The *new certificate* is not an official certificate, as it is not signed as a result of using this operation. The function CL_CertSign should be used to sign a memory-resident certificate.

CSSM_FIELD_PTR CSSMAPI CSSM_CL_CertView ( ) accepts as input a memory-resident certificate. In response, the CL module returns a set of name-value pairs, and the count of the number of pairs returned. The values are in a displayable format.

CSSM_DATA_PTR CSSMAPI CSSM_CL_CertGetFirstFieldValue (-) accepts as input a certificate and the object identifier of a field in that certificate. In response, the CL module returns the value of a selected certificate field, a count of the number of fields matching the object identifier, and a results handle. The results handle is used to get subsequent field values having the same object identifier.

CSSM_DATA_PTR CSSMAPI CSSM_CL_CertGetNextFieldValue (-) accepts as input a results handle returned by the function CSSM_CL_CertGetFirstFieldValue. In response, the CL module returns the next field value selected by the CSSM_CL_CertGetFirstFieldValue call that returned the results handle.

CSSM_RETURN CSSMAPI CSSM_CL_CertAbortQuery ( ) accepts as input a results handle returned by the function CSSM_CL_CertGetFirstFieldValue. In response, the CL module terminates the context of the get operation.

CSSM_KEY_PTR CSSMAPI CSSM_CL_CertGetKeyInfo ( ) accepts as input a certificate. In response, the CL module returns all of the data from the certificate that comprises the Key stored in that certificate. In most certificate formats these are multiple fields. This result could be achieved by multiple calls to the function named CL_CertGetFieldValue, by passing the appropriate field identifiers to extract the values that comprise the Key.

CSSM_FIELD_PTR CSSMAPI CSSM_CL_CertGetAllFields ( ) accepts as input a certificate. In response, the CL module returns a set of name-value pairs for all of the data fields contained in the certificate. This functions differs from CSSM_CL_CertView. This function can return values that cannot be displayed.

CSSM_DATA_PTR CSSMAPI CSSM_CL_CertImport ( ) each CL module manipulates a specific *native* certificate data format. In a heterogeneous world of multiple certificate formats, CL modules may wish to provide a service for converting non-native certificate formats into native formats. The import function accepts as input a certificate in non-native format and the name of that non-native format. The CL module creates and returns a corresponding memory-resident version of the input certificate in the data format native to the CL module.

CSSM_DATA_PTR CSSMAPI CSSM_CL_CertExport (.) each CL module manipulates a specific *native* certificate data format. In a heterogeneous world of multiple certificate formats, CL modules may wish to provide a service for converting their native certificate formats into non-native formats that could be used with other CL modules. The export function accepts as input a memory-resident certificate in native format, and the name of the target, non-native format. The CL module creates and returns a corresponding memory-resident version of the input certificate in the requested non-native format.

CSSM_OID_PTR CSSMAPI CSSM_CL_CertDescribeFormat (.) accepts as input the handle of a CL module. In response, CSSM returns a list of object identifiers representing the certificate field types manipulated by the CL module. These unique identifiers are used as input to CSSM_CL_CertGetFirstFieldValue ( ), and is output by the functions CSSM_CL_CertGetAllFields ( ) and CSSM_CL_CertView ( ).

5.1.3 Operations on Certificate Groups

CSSM_CERTGROUP_PTR CSSMAPI CSSM_CL_CertGroupConstruct (.) accepts as input a certificate group containing one or more certificates and a list of certificate databases that may contain certificates related to those in the input group. In response, the CL module constructs a certificate group consisting of all the certificates in the original group plus certificates selected from the certificate databases. Selection for inclusion is based upon the certificate model implemented by the CL. For example, under the X.509 model of certificates, the input certificate group can contain a leaf certificate only. The result of this operation is the chain of certificates formed by the signing process from the leaf input certificate to a self-signed root certificate.

CSSM_CERTGROUP_PTR CSSMAPI CSSM_CL_CertGroupPrune (.) accepts as input a group of certificates from which certificates should be removed, and a group of certificates that should be removed from the first group if they are present in that group. This operation serve as an inverse of the ConstructCertGroup function by removing any certificates which have only local relevance. Certificates and certificate groups are often exchanged among systems. It may be necessary to remove certificates that have only local significance before sending a certificate group to another system.

CSSM_BOOL CSSMAPI CSSM_CL_CertGroupVerify (.) accepts as input a certificate group, the *scope* of the signing that was performed on every certificate in the group, and a group of trusted certificates (root or pseudo-root certificates). In response, the CL module performs the data format-specific operation of checking the signature(s) on each certificate in the group. It is assumed that all certificates in the group were signed using the same signing scope and that all of the certificates required to verify

signatures on other certificates are included in the input group of certificates or in the group of trusted certificates. For example, if a group to be verified contains an X.509 certificate chain of depth three (certR->certM->certL), then cert M was used to sign certL, cert R was used to sign certM, and certR is in the group of trusted certificates. The function result is true if the required signatures are verified and false otherwise. The CL and the certificate model it implements defines the verification process among the certificates in the group.

5.1.4 Operations on Certificate Revocation Lists

CSSM_DATA_PTR CSSMAPI CSSM_CL_CrlCreate ( ) creates and returns an empty, memory-resident CRL.

CSSM_DATA_PTR CSSMAPI CSSM_CL_CrlAddCert ( ) accepts as input a memory-resident certificate, a memory-resident CRL, the certificate of the revoking agent, and the reason for revocation. In response, the CL module adds to the CRL a record representing the certificate. It then uses the keys associated with the revoker's certificate to sign the newly-added CRL record. The updated CRL is returned to the calling application.

CSSM_DATA_PTR CSSMAPI CSSM_CL_CrlRemoveCert (-) accepts as input a memory-resident certificate and a memory-resident CRL. In response, the CL module removes from the CRL the record which corresponds to the specified certificate. It then returns the updated CRL.

CSSM_DATA_PTR CSSMAPI CSSM_CL_CrlSign ( ) accepts as input a signer's certificate, a memory-resident CRL to be signed, and the *scope* of the signing process. In response, the CL module performs the data format-specific process of generating a digital signature for the CRL according to the scoping request. This function may be used to generate multiple signatures over a CRL. The newly-signed CRL and the associated signature are returned as memory-resident objects. If the CRL also resided in persistent storage prior to invoking this function, the newly-generated signature is not transparently written back to the data store.

CSSM_BOOL CSSMAPI CSSM_CL_CrlVerify ( ) accepts as input a signer's certificate, a memory-resident, signed CRL, and the alleged *scope* of the signing performed using the signer's certificate. In response, the CL module performs the data format-specific operation of checking the signature over the CRL. This determines whether the CRL has been tampered with and whether the signer's certificate was actually used to sign the CRL, according to the specified signing scope.

CSSM_BOOL CSSMAPI CSSM_CL_IsCertInCrl ( ) accepts as input a memory-resident CRL and a memory-resident certificate. In response, the CL module searches the CRL for a record corresponding to the certificate. If such as record is found the function will return true; otherwise the function will return false.

CSSM_DATA_PTR CSSMAPI CSSM_CL_CrlGetFirstFieldValue (-) accepts as input a memory-resident CRL. In response, the CL module returns the value of a selected CRL field, a count of the number of fields matching the object identifier, and a results handle. The results handle is used to get subsequent field values having the same object identifier.

CSSM_DATA_PTR CSSMAPI CSSM_CL_CrlGetNextFieldValue (-) accepts as input a results handle returned by the function CSSM_CL_CrlGetFirstFieldValue. In response, the CL module returns the next field value selected by the CSSM_CL_CrlGetFirstFieldValue call that returned the results handle.

CSSM_RETURN CSSMAPI CSSM_CL_CrlAbortQuery ( ) accepts as input a results handle returned by the function CSSM_CL_CrlGetFirstFieldValue. In response, the CL module terminates the context of the get operation.

CSSM_OID_PTR CSSMAPI CSSM_CL_CrlDescribeFormat (-) accepts as input the handle of a CL module. In response, CSSM returns a list of object identifiers representing the CRL field types manipulated by the CL module. These unique identifiers are used as input to CSSM_CL_CrlGetFirstFieldValue ( ).

5.1.5 Module Management Functions

CSSM_RETURN CSSMAPI CSSM_CL_Install ( ) accepts as input the name and GUID of the CL module, selected attributes describing the module, and information required by CSSM to dynamically load the module if its use is requested by an application. CSSM adds the CL module name, and attributes to the registry of CL modules.

CSSM_RETURN CSSMAPI CSSM_CL_Uninstall ( ) CSSM removes the specified CL module from the CL module registry.

CSSM_LIST_PTR CSSMAPI CSSM_CL_ListModules (-) CSSM returns a list of all the currently-registered CL modules.

CSSM_LIST_PTR CSSMAPI CSSM_CL_ListModulesForCertType (-) accepts as input the name of a certificate type. In response, CSSM returns a list of all the currently-registered CL modules whose associated attribute value for certificate type matches the input certificate type.

CSSM_CL_HANDLECSSMAPI CSSM_CL_Attach ( ) - accepts as input the GUID of a CL module, the module's major and minor versions required for compatibility with the calling application, and the application's memory management upcalls. The caller is requesting a dynamic load of the specified CL module if the available version of the CL module is compatible with the version level specified by the caller. After the module is attached, a handle identifying the module is returned to the caller. The caller may be an application, a TP module, a DL module, or another CL module.

CSSM_RETURN CSSMAPI CSSM_CL_Detach ( ) accepts as input a handle to a currently-attached CL module. The caller is requesting the dynamic unload of the specified CL module.

CSSM_CL_INFO_PTR CSSMAPI CSSM_CL_GetInfo (-) accepts as input the GUID of the CL module whose information is being requested. CSSM returns the information recorded in the CL module registry during module installation. This information includes the major and minor version numbers of the module, the certificate type supported by this CL module, the object identifiers (OIDs) which describe the certificate format, and the non-native certificate types available for certificate translations.

CSSM_RETURN CSSMAPI CSSM_CL_FreeInfo ( )

5.1.6 Extensibility Functions

The certificate library API defines one extensible operation, which allows the certificate library to make additional services available to applications and other modules. These services should be syntactic in nature (they should be dependent on the data format of the certificates and CRLs manipulated by the library).

CSSM_DATA_PTR CSSMAPI CSSM_CL_PassThrough (-) accepts as input an operation ID and an array of arbitrary input parameters. The operation ID may specify any type of operation the CL wishes to export for use by an application or module. Such operations may include queries or services that are specific to the data format of the certificates and CRLs manipulated by the CL module.

5.2 Data Structures

This section describes the data structures which 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 cssmtype.h, distributed with CSSM.

5.2.1 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
```

5.2.2 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 enum cssm_cert_type {
    CSSM_CERT_UNKNOWN=   0x00,
    CSSM_CERT_X_509v1 =  0x01,
    CSSM_CERT_X_509v2 =  0x02,
    CSSM_CERT_X_509v3 =  0x03,
    CSSM_CERT_Fortezza = 0x07,
    CSSM_CERT_PGP =      0x04,
    CSSM_CERT_SPKI =     0x05,
    CSSM_CERT_SDSIv1 =   0x06,
    CSSM_CERT_Intel =    0x08,
    CSSM_CERT_ATTRIBUTE_BER = 0x09, /* ber encoded X.509 attribute cert */
    CSSM_CERT_LAST =     0xFF
} CSSM_CERT_TYPE, *CSSM_CERT_TYPE_PTR;
```

5.2.3 CSSM_OID

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

```
typedef CSSM_DATA CSSM_OID, *CSSM_OID_PTR
```

5.2.4 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 which identifies the certificate or CRL data type or data structure.

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

5.2.5 CSSM_CERTGROUP

This structure contains a set of certificates. It is assumed that the certificates are related based on co-signaturing. The certificate group is a syntatic representation of a trust model.

```
typedef struct {
       uint32 NumCerts;     /* number of elements in CertList array */
       CSSM_DATA_PTR CertList;     /* List of opaque certificates */
       void *reserved;
} CSSM_CERTGROUP, * CSSM_CERTGROUP_PTR;
```

Definition:

NumCerts - number of certificates in the group.

CertList - List of certificates.

reserved - Reserved for future use.

5.2.6 CSSM_CLINFO

This structure contains all of the static data associated with a certificate library add-in module. This information is added to the CL registry at install time. It can be queried using the command **CSSM_CL_GetInfo ( )**

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

Definition:

CertType - An identifier for the type of certificate. This parameter is also used to determine the certificate data format.

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

CertTemplate - A pointer to an array of tag/value pairs which identify the field values of a certificate.

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

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

MultiTaskEnabled - A Boolean variable indicating whether or not this library supports multi-tasking.

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.

5.2.7 CSSM_API_MEMORY_FUNCS

This structure is used by applications to supply memory functions for the CSSM and the add-in modules. The functions are used when memory needs to be allocated by the CSSM or add-ins for returning data structures to the applications.

```
typedef struct cssm_api_memory_funcs {
    void * (*malloc_func) (uint32 size, void *allocRef);
    void (*free_func) (void *memblock, void *allocRef);
    void * (*realloc_func) (void *memblock, uint32 size, void *allocRef);
    void * (*calloc_func) (uint32 num, uint32 size, void *allocRef);
} CSSM_API_MEMORY_FUNCS, *CSSM_API_MEMORY_FUNCS_PTR
```

Definition:

malloc_func - pointer to function that returns a void pointer to the allocated memory block of at least *size* bytes from heap allocRef

free_func - pointer to function that deallocates a previously-allocated memory block (*memblock*) from heap allocRef

realloc_func - pointer to function that returns a void pointer to the reallocated memory block (*memblock*) of at least *size* bytes from heap allocRef

calloc_func - pointer to function that returns a void pointer to an array of *num* elements of length *size* initialized to zero from heap allocRef

See Appendix B for details about the application memory functions

5.3 Certificate Operations

This section describes the function prototypes and error codes which will be supported by various Certificate Library modules. The error codes given in this section constitute the generic error codes which are defined by CSSM for use by all certificate libraries in describing common error conditions. A certificate library may also return module-specific error codes.

5.3.1 CSSM_CL_CertSign

CSSM_DATA_PTR CSSMAPI CSSM_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 indicated in 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_UNSUPPORTED_OPERATION	Add-in does not support this function
CSSM_CL_CERT_SIGN_FAIL	Unable to sign certificate

See Also

CSSM_CL_CertUnsign, CSSM_CL_CertVerify

5.3.2 CSSM_CL_CertUnsign

CSSM_DATA_PTR CSSMAPI CSSM_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 which 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_UNSUPPORTED_OPERATION	Add-in does not support this function
CSSM_CL_CERT_UNSIGN_FAIL	Unable to unsign certificate

See Also

CSSM_CL_CertSign

5.3.3 CSSM_CL_CertVerify

CSSM_BOOL CSSMAPI CSSM_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_UNSUPPORTED_OPERATION	Add-in does not support this function
CSSM_CL_CERT_VERIFY_FAIL	Unable to verify certificate

See Also

CSSM_CL_CertSign

5.3.4 CSSM_CL_CertCreate

CSSM_DATA_PTR CSSMAPI CSSM_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 from 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 which 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_UNSUPPORTED_OPERATION	Add-in does not support this function
CSSM_CL_CERT_CREATE_FAIL	Unable to create certificate

See Also

CSSM_CL_CertSign, CSSM_CL_CertGetFirstFieldValue

5.3.5 CSSM_CL_CertView

CSSM_FIELD_PTR CSSMAPI CSSM_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 *CertTemplate* array.

Return Value

A pointer to an array of CSSM_FIELD structures which contain the viewable fields of the input 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_DATA_POINTER	Invalid pointer input
CSSM_CL_UNKNOWN_FORMAT	Unrecognized certificate format
CSSM_CL_MEMORY_ERROR	Not enough memory
CSSM_CL_UNSUPPORTED_OPERATION	Add-in does not support this function
CSSM_CL_CERT_VIEW_FAIL	Unable to view certificate

See Also

CSSM_CL_CertGetFirstFieldValue, CSSM_CL_CertGetAllFields

5.3.6 CSSM_CL_CertGetFirstFieldValue

CSSM_DATA_PTR CSSMAPI CSSM_CL_CertGetFirstFieldValue (CSSM_CL_HANDLE CLHandle,
const CSSM_DATA_PTR Cert,
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 which identifies the field value to be extracted from the *Cert*.

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

NumberOfMatchedFields (output)
The number of fields which 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_TAG	Unknown field tag in OID
CSSM_CL_MEMORY_ERROR	Not enough memory
CSSM_CL_UNSUPPORTED_OPERATION	Add-in does not support this function
CSSM_CL_CERT_GET_FIELD_VALUE_FAIL	Unable to get field value

See Also

CSSM_CL_CertGetNextFieldValue, CSSM_CL_CertAbortQuery, CSSM_CL_CertGetAllFields

5.3.7 CSSM_CL_CertGetNextFieldValue

CSSM_DATA_PTR CSSMAPI CSSM_CL_CertGetNextFieldValue (CSSM_CL_HANDLE CLHandle, CSSM_HANDLE ResultsHandle)

This function returns the next certificate field which matched the OID and selection predicate 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 which 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_NO_FIELD_VALUES	No more field values for the input handle
CSSM_CL_MEMORY_ERROR	Not enough memory
CSSM_CL_UNSUPPORTED_OPERATION	Add-in does not support this function
CSSM_CL_CERT_GET_FIELD_VALUE_FAIL	Unable to get field value

See Also

CSSM_CL_CertGetFirstFieldValue, CSSM_CL_CertAbortQuery

5.3.8 CSSM_CL_CertAbortQuery

CSSM_RETURN CSSMAPI CSSM_CL_CertAbortQuery (CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE ResultsHandle)

This function terminates the query initiated by CSSM_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)
A pointer to the handle which 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 the certificate query

See Also

CSSM_CL_CertGetFirstFieldValue, CSSM_CL_CertGetNextFieldValue

5.3.9 CSSM_CL_CertGetKeyInfo

CSSM_KEY_PTR CSSMAPI CSSM_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 required to create a cryptographic context.

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 in OID
CSSM_CL_MEMORY_ERROR	Not enough memory
CSSM_CL_UNSUPPORTED_OPERATION	Add-in does not support this function
CSSM_CL_CERT_GET_KEY_INFO_FAIL	Unable to get key information

See Also

CSSM_CL_CertGetFirstFieldValue

5.3.10 CSSM_CL_CertGetAllFields

CSSM_FIELD_PTR CSSMAPI CSSM_CL_CertGetAllFields (CSSM_CL_HANDLE CLHandle,
CSSM_DATA_PTR Cert,
uint32 *NumberOfFields)

This function returns a list of the fields in 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 fields will be returned.

NumberOfFields (output)
The length of the returnedarray of fields.

Return Value

A pointer to an array of CSSM_FIELD structures which contain the values of all of the fields of the input 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 handle
CSSM_CL_INVALID_DATA_POINTER	Invalid DATA pointer
CSSM_CL_MEMORY_ERROR	Error allocating memory
CSSM_CL_CERT_GET_FIELD_VALUE_FAIL	Unable to return the list of fields

See Also

CSSM_CL_CertGetFirstFieldValue, CSSM_CL_CertDescribeFormat, CSSM_CL_CertView

5.3.11 CSSM_CL_CertImport

CSSM_DATA_PTR CSSMAPI CSSM_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 which 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. 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_UNSUPPORTED_OPERATION	Add-in does not support this function
CSSM_CL_CERT_IMPORT_FAIL	Unable to import certificate

See Also

CSSM_CL_CertExport

5.3.12 CSSM_CL_CertExport

CSSM_DATA_PTR CSSMAPI CSSM_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 which 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_UNSUPPORTED_OPERATION	Add-in does not support this function
CSSM_CL_CERT_EXPORT_FAIL	Unable to export certificate

See Also

CSSM_CL_CertImport

5.3.13 CSSM_CL_CertDescribeFormat

CSSM_OID_PTR CSSMAPI CSSM_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 returned array of OIDs.

Return Value

A pointer to the array of CSSM_OIDs which represent the supported certificate format. 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 handle
CSSM_CL_MEMORY_ERROR	Error allocating memory
CSSM_CL_CERT_DESCRIBE_FORMAT_FAIL	Unable to return the list of fields

See Also

CSSM_CL_CertGetAllFields

5.4 Certificate Group Operations

This section describes the function prototypes and error codes supported by Certificate Library modules that manage certificate groups. The error codes given in this section constitute the generic error codes which are defined by CSSM for use by all certificate libraries in describing common error conditions. A certificate library may also return module-specific error codes.

5.4.1 CSSM_CL_CertGroupConstruct

CSSM_CERTGROUP_PTR
CSSM_CL_CertGroupConstruct
(CSSM_CL_HANDLE CLHandle,
CSSM_CERTGROUP_PTR CertGroupFrag,
CSSM_DB_LIST_PTR DBList)

This function constructs an ordered certificate group from the CertGroupFrag certificate group and the contents of the databases passed in DBList. There is no implied ordering for the certificates in CertGroupFrag except that the first certificate in the certificate group is assumed to be the starting point for constructing the certificate group. An ordering relationship may be defined and recorded in the certificates themselves or assumed by the certificate library model. For example, if the certificate model is a hierarchical model of certificate chains, the leaf certificate in the chain is the CertGroupFrag and the complete certificate chain including the self-signed root certificate is the anticipated result of the construction operation.

Parameters

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

CertGroupFrag (input)
A group of certificates to be used to build an ordered certificate group. The first certificate in the group should be the certificate for which the ordered certificate group is being constructed. No particular ordering of the rest of this group is expected or implied.

DBList (input)
A list of certificate databases containing certificates that may be used to construct the ordered certificate group.

Return Value

A pointer to a group of certificates, ordered in the context of the certificate type and trust model.

Error Codes

Value	Description
CSSM_ INVALID_CL_ HANDLE	Invalid certificate library handle
CSSM_CL_INVALID_CERT_GROUP	Invalid certificate group
CSSM_INVALID_DB_HANDLE	Bad database handle
CSSM_ MEMORY_ERROR	Not enough memory to allocate

See Also

CSSM_CL_PruneCertGroup

5.4.2 CSSM_CL_CertGroupPrune

CSSM_CERTGROUP_PTR CSSM_CL_CertGroupPrune (CSSM_CL_HANDLE CLHandle,
CSSM_CERTGROUP_PTR CertGroup,
CSSM_DB_LIST_PTR DBList);

This function prunes all certificates from CertGroup which are not verifiable by an external host. This function determines which root certificates were generated locally by checking the passed DBList for certificates which exist in both an OWNED and a ROOT database. These certificates will be removed from the CertGroup, as well as any certificate signed by them. In addition this function will remove any self-signed certificates from the CertGroup.

Parameters

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

CertGroup (input)
The certificate group to be pruned. Most likely, this parameter would be filled with the return value of a call to CSSM_CL_CertGroupConstruct( ).

DBList (input)
The list of databases to be used to determine which certificates are local roots (certificates signed by a locally-held private key).

Return Value

Returns an exportable certificate group which can be completed and verified by external hosts. If the list returned is NULL, an error has ocurred. Errors can be retrieved by calling CSSM_GetLastError( ).

Error Codes

Value	Description
CSSM_INVALID_ CL_HANDLE	Invalid certificate library handle
CSSM_CL_INVALID_CERT_GROUP	Invalid certificate group
CSSM_MEMORY_ERROR	Internal memory error

See Also

CSSM_CL_CertGroupConstruct

5.4.3 CSSM_CL_CertGroupVerify

CSSM_BOOL CSSM_CL_CertGroupVerify (CSSM_CL_HANDLE CLHandle,
CSSM_CSP_HANDLE CSPHandle,
CSSM_CERTGROUP_PTR OrderedCertGroup,
const CSSM_FIELD_PTR VerifyScope,
uint32 ScopeSize,
CSSM_DB_LIST_PTR DBList,
CSSM_CERTGROUP_PTR RootCerts)

This function verifies the OrderedCertGroup. This function accepts a list of opaque certificates and verifies the signatures on each certificate according to conventions defined by the CLM developer. For example, if the X509 certificate model is being used then a signature chain is verified. The OrderedCertGroup is expected to be the output of a call to CSSM_CL_CertGroupConstruct( ). The verification will fail if a revoked certificate is found in the certificate group, improper ordering of the certificate group is found, a certificate in the group fails to verify, or a self-signed root certificate cannot be found in RootCerts or in the ROOT databases in the DBList. If RootCerts is non-NULL, the ROOT databases in the DBList will not be checked to verify the existance of a root certificate.

Parameters

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

CSPHandle (input)
The handle that describes the add-in cryptographic library module used to perform the verify operation for each certificate in the group to be verified.

OrderedCertGroup (input)
The set of related certificates presented for verification.

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.

DBList (input)
The list of databases to be used to determine which certificates are valid roots (trusted self-signed certificates)

RootCerts (input)
The set of trusted root certificates that may have been used to sign one or more of the certificates in the CertGroup. These certificates are recognized as trusted signers and represent the termination of one verification path among the certificates in the CertGroup. If this value is non-NULL then checking for trusted root certificates will done exclusively in the RootCerts list, and not in any of the ROOT databases included in the DBList.

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_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_INVALID_CERT_GROUP	Invalid certificate group
CSSM_CL_UNKNOWN_FORMAT	Unrecognized certificate format
CSSM_CL_INVALID_SCOPE	Invalid scope
CSSM_CL_UNSUPPORTED_OPERATION	Add-in does not support this function
CSSM_CL_CERT_VERIFY_FAIL	Unable to verify certificate
CSSM_MEMORY_ERROR	Internal memory error

See Also

CSSM_CL_CertGroupConstruct, CSSM_CL_CertGroupPrune

5.5 Certificate Revocation List Operations

This section describes the function prototypes and error codes which will be supported by various Certificate Library modules. The error codes given in this section constitute the generic error codes which are defined by CSSM for use by all certificate libraries in describing common error conditions. A certificate library may also return module-specific error codes.

5.5.1 CSSM_CL_CrlCreate

CSSM_DATA_PTR CSSMAPI CSSM_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

5.5.2 CSSM_CL_CrlAddCert

CSSM_DATA_PTR CSSMAPI CSSM_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

CSSM_CL_CrlRemoveCert

5.5.3 CSSM_CL_CrlRemoveCert

CSSM_DATA_PTR CSSMAPI CSSM_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_CERT_NOT_FOUND_IN_CRL	Certificate not referenced by the CRL
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

CSSM_CL_CrlAddCert

5.5.4 CSSM_CL_CrlSign

CSSM_DATA_PTR CSSMAPI CSSM_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	Signing scope is invalid
CSSM_CL_MEMORY_ERROR	Not enough memory to allocate the CRL
CSSM_CL_CRL_SIGN_FAIL	Unable to sign CRL

See Also

CSSM_CL_CrlVerify

5.5.5 CSSM_CL_CrlVerify

CSSM_BOOL CSSMAPI CSSM_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	Verify scope is invalid
CSSM_CL_MEMORY_ERROR	Not enough memory to allocate the CRL
CSSM_CL_CRL_VERIFY_FAIL	Unable to verify CRL

See Also

CSSM_CL_CrlSign

5.5.6 CSSM_CL_IsCertInCrl

CSSM_BOOL CSSMAPI CSSM_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

5.5.7 CSSM_CL_CrlGetFirstFieldValue

CSSM_DATA_PTR CSSMAPI CSSM_CL_CrlGetFirstFieldValue (CSSM_CL_HANDLE CLHandle,
const CSSM_DATA_PTR Crl,
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 which contains the CRL from which the first revocation record is to be retrieved.

CrlField (input)
An object identifier which 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 which match the *CrlField* OID.

Return Value

Returns a pointer to a CSSM_DATA structure containing the first field which 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 CL handle
CSSM_CL_UNKNOWN_TAG	Unrecognized field tag in OID
CSSM_CL_NO_FIELD_VALUES	No fields match the specified OID
CSSM_CL_INVALID_CRL_PTR	Invalid CRL pointer
CSSM_CL_CRL_GET_FIELD_VALUE_FAIL	Unable to get first field value

See Also

CSSM_CL_CrlGetNextFieldValue, CSSM_CL_CrlAbortQuery

5.5.8 CSSM_CL_CrlGetNextFieldValue

CSSM_DATA_PTR CSSMAPI CSSM_CL_CrlGetNextFieldValue (CSSM_CL_HANDLE CLHandle, CSSM_HANDLE ResultsHandle)

This function returns the next CRL field which 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 which 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 CL handle
CSSM_CL_NO_FIELD_VALUES	No more matches in the CRL
CSSM_CL_INVALID_CRL_PTR	Invalid CRL pointer
CSSM_CL_CRL_GET_FIELD_VALUE_FAIL	Unable to get next value

See Also

CSSM_CL_CrlGetFirstFieldValue, CSSM_CL_CrlAbortQuery

5.5.9 CSSM_CL_CrlAbortQuery

CSSM_RETURN CSSMAPI CSSM_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 which 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 CL handle
CSSM_CL_INVALID_RESULTS_HANDLE	Invalid query handle
CSSM_CL_CRL_ABORT_QUERY_FAIL	Unable to get next item

See Also

CSSM_CL_CrlGetFirstFieldValue, CSSM_CL_CrlGetNextFieldValue

5.5.10 CSSM_CL_CrlDescribeFormat

CSSM_OID_PTR CSSMAPI CSSM_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 returnedarray of OIDs.

Return Value

A pointer to the array of CSSM_OIDs which represent the supported CRL format. 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 handle
CSSM_CL_MEMORY_ERROR	Error allocating memory
CSSM_CL_CRL_DESCRIBE_FORMAT_FAIL	Unable to return the list of fields

5.6 Module Management Functions

5.6.1 CSSM_CL_Install

CSSM_RETURN 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 persistent CSSM 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_OK return value signifies that information has been updated. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_REGISTRY_ERROR	Error in writing registry

See Also

CSSM_CL_Uninstall

5.6.2 CSSM_CL_Uninstall

CSSM_RETURN 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_OK return value signifies that information has been deleted. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_REGISTRY_ERROR	Error in writing registry

See Also

CSSM_CL_Install

5.6.3 CSSM_CL_ListModules

CSSM_LIST_PTR CSSMAPI CSSM_CL_ListModules (void)

This function returns a list containing the GUID/name pair of each of the currently-installed CL modules.

Parameters

None

Return Value

A pointer to the CSSM_LIST structure containing a GUID/name pair for each of the CL modules. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_MEMORY_ERROR	Error in memory allocation

See Also

CSSM_CL_ListModulesForCertType

5.6.4 CSSM_CL_ListModulesForCertType

CSSM_LIST_PTR CSSMAPI CSSM_CL_ListModulesForCertType
(CSSM_CERT_TYPE CertType)

This function returns a list containing the GUID/name pair of each of the currently-installed CL modules which support the specified certificate type.

Parameters

CertType (input)
The certificate type to be compared against in the search for all compatible CLs.

Return Value

A pointer to the CSSM_LIST structure containing the names of CL modules. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_MEMORY_ERROR	Error in memory allocation

See Also

CSSM_CL_ListModules

5.6.5 CSSM_CL_Attach

CSSM_CL_HANDLE CSSMAPI CSSM_CL_Attach
(const CSSM_GUID_PTR GUID,
uint32 CheckCompatibleVerMajor,
uint32 CheckCompatibleVerMinor,
const CSSM_API_MEMORY_FUNCS_PTR MemoryFuncs,
uint32 Application,
const CSSM_NOTIFY_CALLBACK Notification,
const void * Reserved)

This function attaches the CL module to CSSM. The CL module will test for compatibility with the version specified. If it is not compatible, it will not successfully attach. The application must use the *MemoryFuncs* parameter to specify the pointer to its memory allocation and de-allocation routines.

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 a table containing pointers to the application's memory allocation and de-allocation routines.

Application(input/optional)
Nonce passed to the application when its callback is invoked allowing the application to determine the proper context of operation.

Notification (input/optional)
Callback provided by the application that is called by the CL when one of two things takes place: a parallel operation completes or a token running in serial mode surrenders control to the application.

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_POINTER	Invalid pointer
CSSM_MEMORY_ERROR	Internal memory error
CSSM_INCOMPATIBLE_VERSION	Incompatible version
CSSM_ATTACH_FAIL	Unable to attach to CL module

See Also

CSSM_CL_Detach

5.6.6 CSSM_CL_Detach

CSSM_RETURN CSSMAPI CSSM_CL_Detach (CSSM_CL_HANDLE CLHandle)

This function detaches the CL module from CSSM.

Parameters

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

Return Value

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

Error Codes

Value	Description
CSSM_INVALID_ADDIN_HANDLE	Invalid CL handle

See Also

CSSM_CL_Attach

5.6.7 CSSM_CL_GetInfo

CSSM_CLINFO_PTR CSSMAPI CSSM_CL_GetInfo (const CSSM_GUID_PTR GUID)

This function returns the information associated with the CL module.

Parameters

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

Return Value

A pointer to the CSSM_CLINFO structure containing information about the CL module. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_MEMORY_ERROR	Internal memory error
CSSM_INVALID_GUID	No known CL module with specified GUID

See Also

CSSM_CL_FreeInfo

5.6.8 CSSM_CL_FreeInfo

CSSM_RETURN CSSMAPI CSSM_CL_FreeInfo (CSSM_CLINFO_PTR CLInfo)

This function frees the memory allocated by CSSM to hold the CSSM_CLINFO structure returned by the CSSM_CL_GetInfo function.

Parameters

CLInfo (input)
A pointer to the CSSM_CLInfo structure to be freed.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_ INVALID_POINTER	Invalid pointer

See Also

CSSM_CL_GetInfo

5.7 Extensibility Functions

5.7.1 CSSM_CL_PassThrough

CSSM_DATA_PTR CSSMAPI CSSM_CL_CertPassThrough
(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. Such operations may include queries or services that are specific to the domain represented by the CL module.

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 exported function to perform.

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

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

6. Data Storage Library Services API

6.1 Overview

The primary purpose of a data storage library (DL) module is to provide persistent storage of security-related objects including certificates, certificate revocation lists (CRLs), keys, and policy objects. A DL module is responsible for the creation and accessibility of one or more data stores. A single DL module can be tightly tied to a CL and/or TP module, or can be independent of all other module types. A single data store can contain a single object type in one format, a single object type in multiple formats, or multiple object types. The persistent repository can be local or remote (For example, a DL provides client access to a remote directory/storage service). CSSM stores and manages meta-information about a DL in the CSSM registry. This information describes the storage and retrieval capabilities of a DL. Applications can query the CSSM registry to obtain information about the available DLs and attach to a DL that provides the needed services. Each DL should store meta-information about each of the data stores it manages.

The DL APIs define a data storage model that can be implemented using a custom storage device, a traditional local or remote file system service, a database management system package, or a complete information management system. The abstract data model defined by the DL APIs partitions all attributes stored in a data record into two general categories: immutable attributes and mutable attributes.

The DL APIs also support the notion that a DL may not be able to parse/interpret all attributes of a data object. For example, a DL that stores certificates may not be able to examine fields within those certificates when performing retrieval operations. In such cases, the APIs allow the caller to specify the handle of a certificate library module to be invoked to parse field values from the certificate.

The DL APIs defined in CSSM Release 1.0 included operations that were specific to certificates and CRLs. These functions are made obsolete by new CSSM Release 1.2 APIs to store and retrieve an extensible set of security-related data objects. However the obsolete APIs have been retained for backward compatibility. Developers are encouraged to use the new data object APIs defined in this specification exclusively.

6.1.1 Data source Operations

CSSM_DB_HANDLE CSSMAPI CSSM_DL_DbOpen ( ) opens a data store with the specified logical name. Returns a handle to the data store.

CSSM_RETURN CSSMAPI CSSM_DL_DbClose ( ) closes a data store.

CSSM_ DB_HANDLE CSSMAPI CSSM_DL_DbCreate ( ) creates a new, empty data store with the specified logical name, and the specified attribute schema.

CSSM_RETURN CSSMAPI CSSM_DL_DbDelete ( ) deletes all records from the specified data store and removes current state information associated with that data store.

CSSM_RETURN CSSMAPI CSSM_DL_DbImport ( ) accepts as input a filename and a logical name for a data store. The file is an exported copy of an existing data store. The data records contained in the file must be in the native format of a DL module. A DL module imports all security objects in the file (such as certificates and CRLs), creating a new data record for each. The specified logical name is assigned to the new data store. Note: This mechanism can be used to copy data stores among systems or

to restore a persistent data store from a backup copy. It could also be used to import data stores that were created and managed by other DLs, but this is not the typical implementation and use of this interface.

CSSM_RETURN CSSMAPI CSSM_DL_DbExport ( ) accepts as input the logical name of a data store and the name of a target output file. The specified data store contains persistent data records. A representation of the schema for the data store being exported is written to the file along with a copy of each data record in the data store. Note: This mechanism can be used to copy data stores among systems or to create a backup of persistent data stores.

CSSM_DBINFO_PTR CSSMAPI CSSM_DL_DbGetInfo ( ) CSSM returns a CSSM_DBINFO structure describing the Datasource meta data like DbType, Dbrecord type etc.

CSSM_RETURN CSSMAPI DL_ CSSM_DbSetInfo ( ) stores the Data source-specific meta data information.

CSSM_RETURN CSSMAPI CSSM_DL_FreeDbInfo ( ) frees the memory structure returned by the CSSM_DL_DbGetInfo function.

CSSM_NAME_LIST_PTR CSSMAPI CSSM_DL_GetDbHandleToName() retrieves the data source corresponding to an opened database handle. A DL module is responsible for allocating the memory required for the list.

6.1.2 Generic Data Storage Operations

CSSM_RETURN CSSMAPI CSSM_DL_DataInsert ( ) accepts as input a data object, the record type of the object, a handle to a data store, and an optional handle to a module that can parse un-crackable attribute fields of .object using GetFirstField operations. The data object is made persistent in the specified data store. This may or may not include the creation of index entries, etc. The mechanisms used to store and retrieve persistent certificates is private to the implementation of the Data Storage Library.

CSSM_RETURN CSSMAPI CSSM_DL_DataDelete ( ) accepts as input a data object, the record type of the object, and a handle to a data store. The object is removed from the data store. If the object is not found in the specified data store, the operation fails.

CSSM_DATA_PTR CSSMAPI CSSM_DL_DataGetFirst ( ) accepts as input a selection predicate, the type of data record to be retrieved, and a handle to a data store. The specified data store is searched for data objects of the specified type that match the selection criteria. Selection predicates are represented in one or two forms: a predicate string (For example, a string that is appropriate as a *where* clause in an SQL statement), or as a set of (name, value, relational operator) triples that are connected by a conjunctive operator. The conjunctive operators are Boolean-and and Boolean-or. The relational operators include greater-than, less-than, equal-to, and not-equal-to. This function returns a count of the total number of data objects matching the selection criteria, the first data object matching the criteria, and a selection handle that may be used to retrieve

the subsequent objects matched in the search. A data storage library may limit the number of concurrently managed selection handles to exactly one. The library developer must document all such restrictions and application developers should proceed accordingly.

CSSM_DATA_PTR CSSMAPI CSSM_DL_DataGetNext ( ) accepts as input the type of the record to be returned and a selection results handle that was returned by an invocation of the function CSSM_DL_DataGetFirst ( ). In response, a DL module returns the next data record from the set specified by the selection results handle. If all data records have already been returned from the set specified by the selection handle, then the function returns a NULL data record. A data storage library may limit the number of concurrently-managed selection result handles to exactly one. The library developer must document such restrictions, and application developers should proceed accordingly.

CSSM_RETURN CSSMAPI CSSM_DL_DataAbortQuery ( ) cancels the query initiated by CSSM_DL_DataGetFirst function and resets the selection results handle.

6.1.3 Certificate Storage Operations - included for backward compatibility with CSSM 1.0

CSSM_RETURN CSSMAPI CSSM_DL_CertRevoke ( ) accepts as input a certificate to be revoked and a handle to a data store. The knowledge that the certificate has been revoked is made persistent. The representation of this information and the mechanism for creating and managing the *representation of revocation* is private to the implementation of the Data Storage Library.

CSSM_RETURN CSSMAPI CSSM_DL_CertInsert ( ) accepts as input a certificate, and optional handle to a certificate library module (which may be used to parse the certificate), and a handle to a data store. The certificate is made persistent in the specified data store. This may or may not include the creation of index entries, etc. The mechanisms used to store and retrieve persistent certificates is private to the implementation of the Data Storage Library.

CSSM_RETURN CSSMAPI CSSM_DL_CertDelete ( ) accepts as input a certificate and a handle to a data store. The certificate is removed from the data store. If the certificate is not found in the specified data store, the operation fails.

CSSM_DATA_PTR CSSMAPI CSSM_DL_CertGetFirst ( ) accepts as input a set of relational expressions, a single conjunctive operator, and a handle to a data store. The specified data store is searched for certificates that match the selection criteria. The selection criteria is the expression formed by connecting all of the relational expressions using the one conjunctive operator. The conjunctive operators are Boolean-and and Boolean-or. The relational operators include greater-than, less-than, equal-to, and not-equal-to. This function returns a count of the total number of certificates matching the selection criteria, the first certificate matching the criteria, and a

selection handle that may be used to retrieve the subsequent certificates matching the selection criteria. A data storage library may limit the number of concurrently-managed selection handles to exactly one. The library developer must document such restrictions and application developers should be aware of such restrictions.

CSSM_DATA_PTR CSSMAPI CSSM_DL_CertGetNext ( ) accepts as input a selection handle that was returned by an invocation of the function CSSM_DL_CertGetFirst ( ). In response, a DL module returns the next certificate from the set specified by the selection handle. If all certificates have already been returned from the set specified by selection handle, then the function returns a NULL certificate. A data storage library may limit the number of concurrently-managed selection handles to exactly one. The library developer must document such restrictions, and application developers should be aware of such restrictions.

CSSM_RETURN CSSMAPI CSSM_DL_CertAbortQuery ( ) eancels the query initiated by CSSM_DL_CertGetFirst function and resets the selection handle.

6.1.4 CRL Storage Operations - included for backward-compatibility with CSSM 1.0

CSSM_RETURN CSSMAPI CSSM_DL_CrlInsert ( ) accepts as input a CRL record and the handle of a data store of CRL records. The new record is made persistent in the data store of CRL records. This may or may not include the creation of index entries, etc. The mechanisms used to store and retrieve persistent CRL records is private to the implementation of the data storage library.

CSSM_RETURN CSSMAPI CSSM_DL_CrlDelete ( ) accepts as input a CRL record and the handle of a data store of CRL records. The single record is removed from the data store. If the record is not found in the specified data store, the operation fails.

CSSM_DATA_PTR CSSMAPI CSSM_DL_CrlGetFirst ( ) accepts as input a set of relational expressions, a single conjunctive operator, and a handle to a data store. The specified data store is searched for CRL records that match the selection criteria. The selection criteria is the expression formed by connecting all of the relational expressions using the one conjunctive operator. The conjunctive operators are Boolean-and and Boolean-or. The relational operators include greater-than, less-than, equal-to, and not-equal-to. This function returns a count of the total number of CRL records matching the selection criteria, the first CRL record matching the criteria, and a selection handle that may be used to retrieve the subsequent CRL records matching the selection criteria. A data storage library may limit the number of concurrently-managed selection handles to exactly one. The library developer must document such restrictions and application developers should be aware of such restrictions.

CSSM_DATA_PTR CSSMAPI CSSM_DL_CrlGetNext ( ) accepts as input a selection handle that was returned by an invocation of the function CSSM_DL_CrlGetFirst ( ). In response, a DL module returns

the next CRL record from the set specified by the selection handle. If all CRL records have already been returned from the set specified by selection handle, then the function returns a NULL CRL pointer. A data storage library may limit the number of concurrently-managed selection handles to exactly one. The library developer must document such restrictions and application developers should be aware of such restrictions.

CSSM_RETURN CSSMAPI CSSM_DL_CrlAbortQuery ( ) cancels the query initiated by the DL_CrlGetFirst function and resets the selection handle.

6.1.5 Module Management Functions

CSSM_RETURN CSSMAPI CSSM_DL_Install ( ) accepts as input the name and GUID of a DL module, selected attributes describing the module, and information required by CSSM to dynamically load the module if its use is requested by some application. The storage capabilities of some storage devices and implementations can not be fully determined until execution time. DLs using such devices must register all known, static capability information at install time. Incremental capability information can be added to the registry when a caller attaches to the module or at any time during the use of the module at install time. CSSM adds a DL module name and attributes to the registry of DL modules.

CSSM_RETURN CSSMAPI CSSM_DL_Uninstall ( ) CSSM removes a specified DL module from the registry.

CSSM_DLINFO_PTR CSSMAPI CSSM_DL_GetInfo (-) CSSM returns a structure describing the identity and storage capabilities of a DL, as it is currently recorded in a CSSM registry. Multiple capability structures are returned if the DL manages multiple storage mediums (such as two custom hardware storage devices). The storage capabilities of certain types of devices are not known at install time. The caller can specify if such incomplete capability descriptions should or should not be included in the returned information.

CSSM_RETURN CSSMAPI CSSM_DL_FreeInfo ( ) frees the memory structure CSSM allocated to hold the module-descriptive information returned by the CSSM_DL_GetInfo function. Multiple structures can be freed if multiple structures were returned by the DL_GetInfo function.

CSSM_LIST_PTR CSSMAPI CSSM_DL_ListModules (-) CSSM returns a list of all currently-registered DL modules.

CSSM_DL_HANDLE CSSMAPI CSSM_DL_Attach ( ) accepts as input the GUID of a DL module, the major and minor version number desired by the caller, and an optional deviceID and device access flags for special storage devices. The caller is requesting a dynamic load of the specified DL module, if the available version of a DL module is compatible with the version level specified by the caller. The caller may be an application, a TP module, a CL module, or another DL module.

CSSM_RETURN CSSMAPI CSSM_DL_Detach ( ) the caller is requesting the dynamic unload of a specified DL module.

CSSM_NAME_LIST_PTR CSSMAPI CSSM_DL_GetDbNames (-) the specified DL module returns a memory-resident list of the logical data store names that this module can access and a count of the number of logical names in that list. A DL module is responsible for allocating the memory required for the list.

CSSM_RETURN CSSMAPI CSSM_DL_FreeNameList ( ) frees the list returned by DL_GetDbNames function.

6.1.6 Extensibility Functions

CSSM_DATA_PTR CSSMAPI CSSM_DL_PassThrough (-) accepts as input an operation ID and a set of arbitrary input parameters. The operation ID may specify any type of operation a DL wishes to export for use by an application or by another module. Such operations may include queries or services that are specific to certain types of certificates, or to the relationships between the certificates and CRLs manipulated by a DL module.

6.2 Data storage Data Structures

```
typedef uint32 CSSM_DL_HANDLE    /* data storage library Handle */
typedef uint32 CSSM_DB_HANDLE    /* Data storage Handle */
typedef uint32 CSSM_MODULE_HANDLE, *CSSM_MODULE_HANDLE_PTR /* Service provider
                                                              Handle*/
```

6.2.1 CSSM_DB_LONGHANDLE

A pair consisting of data library module handle and database handle.

```
typedef struct {
    CSSM_DL_HANDLE DLHandle;
    CSSM_DB_HANDLE DBHandle;
} CSSM_DB_LONGHANDLE, *CSSM_DB_LONGHANDLE_PTR;
```

6.2.2 CSSM_DB_LIST

This list pairs a data store with a data storage library Module that can be used to manage that data store. The list is often used to provide other modules with the set of handles required to search an application-selected data store.

```
typedef struct {
    uint32 NumHandles;
    CSSM_DB_LONGHANDLE_PTR DBLongHandle;
} CSSM_DB_LIST, *CSSM_DB_LIST_PTR;
```

Definition:

NumHandles- Number of database sources in the list.

DBLongHandle - List of data library module/database pairs.

6.2.3 CSSM_DB_TYPE

This is the initial enumeration of semantic, user-defined data store types. Each data store can have an associated, semantic type defined by the user/creator of the data store. New types should be added as required to describe key databases and policy object databases, among others.

```
typedef enum {
       CSSM_DB_TAG_NONE=0L,
       CSSM_DB_TAG_ROOTS,    /* contains self-signed root certs */
       CSSM_DB_TAG_TRUSTED,         /* re-issued locally (DL must protect?) */
       CSSM_DB_TAG_SYSTEM,   /* contains CSSM system certs */
       CSSM_DB_TAG_OWNER,   /* certs owned by users (ie private key in CSP) */
       CSSM_DB_TAG_REVOKED   /* contains revocation info - used with CRL APIs */
       /* others? */
} CSSM_DB_TYPE;
```

6.2.4 CSSM_DATA_RECORD_TYPE

These are data record types that may be stored and managed by data storage library modules. New types can be added to this list. Storage modules may store and manage multiple types in a single data store or in separate data stores.

```
typedef enum  cssm_data_record_type {
       CSSM_DATA_RECORD_ANY,           /* lib can store and manage any objects*/
       CSSM_DATA_RECORD_CERTS,         /* lib can store & manage certs */
       CSSM_DATA_RECORD_CRLS,          /* lib can store & manage CRLS */
       CSSM_DATA_RECORD_KEYS,          /* lib can store & manage keys */
       CSSM_DATA_RECORD_POLICIES,      /* lib can store & manage policies */
       CSSM_DATA_RECORD_GENERIC        /* lib can store & manage generic data */
} CSSM_DATA_RECORD_TYPE, *CSSM_DATA_RECORD_TYPE_PTR;
```

6.2.5 CSSM_DB_ATTRIBUTE_USAGE

These are the ways in which an attribute of a data store record can be treated by the data storage library module. The current options include indexed attributes and non-indexed attributes. Indexes may be one-to-many (which is a regular index), or one-to-one (which is a unique index). Any given DL may or may not support all attribute usages.

```
typedef enum  cssm_db_attribute_usage {
       CSSM_DB_INDEX,
       CSSM_DB_UNIQUE_INDEX,
       CSSM_DB_NON_INDEX
}   CSSM_DB_ATTRIBUTE_USAGE, *CSSM_DB_ATTRIBUTE_USAGE_PTR;
```

6.2.6 CSSM_ATTRIBUTE_ID_FORMAT

These are the distinct representations for an attribute identifier. An attribute field of a data record may be identified by an OID value or an attribute name.

```
typedef enum cssm_attribute_id_format {
    CSSM_OID_NAME_FORMAT,
    CSSM_STRING_NAME_FORMAT
} CSSM_ATTRIBUTE_ID_FORMAT, *CSSM_ATTRIBUTE_ID_FORMAT_PTR
```

6.2.7 CSSM_ATTRIBUTE_NAME

This structure defines an attribute name that can be used to reference an attribute field of a data store record when selecting records from the data store or when creating a new data store.

```
typedef struct cssm_attribute_name {
       CSSM_ATTRIBUTE_ID_FORMAT AttributeIdFormat;
       CSSM_DATA AttributeId;
} CSSM_ATTRIBUTE_NAME, *CSSM_ATTRIBUTE_NAME_PTR
```

Definition:
AttributeIdFormat - Indicates the format of the attribute identifier in the *AttributeId*.

AttributeId - An identifier that uniquely identifies an attribute field contained in data store records. The format of the identifier is specified by the *AttributeIdFormat*.

6.2.8 CSSM_DB_ATTRIBUTE_INFO

These are the semantic types of data stores that can be managed by a data storage library module.

```
typedef struct {
       CSSM_DB_ATTRIBUTE_NAME  AttributeName;
       CSSM_DB_ATTRIBUTE_USAGE  AttributeUsage;
}  CSSM_DB_ATTRIBUTE_INFO, CSSM_ATTRIBUTE_INFO_PTR;
```

Definition:

AttributeName - Name of an attribute field of a data store record.

AttributeUsage - Indicates how this attribute can or should be managed by the data storage library module.

6.2.9 CSSM_DB_RECORD_INFO

This structure defines the meta-information about an individual record type in a single data store. This information is maintained by the data storage library module for each data store it manages.

```
typedef struct {
       CSSM_DATA_RECORD_TYPE DataRecordType;
       uint32 NumberOfAttributes;
       CSSM_DB_ATTRIBUTE_INFO_PTR AttributeInfo;
} CSSM_DB_RECORD_INFO,  *CSSM_DB_RECORD_INFO_PTR;
```

Definition:

DataRecordType - A type of data record stored in this data store.

NumberOfAttributes - The number of directly or indirectly named attributes for this record type.

AttributeInfo - A pointer to an array of structures describing each attribute for this record type. Each structure provides a name for the attribute, and a usage mode for the attribute. Usage modes include indexed attributes and non-indexed attributes. An index may be one-to-many or one-to-one with the data store records. The array contains *NumberOfAttributes* entries.

6.2.10 CSSM_DBINFO

This structure defines all meta-information about an individual data store. This information is maintained by the data storage library module for each data store it manages.

```
typedef struct {
       CSSM_DB_TYPE DbType;
       uint32 NumberOfRecordTypes;
       CSSM_DB_RECORD_INFO_PTR RecordInfo;
       void *reserved;
} CSSM_DBINFO,  *CSSM_DBINFO_PTR;
```

Definition:

DbType - Indicates the user-defined semantic type of the records stored in this data store. Typically this indicates how applications should use the records in this data store.

NumberOfRecordTypes-The number of distinct record type (formats) stored in this data store.

RecordInfo - A pointer to an array of structures describing the attribute format of each record type stored in this data store. The array contains*NumberOfRecordTypes*entries.

Reserved1 - Reserved for future use.

6.2.11 CSSM_DB_CONJUNCTIVE

These are the conjunctive operations which can be used when specifying a selection criterion.

```
typedef enum cssm_db_conjunctive{
     CSSM_NONE
     CSSM_AND,
     CSSM_OR
} CSSM_DB_CONJUNCTIVE
```

6.2.12 CSSM_DB_OPERATOR

These are the logical operators which can be used when specifying a selection predicate.

```
typedef enum cssm_db_operator {
    CSSM_EQUAL,
    CSSM_NOT_EQUAL,
    CSSM_LESS_THAN,
    CSSM_GREATER_THAN
} CSSM_DB_OPERATOR
```

6.2.13 CSSM_ QUERY_TAG

This tag decides whether the data object retrieval is based on query string or selection predicates.

```
typedef enum cssm_tag_query {
    CSSM_QUERY_NONE,
    CSSM_QUERY_STRING,
    CSSM_QUERY_PREDICATES
} CSSM_QUERY_TAG
```

6.2.14 CSSM_SELECTION_PREDICATE

This structure defines the selection predicate to be used for database queries.

```
typedef struct cssm_selection_predicate {
       CSSM_DB_OPERATOR dbOperator;
       CSSM_ATTRIBUTE_NAME AttributeName;
       CSSM_DATA AttributeValue;
} CSSM_SELECTION_PREDICATE, *CSSM_SELECTION_PREDICATE_PTR
```

Definition:

dbOperator - The relational operator to be used when comparing the value contained in *AttributeValue* to values contained in the data store.

AttributeName - The name of an attribute in a data store record. This attribute is a target for comparison when selecting records from the data store.

AttributeValue - The value used in comparisons with values stored in the *AttributeName* field of data store records. If no *AttributeName* is specified, then the *AttributeValue* can be a selection predicate of any format supported by the data storage library module.

6.2.15 CSSM_QUERY_PREDICATE

This structure defines the Query predicate to be used for database queries.

```
typedef struct cssm_query_predicate {
       CSSM_DB_CONJUNCTIVE Conjunctive;
       uint32 NumSelectionPredicates;
       CSSM_SELECTION_PREDICATE_PTR SelectionPredi    cate;
} CSSM_QUERY_PREDICATE, *CSSM_QUERY_PREDICATE_PTR
```

Definition:
Conjunctive - Indicates the conjunctive to be used to join the predicates.

NumSelectionPredicates- Number of election predicates in the query.

SelectionPredicate - Pointer to the array of selection predicates.

6.2.16 CSSM_QUERY

This union contains the array of selection predicates and a CSSM_DATA_PTR.
Depending on the CSSM_QUERY_TAG, one of these has to be selected..

```
typedef union cssm_query {
       CSSM_DATA QueryString;
       CSSM_QUERY_P REDICATE_PTR QueryPredicate;
} CSSM_QUERY, *CSSM_QUERY_PTR
```

Definition:
QueryString - Pointer to query string.

QueryPredicate - Query predicate pointer which points to an array of selection predicates.

6.2.17 CSSM_INDEX_RECORD

This structure defines the index values to be inserted or deleted.

```
typedef struct cssm_index_record {
       CSSM_ATTRIBUTE_NAME IndexName;
       CSSM_DATA IndexValue;
} CSSM_INDEX_RECORD, *CSSM_INDEX_RECORD_PTR
```

Definition:

IndexName - The name of an attribute in a data store record. This attribute is a target for comparison when selecting records from the data store.

IndexValue - The value used in comparisons with values stored in the *IndexName* field of data store records. If no *IndexName* is specified, then the *IndexValue* can be any format supported by the data storage library module.

6.2.18 CSSM_DL_MODULE_TYPE

These are the module implementation types for data storage library modules.

typedef uint32 CSSM_DL_MODULE_TYPE

```
#define CSSM_STORE_LOCAL_MEMORY 0x00000001       /* implementation uses a local
                                                  memory cache */
#define CSSM_STORE_LOCAL_FILE   0x00000002       /* implement. uses a file system
                                                  service */
#define CSSM_STORE_LOCAL_DBMS   0x00000004       /* implementation uses a local
                                                  DBMS */
#define CSSM_STORE_REMOTE_DBMS  0x00000008       /* implementation uses a remote
                                                 DBMS */
#define CSSM_STORE_REMOTE_DIR   0x00000010       /* implementation uses a remote
                                                  directory service */
#define CSSM_STORE_TOKEN        0x00000020       /* implementation uses a
                                                 hardware token */
#define CSSM_STORE_REMOVEABLE   0x00000040       /* implementati on uses a
removable
                                                 store device */
#define CSSM_STORE_UNSPECIFIED  0x00000080       /* implementation is unspecified
                                                 */
```

6.2.19 CSSM_DL_ACCESS_TYPE

```
#define       CSSM_DL_STORE_ACCESS_SERIAL              CSSM_CSP_SESSION_SERIAL
#define       CSSM_DL_STORE_ACCESS_EXCLUSIVE     CSSM_CSP_SESSION_EXCLUSIVE
```

6.2.20 CSSM_DL_INFO

This structure contains all of the static data associated with a data storage library add-in module. This information is added to the CSSM registry at install time. It can be queried using the command **CSSM_DL_GetInfo ( )**

```
typedef struct cssm_dlinfo{
     uint32 VerMajor;
     uint32 VerMinor;
     CSSM_DL_MODULE_TYPE DLModuleType;
     uint32 DeviceID;
     CSSM_BOOL CapabilitiesInitialized;
     uint32 DeviceAccessFlags;
     CSSM_DATA ExclusiveDLMCertificate;
     CSSM_BOOL LoginRequired;
     uint32 NumberOfRecordTypes;
     CSSM_DATA_RECORD_TYPE_PTR DataRecordTypes;
     uint32 NumberOfAttributeUsageTypes;
     CSSM_DB_ATTRIBUTE_USAGE_PTR AttributeUsageTypes;
     uint32 NumberOfAttributeIdFormats;
     CSSM_ATTRIBUTE_ID_FORMAT_PTR AttributeIdFormats;
     uint32 NumberOfRelOperatorTypes;
     CSSM_DB_OPERATOR_PTR RelOperatorTypes;
     uint32 NumberOfConjOperatorTypes;
     CSSM_DB_CONJUNCTIVE_PTR ConjOperatorTypes;
     CSSM_DATA_PTR Reserved1;
}CSSM_DLINFO, *CSSM_DLINFO_PTR
```

Definition:

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

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

DLModuleType - Indicates the underlying implementation approach for this library module.

DeviceID - The ID of a hardware storage device managed by this data storage library module.

CapabilitiesInitialized - True or false, indicating whether complete capabilities are currently specified in this DLinfo structure.

DeviceAccessFlags - A bitmask of the device access modes supported by this library module.

ExclusiveDLMCertificate - The certificate used to sign certificates issued to exclusive users of this data storage library module.

LoginRequired - True or false, indicating whether a DL requires caller login and logout.

NumberOfRecordTypes - The number of distinct data record types that can be stored and managed by this library module in one or more data stores.

DataRecordTypes - An array listing the data record types that can be stored and managed by this library module in one or more data stores. The array contains *NumberOfRecordTypes* entries.

NumberOfAttributeUsageTypes - The number of distinct attribute usages supported by a DL.

AttributeUsageTypes - An array listing the attribute usages supported by the data storage library when defining the schema for a new data store. Currently-defined usages include: indexed attribute, unique-index attribute, and non-indexed attribute. The array contains *NumberOfAttributeUsageTypes* entries.

NumberOfAttributeIdFormats - The number of distinct attribute identification formats that are supported by this data storage library module.

AttributeIdFormats - An array listing the formats accepted by the data storage library to identify record fields in a selection query. Currently-defined usages include: OID-name format and string-name format. The array contains *NumberOfAttributeIdFormats* entries.

NumberOfRelOperatorTypes - The number of distinct binary relational operators supported by this library module.

RelOperatorTypes - An array listing the relational operators supported by this library module for defining selection predicates for retrieving stored data objects. The array contains *NumberOfRelOperatorTypes* entries.

NumberOfConjOperatorTypes - The number of distinct conjunctive operators supported by this library module.

ConjOperatorTypes - An array listing the conjunctive operators supported by this library module for defining selection predicates for retrieving stored data objects. The array contains *NumberOfConjOperatorTypes* entries.

Reserved1 - Reserved for future use.

6.3 Data storage Data Structures

6.3.1 CSSM_DL_DbOpen

CSSM_DB_HANDLE CSSMAPI CSSM_DL_DbOpen (CSSM_DL_HANDLE DLHandle,
const char *DbName)

This function opens the data store with the specified logical name.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DbName (input)
A pointer to the string containing the logical name of the data store.

Return Value

Returns the CSSM_DB_HANDLE of the opened data store. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_DATASTORE_NOT_EXISTS	The data store with the logical name does not exist
CSSM_DL_DB_OPEN_FAIL	Open caused an exception
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

CSSM_DL_DbClose.

6.3.2 CSSM_DL_DbClose

CSSM_RETURN CSSMAPI CSSM_DL_DbClose (CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle)

This function closes an open data store.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_DB_CLOSE_FAIL	Close caused an exception

See Also

CSSM_DL_DbOpen.

6.3.3 CSSM_DL_DbCreate

CSSM_DB_HANDLE CSSMAPI CSSM_DL_DbCreate
(CSSM_DL_HANDLE DLHandle,
CSSM_TP_HANDLE TPHandle,
const char *DbName,
CSSM_MODULE_HANDLE_PTR ModuleHandles,
CSSM_DBINFO_PTR DataStoreSchema)

This function creates and opens a new data store. The name of the new data store is specified by the input parameter DbName. The record schema for the data store is specified in the DBINFO structure.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module used to perform this function.

TPHandle (input)
The handle that describes the module to be used to determine the semantic, user-defined type to be associated with the new data store.

DbName (input)
The general, external name for the new data store.

ModuleHandles (input)
An array of handles, one per record type to be stored in this data store. Each handle describes a service provider module that can be invoked to indirectly access values contained in records that will be stored in this new data store. The array must contain *DataStoreSchema.NumberOfRecordTypes* entries. Handles corresponding to attributes named by user-defined strings rather than by library-processed OIDs are ignored.

DataStoreSchema (input)
A pointer to a structure describing the format/schema of each record type that will be stored in the new data store.

Return Value

A handle to the newly created, open data store. When NULL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_CSP_HANDLE	Invalid Cryptographic Service Provider handle
CSSM_DL_INVALID_TP_HANDLE	Invalid Trust Policy Module handle
CSSM_DL_INVALID_MODULE_HANDLE	Invalid Module handle for accessing the data record
CSSM_DL_INVALID_DATA_PTR	Invalid data pointer
CSSM_DL_DB_CREATE_FAIL	Create caused an exception
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

CSSM_DL_DbOpen, CSSM_DL_DbClose, CSSM_DL_DbDelete.

6.3.4 CSSM_DL_DbDelete

CSSM_RETURN CSSMAPI CSSM_DL_DbDelete (CSSM_DL_HANDLE DLHandle,
const char *DbName)

This function deletes all records from the specified data store and removes all state information associated with that data store.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DbName (input)
A pointer to the string containing the logical name of the data store.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_DB_DELETE_FAIL	Delete caused an exception

See Also

CSSM_DL_DbCreate, CSSM_DL_DbOpen, CSSM_DL_DbClose.

6.3.5 CSSM_DL_DbImport

CSSM_RETURN CSSMAPI CSSM_DL_DbImport (CSSM_DL_HANDLE DLHandle,
const char *DbDestLogicalName,
const char *DbSrcFileName)

This function imports data store records from a file into a new data store.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DbDestLogicalName (input)
The name of the destination data store in which to insert the records.

DbSrcFileName(input)
The name of the source file from which to obtain the records that are added to the data store.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_PTR	NULL source or destination file names
CSSM_DL_DB_IMPORT_FAIL	DB exception doing import function
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

CSSM_DL_DbExport.

6.3.6 CSSM_DL_DbExport

CSSM_RETURN CSSMAPI CSSM_DL_DbExport (CSSM_DL_HANDLE DLHandle,
const char *DbSrcLogicalName,
const char *DbDestFileName)

This function exports a copy of the data store records from the source data store to a file.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DbSrcLogicalName (input)
The name of the data store from which the records are to be exported.

DbDestFileName(input)
The name of the destination file which will contain a copy of the source data store's records.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_PTR	NULL source or destination file names
CSSM_DL_DB_EXPORT_FAIL	DB exception doing export function
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

CSSM_DL_DbImport.

6.3.7 CSSM_DL_DbSetInfo

CSSM_RETURN CSSMAPI CSSM_DL_DbSetInfo (CSSM_DL_HANDLE DLHandle,
const char* DbName,
CSSM_DBINFO_PTR DbInfo)

This function sets information describing the datasource.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module used to perform this function.

DbName(input)
A pointer to the string containing the data source name.

DbInfo
A pointer to CSSM_DBINFO structure.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_POINTER	Invalid pointer
CSSM_REGISTRY_ERROR	Unable to write to registry
CSSM_ INVALID_DBINFO_ POINTER	Invalid CSSM_DBINFO pointer

See Also

CSSM_DL_GetInfo, CSSM_DL_FreeDbInfo.

6.3.8 CSSM_DL_DbGetInfo

CSSM_DBINFO_PTR CSSMAPI CSSM_DL_DbGetInfo (CSSM_DL_HANDLE DLHandle,
const char* DbName)

This function returns information describing a DL module and its capabilities.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module used to perform this function.

DbName(input)
A pointer to the string containing the data source name.

Return Value

A pointer to an array of one or more CSSM_DLINFO structures containing information about a DL module. There is one Dlinfo structure for each distinct hardware storage device managed by this DL. Most DLs manage only one device. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_MEMORY_ERROR	Internal memory error

See Also

CSSM_DL_DbSetInfo, CSSM_DL_DbFreeDbInfo.

6.3.9 CSSM_DL_FreeDbInfo

CSSM_RETURN CSSMAPI CSSM_DL_FreeDbInfo (CSSM_DBNFO_PTR DbInfo,
uint32 NumberOfDbInfos)

This function frees the memory allocated by a DL module for the CSSM_DLINFO structure returned by the CSSM_DL_GetInfo function.

Parameters

DLInfo (input)
A pointer to CSSM_DL_Info structure.

numberOfInfos (input)
The number of DL Info structures to be freed.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_ INVALID_DLINFO_ POINTER	Invalid CSSM_DLINFO pointer

See Also

CSSM_DL_DbGetInfo.

6.3.10 CSSM_DL_GetDbHandleToName

CSSM_NAME_LIST_PTR CSSMAPI CSSM_DL_GetDbHandleToName
(CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DbHandle)

This function retrieves the data source name corresponding to an opened database handle. A DL module is responsible for allocating the memory required for the list.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DbHandle (input)
The handle to an opened database.

Return Value

Returns a pointer to a CSSM_NAME_LIST structure which contains a list of data store names. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_MEMORY_ERROR	Error allocating memory
CSSM_DL_INVALID_DB_HANDLE	Invalid DB Handle
CSSM_DL_INVALID_DL_HANDLE	Invalid DL Handle

See Also

CSSM_DL_FreeNameList.

6.4 Generic Security Object Storage Operations

6.4.1 CSSM_DL_DataInsert

CSSM_RETURN CSSMAPI CSSM_DL_DataInsert
(CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_MODULE_HANDLE_PTR ModuleHandles,
CSSM_DATA_RECORD_TYPE DataRecordType,
CSSM_INDEX_RECORD_PTR IndexRecord,
const CSSM_DATA_PTR DataRecord)

This function makes the DataRecord persistent by inserting it into the specified data store. The DataRecord contains two logically distinct sets of data values: immutable values and update-able values. The ModuleHandle specifies the library a DL should use to access sub-fields of the data record as required during the add process.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

ModuleHandles (input/optional)
An array of handles, one per record type to be stored in this data store. Each handle describes a service provider module that can be invoked to indirectly access values contained in records that will be stored in this new data store. The array must contain *DataStoreSchema.NumberOfRecordTypes* entries. Handles corresponding to attributes named by user-defined strings rather than by library-processed OIDs are ignored.

DataRecordType (input)
Indicates the type of data record being added to the data store. The value cannot be CSSM_DATA_RECORD_ANY.

IndexRecord (input/Optional)
Indicates the values for the meta data for the record if module handle is NULL

DataRecord (input)
A pointer to the CSSM_DATA structure which contains the data to be added to the data store.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid data storage library handle
CSSM_DL_INVALID_DB_HANDLE	Invalid Data Store handle
CSSM_DL_INVALID_MODULE_HANDLE	Invalid Module handle for accessing the data record

CSSM_DL_INVALID_DATA_PTR	Invalid data pointer
CSSM_DL_DATA_INSERT_FAIL	Add caused an exception

See Also

CSSM_DL_DataDelete.

6.4.2 CSSM_DL_DataDelete

CSSM_RETURN CSSMAPI CSSM_DL_DataDelete
(CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_MODULE_HANDLE_PTR ModuleHandles,
CSSM_DATA_RECORD_TYPE DataRecordType,
CSSM_INDEX_RECORD_PTR IndexRecord,
const CSSM_DATA_PTR DataRecord)

This function removes the DataRecord of the specified DataRecordType from the specified data store.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

ModuleHandles (input/Optional)
An array of handles, one per record type to be stored in this data store. Each handle describes a service provider module that can be invoked to indirectly access values contained in records that will be stored in this new data store. The array must contain *DataStoreSchema.NumberOfRecordTypes* entries. Handles corresponding to attributes named by user-defined strings rather than by library-processed OIDs are ignored.

DataRecordType (input)
Indicates the type of data record being deleted from the data store.

IndexRecord (input/Optional)
Indicates the values for the meta data for the record if module handle is NULL

DataRecord (input)
A pointer to the CSSM_DATA structure which contains the data to be deleted from the data store.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid data storage library handle
CSSM_DL_INVALID_DATA_PTR	Invalid data pointer

CSSM_DL_INVALID_DB_HANDLE	Invalid Data Storage handle
CSSM_DL_DATA_NOT_IN_DB	Data record not in Data Store
CSSM_DL_DATA_DELETE_FAIL	Delete caused an exception

See Also

CSSM_DL_DataInsert.

6.4.3 CSSM_DL_DataGetFirst

CSSM_DATA_PTR CSSMAPI CSSM_DL_DataGetFirst
(CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_DATA_RECORD_TYPE DataRecordType,
CSSM_QUERY_TAG QueryTag,
CSSM_QUERY_PTR Query,
CSSM_HANDLE_PTR ResultsHandle,
uint32 *NumberOfMatchedDataRecords)

This function retrieves the first object in the data store that matches the selection criteria. The selection criteria is expressed in one of two ways: a predicate formed from a set of triples (OID, value, relational operator) connected by a single conjunctive operator, or a single string whose required substructure and semantics are defined by the implementing module. This function returns a count of the total number of objects matching the selection criteria, the first object matching the criteria, and a selection handle that may be used to retrieve the subsequent objects matching the selection criteria.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

DataRecordType (input)
A type of data records to be searched by this query.

QueryTag (input/Optional)
Indicates whether a Query string or an array of seletion predicates is used for the query.

Query (input/Optional)
If the Query Tag is Query_String, CSSM_DATA_PTR is used as query string, otherwise CSSM_SELECTIONA_PREDICATE_PTR is used to build the query string.

ResultsHandle (output)
This handle should be used to retrieve subsequent records that matched this selection criteria.

NumberOfMatchedDataRecords (output)
Returns the total number of data records that match the selection criteria.

Return Value

Returns a pointer to a CSSM_DATA structure which contains the first data record in the data store that matches the selection criteria. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_SELECTION_PTR	Invalid selection predicate pointer
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_NO_DATA_FOUND	No data records match the selection predicate
CSSM_DL _DATA_GETFIRST_FAIL	Opening the records caused an exception
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

CSSM_DL_DataGetNext, CSSM_DL_DataAbortQuery.

6.4.4 CSSM_DL_DataGetNext

CSSM_DATA_PTR CSSMAPI CSSM_DL_DataGetNext
(CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_HANDLE ResultsHandle,
CSSM_DATA_RECORD_TYPE_PTR DataRecordType)

This function returns the next object referenced by the ResultsHandle. The ResultsHandle was returned by the DataGetFirst functions.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

ResultsHandle(input)
The selection handle returned from the DL_DataGetFirst function.

DataRecordType (output)
The type of the data record returned by this query.

Return Value

Returns a pointer to a CSSM_DATA structure which contains the next data record in the data store that matches the original selection criteria. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid data storage library handle
CSSM_DL_INVALID_RESULTS_HANDLE	Invalid query handle
CSSM_DL_INVALID_DB_HANDLE	Invalid Data Store handle
CSSM_DL_NO_MORE_RECORDS	No more records for this selection handle
CSSM_DL_DATA_GETNEXT_FAIL	Opening the records caused an exception
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

CSSM_DL_DataGetFirst, CSSM_DL_DataAbortQuery.

6.4.5 CSSM_DL_DataAbortQuery

CSSM_RETURN CSSMAPI CSSM_DL_DataAbortQuery (CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_HANDLE ResultsHandle)

This function terminates the query initiated by DL_DataGetFirst, and allows a DL to release all intermediate state information associated with the query.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module used to perform this function.

DBHandle (input)
The handle that describes the data store associated with the query.

ResultsHandle (input)
The selection handle returned from the initial query function.

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_DL_INVALID_DL_HANDLE	Invalid data storage library Handle
CSSM_DL_INVALID_DB_HANDLE	Invalid data store handle
CSSM_DL_INVALID_RESULTS_HANDLE	Invalid results handle
CSSM_DL_DATA_ABORT_QUERY_FAIL	Unable to abort query

See Also

CSSM_DL_DataGetFirst, CSSM_DL_DataGetNext.

6.5 Certificate Storage Operations

6.5.1 CSSM_DL_CertInsert

CSSM_RETURN CSSMAPI CSSM_DL_CertInsert (CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_CL_HANDLE CLHandle,
const CSSM_DATA_PTR Cert)

This function makes the certificate persistent by inserting it into the specified data store.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

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

Cert (input)
A pointer to the CSSM_DATA structure which contains the certificate to be added to the data store.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_CERTIFICATE_PTR	Invalid certificate pointer
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_CERT_INSERT_FAIL	Add caused an exception

See Also

CSSM_DL_CertDelete.

6.5.2 CSSM_DL_CertDelete

CSSM_RETURN CSSMAPI CSSM_DL_CertDelete (CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_CL_HANDLE CLHandle,
const CSSM_DATA_PTR Cert)

This function removes the certificate from the specified data store.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

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

Cert (input)
A pointer to the CSSM_DATA structure which contains the certificate to be deleted from the data store.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_CERTIFICATE_PTR	Invalid certificate pointer
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_CERTIFICATE_NOT_IN_DB	Certificate not in DB
CSSM_DL_CERT_DELETE_FAIL	Delete caused an exception

See Also

CSSM_DL_CertInsert.

6.5.3 CSSM_DL_CertRevoke

CSSM_RETURN CSSMAPI CSSM_DL_CertRevoke (CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
const CSSM_DATA_PTR CertToBeRevoked)

This function makes persistent the knowledge that this certificate has been revoked.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

CertToBeRevoked (input)
A pointer to the CSSM_DATA structure which contains the certificate to be marked as revoked.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_CERTIFICATE_PTR	Invalid certificate pointer
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_CERT_REVOKE_FAIL	Update caused an exception

6.5.4 CSSM_DL_CertGetFirst

CSSM_DATA_PTR CSSMAPI CSSM_DL_CertGetFirst
(CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_SELECTION_PREDICATE_PTR SelectionPredicate,
uint32 SizeSelectionPredicate,
CSSM_DB_CONJUNCTIVE Conjunctive,
CSSM_HANDLE_PTR ResultsHandle,
uint32 *NumberOfMatchedCerts)

This function locates the first certificate in the data store which matches the selection criteria. The selection criteria is the expression formed by connecting all of the relational expressions of the selection predicate array using the one conjunctive operator. This function returns a count of the total number of certificates matching the selection criteria, the first certificate matching the criteria, and a selection handle that may be used to retrieve the subsequent certificates matching the selection criteria.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

SelectionPredicate (input)
A pointer to a CSSM_SELECTION_PREDICATE array which contains field-value/operator pairs.
If NULL, the first certificate in the data store is returned.

SizeSelectionPredicate(input)
The size of the selection predicate array.

Conjunctive (input)
The Boolean operator used to connect the selection predicates. If the selection predicate is null, this parameter is ignored.

ResultsHandle (output)
This handle should be used for subsequent retrievals for the same selection criteria.

NumberOfMatchedCerts (output)
Returns the total number of certificates that match the selection criteria.

Return Value

Returns a pointer to a CSSM_DATA structure which contains the first certificate in the data store that matches the selection criteria. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_SELECTION_PTR	Invalid selection predicate pointer
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_NO_CERTIFICATE_FOUND	No certificates that match the selection predicate
CSSM_DL _CERT_GETFIRST_FAIL	Opening the records caused an exception
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

CSSM_DL_CertGetNext, CSSM_DL_CertAbortQuery.

6.5.5 CSSM_DL_CertGetNext

CSSM_DATA_PTR CSSMAPI CSSM_DL_CertGetNext (CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_HANDLE ResultsHandle)

This function returns the next certificate matching the selection criteria used to establish the ResultsHandle.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

ResultsHandle(input)
The selection handle obtained from the DL_CertGetFirst function call.

Return Value

Returns a pointer to a CSSM_DATA structure which contains the next certificate in the data store that matches the selection criteria. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_RESULTS_HANDLE	Invalid query handle
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_NO_MORE_CERTS	No more certificates for that selection handle
CSSM_DL_CERT_GETNEXT_FAIL	Opening the records caused an exception
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

CSSM_DL_CertGetFirst, CSSM_DL_CertAbortQuery.

6.5.6 CSSM_DL_CertAbortQuery

CSSM_RETURN CSSMAPI CSSM_DL_CertAbortQuery (CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_HANDLE ResultsHandle)

This function terminates the query initiated by DL_CertGetFirst and allows a DL to release all intermediate state information associated with the query.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module used to perform this function.

DBHandle (input)
The handle that describes the data store associated with the query.

ResultsHandle (input)
The selection handle returned from the DL_CertGetFirst function.

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_DL_INVALID_DL_HANDLE	Invalid data storage library Handle
CSSM_DL_INVALID_DB_HANDLE	Invalid data store handle
CSSM_DL_INVALID_RESULTS_HANDLE	Invalid results handle
CSSM_DL_CERT_ABORT_QUERY_FAIL	Unable to abort query

See Also

CSSM_DL_CertGetFirst, CSSM_DL_CertGetNext.

6.6 CRL Storage Operations

6.6.1 CSSM_DL_CrlInsert

CSSM_RETURN CSSMAPI CSSM_DL_CrlInsert (CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_CL_HANDLE CLHandle,
const CSSM_DATA_PTR Crl)

This function makes the CRL persistent by inserting it into the specified data store.

Parameters

DLHandle (input)
The handle that describes the add-in data store library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

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

Crl (input)
A pointer to the CSSM_DATA structure which contains the CRL to be added to the data store.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_CRL_PTR	Invalid CRL pointer
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_CRL_INSERT_FAIL	Add caused an exception

See Also

CSSM_DL_CrlDelete

6.6.2 CSSM_DL_CrlDelete

CSSM_RETURN CSSMAPI CSSM_DL_CrlDelete (CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_CL_HANDLE CLHandle,
const CSSM_DATA_PTR Crl)

This function removes the CRL from the specified data store.

Parameters

DLHandle (input)
The handle that describes the add-in data store library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

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

Crl (input)
A pointer to the CSSM_DATA structure which contains the CRL to be removed from the data store.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_CRL_PTR	Invalid CRL pointer
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_CRL_NOT_IN_DB	CRL not in DB
CSSM_DL_CRL_DELETE_FAIL	Delete caused an exception

See Also

CSSM_DL_CrlInsert.

6.6.3 CSSM_DL_CrlGetFirst

CSSM_DATA_PTR CSSMAPI CSSM_DL_CrlGetFirst
(CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_SELECTION_PREDICATE_PTR SelectionPredicate,
uint32 SizeSelectionPredicate,
CSSM_DB_CONJUNCTIVE Conjunctive,
CSSM_HANDLE_PTR ResultsHandle,
uint32 *NumberOfMatchedCrls)

This function locates the first CRL in the data store which matches the selection criteria. The selection criteria is the expression formed by connecting all of the relational expressions of the selection predicate array using the one conjunctive operator. This function returns a count of the total number of CRLs matching the selection criteria, the first CRL matching the criteria, and a selection handle that may be used to retrieve the subsequent CRLs matching the selection criteria.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

SelectionPredicate (input)
A pointer to a CSSM_SELECTION_PREDICATE array which contains field-value/operator pairs. If NULL, the first CRL in the data store is returned.

SizeSelectionPredicate(input)
The size of the selection predicate array.

Conjunctive (input)
The Boolean operator used to connect the selection predicates. If the selection predicate is null, this parameter is ignored.

ResultsHandle (output)
This handle should be used for subsequent retrievals for the same selection criteria.

NumberOfMatchedCrls (output)
Returns the total number of CRLs that match the selection criteria.

Return Value

Returns a pointer to a CSSM_DATA structure which contains the first CRL in the data store that matches the selection criteria. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_SELECTION_PTR	Invalid selection predicate pointer
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_NO_CRL_FOUND	No Crls that match the selection predicate
CSSM_DL_CRL_GET_FIRST_FAIL	Get first caused an exception
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

CSSM_DL_CrlGetNext, CSSM_DL_CrlAbortQuery.

6.6.4 CSSM_DL_CrlGetNext

CSSM_DATA_PTR CSSMAPI CSSM_DL_CrlGetNext (CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_HANDLE ResultsHandle)

This function returns the next certificate which matches the selection criteria used to establish the ResultsHandle.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

ResultsHandle(input)
The selection handle obtained from the DL_CrlGetFirst function call.

Return Value

Returns a pointer to a CSSM_DATA structure which contains the next CRL in the data store that matches the selection criteria. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_RESULTS_HANDLE	Invalid query handle
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_NO_MORE_CRLS	No more Crls for that selection handle
CSSM_DL_CRL_GET_NEXT_FAIL	Opening the records caused an exception
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

CSSM_DL_CrlGetFirst, CSSM_DL_CrlAbortQuery.

6.6.5 CSSM_DL_CrlAbortQuery

CSSM_RETURN CSSMAPI CSSM_DL_CrlAbortQuery (CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_HANDLE ResultsHandle)

This function terminates the query initiated by DL_CrlGetFirst and allows a DL to release all intermediate state information associated with the query.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module used to perform this function.

DBHandle (input)
The handle that describes the data store associated with the query.

ResultsHandle (input)
The selection handle returned from the DL_CrlGetFirst function.

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_DL_INVALID_DL_HANDLE	Invalid data storage library Handle
CSSM_DL_INVALID_DB_HANDLE	Invalid data store handle
CSSM_DL_INVALID_RESULTS_HANDLE	Invalid results handle
CSSM_DL_CRL_ABORT_QUERY_FAIL	Unable to abort query

See Also

CSSM_DL_CrlGetFirst, CSSM_DL_CrlGetNext.

6.7 Module Management Functions

6.7.1 CSSM_DL_Install

CSSM_RETURN CSSMAPI CSSM_DL_Install
(const char *DLName,
const char *DLFileName,
const char *DLPathName,
const CSSM_GUID_PTR GUID,
const CSSM_DLINFO_PTR DLInfo,
const void *Reserved1,
const CSSM_DATA_PTR Reserved2)

This function updates the CSSM-persistent internal information about a DL module.

Parameters

DLName (input)
The name of the data storage library module to be installed.

DLFileName (input)
The name of the file that contains the data storage library implementation.

DLPathName (input)
The path to the file that implements the data storage library.

GUID (input)
A pointer to the CSSM_DATA structure containing the global unique identifier for a DL module.

DLInfo (input)
A pointer to the CSSM_DLINFO structure containing information about a DL module.

Reserved1
Reserved data for the function.

Reserved2
Reserved data for the function.

Return Value

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

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_REGISTRY_ERROR	Error in writing registry

See Also

CSSM_DL_Uninstall.

6.7.2 CSSM_DL_Uninstall

CSSM_RETURN CSSMAPI CSSM_DL_Uninstall (const CSSM_GUID_PTR GUID)

This function deletes the CSSM-persistent internal information about a DL module.

Parameters

GUID (input)
A pointer to the CSSM_DATA structure containing the global unique identifier for a DL module.

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_POINTER	Invalid pointer
CSSM_REGISTRY_ERROR	Error in writing registry

See Also

CSSM_DL_Install.

6.7.3 CSSM_DL_ListModules

CSSM_LIST_PTR CSSMAPI CSSM_DL_ListModules (void)

This function returns a list containing the GUID/name pair for each of the currently-installed DL modules.

Parameters

None

Return Value

A pointer to the CSSM_LIST structure containing the names of DL modules. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_MEMORY_ERROR	Error in memory allocation

See Also

CSSM_FreeList.

6.7.4 CSSM_DL_Attach

CSSM_DL_HANDLE CSSMAPI CSSM_DL_Attach
(const CSSM_GUID_PTR GUID,
uint32 CheckCompatibleVerMajor,
uint32 CheckCompatibleVerMinor,
const CSSM_API_MEMORY_FUNCS_PTR MemoryFuncs,
const uint32 DeviceID,
const uint32 DeviceAccessFlags,
uint32 Application,
const CSSM_NOTIFY_CALLBACK Notification,
const void *Reserved)

This function attaches the application with a DL module. A DL module tests for compatibility with the version specified.

Parameters

GUID (input)
A pointer to the CSSM_DATA structure containing the global unique identifier for a DL module.

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

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

MemoryFuncs (input)
The caller's memory allocation and deallocation functions that can be jointly used by a DL and the caller to manage a common memory pool.

DeviceID (input)
Device ID number of the target hardware storage device. This value should always be taken from the CSSM_DLINFO structure to insure that a compatible identifier is used. (Software-only implementations can always use zero for this value.)

DeviceAccessFlags(input)
Bitmask of default access modes. Legal values are defined in the table below. DL service providers may or may not support all combinations of access modes.

Application(input/optional)
Passed to the application when its callback is invoked allowing the application to determine the proper context of operation.

Notification (input/optional)
Callback provided by the application that is called by a DL when one of three things takes place: a parallel operation completes, a token running in serial mode surrenders control to the application, or the token is removed (hardware specific).

Reserved
A reserved input.

Valid *DeviceAccessFlags* Values

Value	Description
CSSM_DL_STORE_ACCESS_SERIAL	All storage accesses are in serial mode
CSSM_ DL_STORE _ACCESS_EXCLUSIVE	Storage access is exclusive to this caller

Return Value

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

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_MEMORY_ERROR	Internal memory error
CSSM_INCOMPATIBLE_VERSION	Incompatible version
CSSM_ATTACH_FAIL	Unable to attach to DL module

See Also

CSSM_DL_Detach.

6.7.5 CSSM_DL_Detach

CSSM_RETURN CSSMAPI CSSM_DL_Detach (CSSM_DL_HANDLE DLHandle)

This function detaches the application from a DL module.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be detached.

Return Value

A CSSM_OK return value signifies that a DL module has been detached. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_ADDIN_HANDLE	Invalid DL handle

See Also

CSSM_DL_Attach.

6.7.6 CSSM_DL_GetInfo

CSSM_DLINFO_PTR CSSMAPI CSSM_DL_GetInfo (const CSSM_GUID_PTR GUID,
CSSM_BOOL CompleteCapabilitiesOnly,
uint32 *NumberOfInfos)

This function returns information describing a DL module and its capabilities.

Parameters

GUID (input)
A pointer to the CSSM_DATA structure containing the global unique identifier for a DL module.

CompleteCapabilitiesOnly (input)
Boolean value indicating whether or not partially-specified capabilities should be returned. If set to TRUE only a completely-specified capability should be returned. If set to false, the registered structure should be returned regardless of the completeness of its current state.

NumberOfInfos (output)
The number of DLinfo structures returned by the execution of this function.

Return Value

A pointer to an array of one or more CSSM_DLINFO structures containing information about a DL module. There is one Dlinfo structure for each distinct hardware storage device managed by this DL. Most DLs manage only one device. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_MEMORY_ERROR	Internal memory error
CSSM_INVALID_GUID	No known DL module with specified GUID

See Also

CSSM_DL_FreeInfo.

6.7.7 CSSM_DL_FreeInfo

CSSM_RETURN CSSMAPI CSSM_DL_FreeInfo (CSSM_DLINFO_PTR DLInfo,
uint32 NumberOfInfos)

This function frees the memory allocated by a DL module for the CSSM_DLINFO structure returned by the CSSM_DL_GetInfo function.

Parameters

DLInfo (input)
A pointer to CSSM_DL_Info structure.

numberOfInfos (input)
The number of DL Info structures to be freed.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_ INVALID_DLINFO_ POINTER	Invalid CSSM_DLINFO pointer

See Also

CSSM_DL_GetInfo.

6.7.8 CSSM_DL_GetDbNames

CSSM_NAME_LIST_PTR CSSMAPI CSSM_DL_GetDbNames (CSSM_DL_HANDLE DLHandle)

This function returns a list of the logical data store names that the specified DL module can access and a count of the number of logical names in that list.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

Return Value

Returns a pointer to a CSSM_NAME_LIST structure which contains a list of data store names. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_MEMORY_ERROR	Error allocating memory
CSSM_DL_NO_DATA_SOURCES	No known data store names
CSSM_DL_GET_DB_NAMES_FAIL	Get DB Names failed
CSSM_DL_INVALID_DL_HANDLE	Invalid DL Handle

See Also

CSSM_DL_FreeNameList.

6.7.9 CSSM_DL_FreeNameList

CSSM_RETURN CSSMAPI CSSM_DL_FreeNameList (CSSM_DL_HANDLE DLHandle,
CSSM_NAME_LIST_PTR NameList)

This function frees the list of the logical data store names that was returned by DL_GetDbNames ( ).

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

NameList (input)
A pointer to the CSSM_NAME_LIST.

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_DL_MEMORY_ERROR	Error allocating memory
CSSM_DL_INVALID_PTR	Invalid pointer to the name list
CSSM_DL_INVALID_DL_HANDLE	Invalid DL Handle

See Also

CSSM_DL_GetDbNames.

6.8 Extensibility Functions

6.8.1 CSSM_DL_PassThrough

CSSM_DATA_PTR CSSMAPI DL_PassThrough (CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
uint32 PassThroughId,
const CSSM_DATA_PTR InputParams)

This function allows applications to call data storage library module-specific operations that have been exported. Such operations may include queries or services that are specific to the domain represented by a DL module.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

PassThroughId (input)
An identifier assigned by a DL module to indicate the exported function to perform.

InputParams (input)
A pointer to the CSSM_DATA structure containing parameters to be interpreted in a function-specific manner by the requested DL module. This parameter can be used as a pointer to an array of CSSM_DATA_PTRs.

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_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_INVALID_PASSTHROUGH_ID	Invalid passthrough ID
CSSM_DL_INVALID_ PTR	Invalid pointer
CSSM_DL_ PASS_THROUGH_FAIL	DB exception doing passthrough function
CSSM_DL_MEMORY_ERROR	Error in allocating memory

7. Appendix A. CSSM Error-Handling

7.1 Introduction

This section presents a specification for error handling in CSSM that provides a consistent mechanism across all layers of CSSM for returning errors to the caller.

All CSSM API functions will return one of the following:

1. CSSM_RETURN - an enumerated type consisting of CSSM_OK and CSSM_FAIL. If it is CSSM_FAIL, an error code indicating the reason for failure can be obtained by calling CSSM_GetError ( ).
2. CSSM_BOOL - an enumerated type consisting of CSSM_TRUE and CSSM_FALSE. If it is CSSM_FALSE, an error code may be available (but not always) by calling CSSM_GetError.
3. A pointer to a data structure, a handle, a file size or whatever is logical for the function to return. An error code may be available (but not always) by calling CSSM_GetError.

Check documentation for individual functions to determine if error information will be available and what error values the function uses. Note that there will be additional error values defined by add-in modules. The information available from CSSM_GetError will include both the error number and a GUID (global unique ID) that will associate the error with the add-in module that set it. The GUID of each add-in module can be obtained by calling CSSM_XX_ListModules (where XX = CSP, CL, DL, or TP). CSSM_CompareGuids can then be called to determine from which module an error came.

Each add-in module must have a mechanism for reporting their errors to the calling application. In general, there are two types of errors an add-in module can return:

- Errors CSSM has defined for it to use (CSSM_CSP_INVALID_SECURITY_LIST)
- Errors particular to an add-in module (XXX_CSP_BAD_HW_TOKEN_SERIAL_NUMBER)

Since some errors are predefined by CSSM, those errors have a set of pre-defined numeric values which are reserved by CSSM, and cannot be used arbitrarily by add-in modules. For errors which are particular to an add-in module, a different set of predefined values has been reserved for their use.

It will be up to the calling application to determine how to handle the error returned by CSSM_GetError ( ). Detailed descriptions of the error values will be available in the corresponding specification, the cssmerr.h header file, and the documentation for specific add-in modules. If a routine does not know how to handle the error, it may choose to pass the error on up the chain to its caller.

Error values should not be overwritten, if at all possible. For example, if a CSP call returns an error indicating that it could not encrypt the data, the caller should not overwrite it with an error simply indicating that the CSP failed, as it destroys valuable error handling and debugging information. For example, after a call to CL module function, the error could actually be a CSP error.

7.2 Data Structures

```
typedef enum cssm_bool {
        CSSM_FALSE = 0,
        CSSM_TRUE = 1,
} CSSM_BOOL

typedef enum cssm_return {
        CSSM_OK = 0,
        CSSM_FAIL = -1
} CSSM_RETURN

typedef struct cssm_error {
        uint32          error;
        CSSM_GUID       guid;
} CSSM_ERROR, *CSSM_ERROR_PTR
```

7.3 Error Codes

Below is a tentative list of error codes. This list is not complete, but is to serve as a representation of errors returned by CSSM and its add-in modules. Each module in CSSM has its own range of error values as defined below. Given an error value, the module to which it belongs can be determined by a series of macro calls (see CSSM_IsCSSMError, CSSM_IsCLError, CSSM_IsDLError, CSSM_IsTPError, CSSM_IsCSPError). When a call to CSSM_SetError is made, the value being passed will be checked to ensure that it falls within one of the ranges below.

7.3.1 CSSM Error Codes

7.3.1.1 Core Errors

CSSM_INVALID_POINTER	Invalid pointer
CSSM_INCOMPATIBLE_VERSION	Incompatible version
CSSM_MEMORY_ERROR	Error in allocating memory
CSSM_NOT_INITIALIZE	CSSM has not been initialized
CSSM_VERIFY_COMPONENTS_FAILED	Unable to verify components
CSSM_INTEGRITY_COMPROMISED	Integrity check failed

7.3.1.2 Common Function Errors

CSSM_INVALID_POINTER	Invalid pointer
CSSM_INVALID_ADDIN_HANDLE	Invalid add-in handle
CSSM_MEMORY_ERROR	Internal memory error

7.3.2 CSP Error Codes

7.3.2.1 Cryptographic Context Operation Errors

CSSM_INVALID_CSP_HANDLE	Invalid provider handle
CSSM_MEMORY_ERROR	Internal memory error

CSSM_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_INVALID_CONTEXT_POINTER	Invalid context pointer
CSSM_NO_WRAPPING_INFORMATION	No pass phrase or wrapping key supplied
CSSM_INVALID_POINTER	Invalid pointer to attributes

7.3.2.2 Cryptographic sessions and Logon Errors

CSSM_CSP_INVALID_CSP_HANDLE	Invalid CSP handle
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_INVALID_PASSWORD	Invalid password
CSSM_CSP_ALREADY_LOGGED_IN	User attempted to log in more than once
CSSM_CSP_NOT_LOGGED_IN	No login session existed
CSSM_CSP_INVALID_PASSWORD	Old password is invalid

7.3.2.3 Cryptographic Operation Errors

CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_NO_METHOD	Service not provided
CSSM_CSP_QUERY_SIZE_FAILED	Unable to query size
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer
CSSM_CSP_INVALID_DATA_COUNT	Invalid data count
CSSM_CSP_SIGN_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_SIGN_NO_METHOD	Service not provided
CSSM_CSP_SIGN_FAILED	Sign failed
CSSM_CSP_PRIKEY_NOT_FOUND	Cannot find the corresponding private key
CSSM_CSP_PASSWORD_INCORRECT	Password incorrect
CSSM_CSP_UNWRAP_FAILED	Unwrapped the private key failed
CSSM_CSP_NOT_ENOUGH_BUFFER	The output buffer is not big enough
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_VECTOROFBUFS_UNSUPPORTED	Supports only a single buffer of input
CSSM_CSP_SIGN_INIT_FAILED	Staged sign initialize function failed
CSSM_CSP_STAGED_OPERATION_UNSUPPORTED	Supports only single stage operations
CSSM_CSP_SIGN_UPDATE_FAILED	Staged sign update function failed
CSSM_CSP_SIGN_FINAL_FAILED	Staged sign final function failed
CSSM_CSP_VERIFY_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_VERIFY_NO_METHOD	Service not provided
CSSM_CSP_VERIFY_FAILED	Unable to perform verification on data
CSSM_CSP_VERIFY_INIT_FAILED	Staged verify initialize function failed
CSSM_CSP_VERIFY_UPDATE_FAILED	Staged verify update function failed
CSSM_CSP_VERIFY_FINAL_FAILED	Staged verify final function failed
CSSM_CSP_DIGEST_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_DIGEST_NO_METHOD	Service not provided
CSSM_CSP_DIGEST_FAILED	Unable to perform digest on data
CSSM_CSP_DIGEST_INIT_FAILED	Unable to perform digest initialization
CSSM_CSP_DIGEST_UPDATE_FAILED	Unable to perform digest on data
CSSM_CSP_DIGEST_CLONE_FAILED	Unable to clone the digest context
CSSM_CSP_DIGEST_FINAL_FAILED	Staged digest final failed
CSSM_CSP_MAC_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_MAC_NO_METHOD	Service not provided
CSSM_CSP_MAC_FAILED	Unable to perform mac on data
CSSM_CSP_MAC_INIT_FAILED	Unable to perform staged mac init
CSSM_CSP_MAC_UPDATE_FAILED	Unable to perform staged mac update
CSSM_CSP_MAC_FINAL_FAILED	Unable to perform staged mac final
CSSM_CSP_ENC_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_ENC_NO_METHOD	Service not provided
CSSM_CSP_ENC _FAILED	Unable to encrypt data
CSSM_CSP_ENC_BAD_IV_LENGTH	Length of IV unsupported
CSSM_CSP_ENC_BAD_KEY_LENGTH	Length of key unsupported
CSSM_CSP_ENC_INIT_FAILED	Unable to perform encrypt initialization
CSSM_CSP_ENC_FINAL_FAILED	Unable to encrypt data
CSSM_CSP_DEC_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_DEC_NO_METHOD	Service not provided
CSSM_CSP_DEC _FAILED	Unable to encrypt data

CSSM_CSP_DEC _BAD_IV_LENGTH	Length of IV unsupported
CSSM_CSP_DEC _BAD_KEY_LENGTH	Length of key unsupported
CSSM_CSP_DEC_INIT_FAILED	Unable to perform decrypt initialization
CSSM_CSP_DEC _UPDATE_FAILED	Staged encryption update failed
CSSM_CSP_DEC_FINAL_FAILED	Stages encrypt final failed
CSSM_CSP_KEYGEN_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_KEYGEN_NO_METHOD	Service not provided
CSSM_CSP_KEYGEN_FAILED	Unable to generate key pair
CSSM_CSP_RNG_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_RNG_NO_METHOD	Service not provided
CSSM_CSP_RNG_FAILED	Unable to generate keys
CSSM_CSP_UIDG_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_UIDG_NO_METHOD	Service not provided
CSSM_CSP_UIDG_FAILED	Unable to generate unique ID
CSSM_INVALID_KEY	Invalid wrapping key
CSSM_CSP_PRIKEY_NOT_FOUND	Cannot find the corresponding private key
CSSM_CSP_PASSWORD_INCORRECT	Password incorrect
CSSM_INVALID_SUBJECT_KEY	Invalid key to be wrapped
CSSM_INVALID_PASSPHRASE	Invalid passphrase for the unwrapping key or invalid passphrase for securing the unwrapped key in persistent storage
CSSM_INVALID_WRAPPED_KEY	Invalid wrapped key
CSSM_CSP_DERIVE _FAILED	Unable to derive key
CSSM_CSP_KEYEXCH_GENPARAM_FAILED	Unable to generate exchange param data
CSSM_CSP_KEYEXCH_PHASE1_FAILED	Unable to generate to stage key exchange
CSSM_CSP_KEYEXCH_PHASE2_FAILED	Unable to stage key exchange

7.3.2.4 Cryptographic Module Management Function Errors

CSSM_INVALID_POINTER	Invalid pointer
CSSM_REGISTRY_ERROR	Error in writing registry
CSSM_NO_ADDIN	No add-ins found
CSSM_INVALID_ GUID	CSP module was not installed
CSSM_MEMORY_ERROR	Error in memory allocation
CSSM_INCOMPATIBLE_VERSION	Incompatible version
CSSM_EXPIRE	Add-in has expired
CSSM_INVALID_ARGS	Invalid argument pointer
CSSM_ATTACH_FAIL	Unable to load CSP module
CSSM_INVALID_ADDIN_HANDLE	Invalid CSP handle
CSSM_INVALID_CSPINFO_POINTER	Invalid pointer

7.3.2.5 Cryptographic Extensibility Function Errors

CSSM_CSP_INVALID_CSP_HANDLE	Invalid CSP handle
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_CONTEXT_POINTER	Invalid context pointer
CSSM_CSP_INVALID_DATA_POINTER	Invalid pointer for input data
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_UNSUPPORTED_OPERATION	Add-in does not support this function
CSSM_CSP_PASS_THROUGH_FAIL	Unable to perform custom function

7.3.3 TP Error Codes

7.3.3.1 Trust Policy Operation Errors

CSSM_TP_INVALID_TP_HANDLE	Invalid handle
CSSM_TP_INVALID_CL_HANDLE	Invalid handle
CSSM_TP_INVALID_DL_HANDLE	Invalid handle
CSSM_TP_INVALID_DB_HANDLE	Invalid handle
CSSM_TP_INVALID_CC_HANDLE	Invalid handle
CSSM_TP_INVALID_CERTIFICATE	Invalid certificate
CSSM_TP_NOT_SIGNER	Signer certificate is not signer of subject
CSSM_TP_NOT_TRUSTED	Signature can't be trusted
CSSM_TP_CERT_VERIFY_FAIL	Unable to verify certificate
CSSM_FUNCTION_NOT_IMPLEMENTED	Function not implemented
CSSM_TP_CERTIFICATE_CANT_OPERATE	Signer certificate can't sign subject
CSSM_TP_MEMORY_ERROR	Error in allocating memory
CSSM_TP_CERT_SIGN_FAIL	Unable to sign certificate
CSSM_TP_INVALID_CRL	Invalid CRL
CSSM_TP_CERT_REVOKE_FAIL	Unable to revoke certificate
CSSM_TP_CRL_VERIFY_FAIL	Unable to verify certificate
CSSM_TP_CRL_SIGN_FAIL	Unable to sign certificate revocation list
CSSM_TP_APPLY_CRL_TO_DB_FAIL	Unable to apply certificate revocation list on database

7.3.3.2 Trust Policy Extensibility Function Errors

CSSM_TP_INVALID_TP_HANDLE	Invalid handle
CSSM_TP_INVALID_CL_HANDLE	Invalid handle
CSSM_TP_INVALID_DL_HANDLE	Invalid handle
CSSM_TP_INVALID_DB_HANDLE	Invalid handle
CSSM_TP_INVALID_CC_HANDLE	Invalid handle
CSSM_TP_INVALID_CERTIFICATE	Invalid certificate
CSSM_TP_INVALID_ACTION	Invalid action
CSSM_TP_NOT_TRUSTED	Certificate not trusted for action
CSSM_TP_VERIFY_ACTION_FAIL	Unable to determine trust for action
CSSM_FUNCTION_NOT_IMPLEMENTED	Function not implemented
CSSM_TP_INVALID_DATA_POINTER	Invalid pointer for input data
CSSM_TP_INVALID_ID	Invalid pass-through ID
CSSM_TP_MEMORY_ERROR	Error in allocating memory
CSSM_TP_PASS_THROUGH_FAIL	Unable to perform pass through

7.3.3.3 Trust Policy Module Management Function Errors

CSSM_INCOMPATIBLE_VERSION	Version is not compatible
CSSM_TP_INVALID_POINTER	Invalid pointer
CSSM_TP_REGISTRY_ERROR	Error in writing registry
CSSM_INVALID_POINTER	Invalid pointer
CSSM_REGISTRY_ERROR	Error in writing registry
CSSM_NO_ADDIN	No add-ins found
CSSM_MEMORY_ERROR	Error in memory allocation

CSSM_EXPIRE	Add-in has expired
CSSM_ATTACH_FAIL	Unable to load TP module
CSSM_INVALID_ADDIN_HANDLE	Invalid TP handle
CSSM_INVALID_GUID	Unknown GUID
CSSM_INVALID_TPINFO_POINTER	Invalid pointer

7.3.4 CL Error Codes

7.3.4.1 Certificate Operation Errors

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_UNSUPPORTED_OPERATION	Add-in does not support this function
CSSM_CL_CERT_SIGN_FAIL	Unable to sign certificate
CSSM_CL_CERT_UNSIGN_FAIL	Unable to unsign certificate
CSSM_CL_CERT_VERIFY_FAIL	Unable to verify certificate
CSSM_CL_INVALID_FIELD_POINTER	Invalid pointer input
CSSM_CL_INVALID_TEMPLATE	Invalid template for this certificate type
CSSM_CL_CERT_CREATE_FAIL	Unable to create certificate
CSSM_CL_INVALID_FIELD_POINTER	Invalid pointer input
CSSM_CL_CERT_VIEW_FAIL	Unable to view certificate
CSSM_CL_UNKNOWN_TAG	Unknown field tag in OID
CSSM_CL_CERT_GET_FIELD_VALUE_FAIL	Unable to get field value
CSSM_CL_INVALID_RESULTS_HANDLE	Invalid Results Handle
CSSM_CL_NO_FIELD_VALUES	No more field values for the input handle
CSSM_CL_CERT_ABORT_QUERY_FAIL	Unable to abort the certificate query
CSSM_CL_CERT_GET_KEY_INFO_FAIL	Unable to get key information
CSSM_CL_CERT_GET_FIELD_VALUE_FAIL	Unable to return the list of fields
CSSM_CL_CERT_IMPORT_FAIL	Unable to import certificate
CSSM_CL_CERT_EXPORT_FAIL	Unable to export certificate
CSSM_CL_CERT_DESCRIBE_FORMAT_FAIL	Unable to return the list of fields

7.3.4.2 Certificate Group Operation Errors

CSSM_ INVALID_CL_ HANDLE	Invalid certificate library handle
CSSM_CL_INVALID_CERT_GROUP	Invalid certificate group
CSSM_INVALID_DB_HANDLE	Bad database handle
CSSM_ MEMORY_ERROR	Not enough memory to allocate
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_UNSUPPORTED_OPERATION	Add-in does not support this function
CSSM_CL_CERT_VERIFY_FAIL	Unable to verify certificate

7.3.4.3 Certificate Revocation List Operation Errors

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
CSSM_CL_INVALID_CC_HANDLE	Invalid Context Handle
CSSM_CL_INVALID_CERTIFICATE_PTR	Invalid Certificate
CSSM_CL_INVALID_CRL	Invalid CRL
CSSM_CL_CRL_ADD_CERT_FAIL	Unable to add certificate to CRL
CSSM_CL_CERT_NOT_FOUND_IN_CRL	Certificate not referenced by the CRL
CSSM_CL_CRL_REMOVE_CERT_FAIL	Unable to remove certificate from CRL
CSSM_CL_INVALID_CRL_PTR	Invalid CRL pointer
CSSM_CL_INVALID_SCOPE	Signing scope is invalid
CSSM_CL_CRL_SIGN_FAIL	Unable to sign CRL
CSSM_CL_INVALID_SCOPE	Verify scope is invalid
CSSM_CL_CRL_VERIFY_FAIL	Unable to verify CRL
CSSM_CL_UNKNOWN_TAG	Unrecognized field tag in OID
CSSM_CL_NO_FIELD_VALUES	No fields match the specified OID
CSSM_CL_CRL_GET_FIELD_VALUE_FAIL	Unable to get first field value
CSSM_CL_NO_FIELD_VALUES	No more matches in the CRL
CSSM_CL_INVALID_RESULTS_HANDLE	Invalid query handle
CSSM_CL_CRL_ABORT_QUERY_FAIL	Unable to get next item
CSSM_CL_CRL_DESCRIBE_FORMAT_FAIL	Unable to return the list of fields

7.3.4.4 Certificate Library Module Management Function Errors

CSSM_INVALID_POINTER	Invalid pointer
CSSM_REGISTRY_ERROR	Error in writing registry
CSSM_MEMORY_ERROR	Error in memory allocation
CSSM_INCOMPATIBLE_VERSION	Incompatible version
CSSM_ATTACH_FAIL	Unable to attach to CL module
CSSM_INVALID_ADDIN_HANDLE	Invalid CL handle
CSSM_INVALID_GUID	No known CL module with specified GUID

7.3.4.5 Certificate Library Extensibility Function Errors

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

7.3.5 DL Error Codes

7.3.5.1 Data Source Operation Errors

CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle

CSSM_DL_DATASTORE_NOT_EXISTS	The data store with the logical name does not exist
CSSM_DL_DB_OPEN_FAIL	Open caused an exception
CSSM_DL_MEMORY_ERROR	Error in allocating memory
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_DB_CLOSE_FAIL	Close caused an exception
CSSM_DL_INVALID_CL_HANDLE	Invalid CL handle
CSSM_DL_INVALID_PTR	Invalid pointer to the data store name
CSSM_DL_DB_CREATE_FAIL	Create caused an exception
CSSM_DL_DB_DELETE_FAIL	Delete caused an exception
CSSM_DL_DB_IMPORT_FAIL	DB exception doing import function
CSSM_DL_DB_EXPORT_FAIL	DB exception doing export function

7.3.5.2 Generic Security Object Storage Operation Errors

CSSM_DL_INVALID_DL_HANDLE	Invalid Data Storage Library handle
CSSM_DL_INVALID_DB_HANDLE	Invalid Data Store handle
CSSM_DL_INVALID_MODULE_HANDLE	Invalid Module handle for accessing the data record
CSSM_DL_INVALID_DATA_PTR	Invalid data pointer
CSSM_DL_DATA_INSERT_FAIL	Add caused an exception
CSSM_DL_DATA_NOT_IN_DB	Data record not in Data Store
CSSM_DL_DATA_DELETE_FAIL	Delete caused an exception
CSSM_DL_INVALID_SELECTION_PTR	Invalid selection predicate pointer
CSSM_DL_NO_DATA_FOUND	No data records match the selection predicate
CSSM_DL _DATA_GETFIRST_FAIL	Opening the records caused an exception
CSSM_DL_MEMORY_ERROR	Error in allocating memory
CSSM_DL_INVALID_RESULTS_HANDLE	Invalid query handle
CSSM_DL_NO_MORE_RECORDS	No more records for this selection handle
CSSM_DL_DATA_GETNEXT_FAIL	Opening the records caused an exception
CSSM_DL_DATA_ABORT_QUERY_FAIL	Unable to abort query

7.3.5.3 Certificate Storage Operation Errors

CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_CERTIFICATE_PTR	Invalid certificate pointer
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_CERT_INSERT_FAIL	Add caused an exception
CSSM_DL_CERTIFICATE_NOT_IN_DB	Certificate not in DB
CSSM_DL_CERT_DELETE_FAIL	Delete caused an exception
CSSM_DL_CERT_REVOKE_FAIL	Update caused an exception
CSSM_DL_INVALID_SELECTION_PTR	Invalid selection predicate pointer
CSSM_DL_NO_CERTIFICATE_FOUND	No certificates that match the selection predicate
CSSM_DL_CERT_GETFIRST_FAIL	Opening the records caused an exception
CSSM_DL_MEMORY_ERROR	Error in allocating memory
CSSM_DL_INVALID_RESULTS_HANDLE	Invalid query handle

CSSM_DL_NO_MORE_CERTS	No more certificates for that selection handle
CSSM_DL_CERT_GETNEXT_FAIL	Opening the records caused an exception
CSSM_DL_CERT_ABORT_QUERY_FAIL	Unable to abort query
CSSM_DL_INVALID_CRL_PTR	Invalid CRL pointer
CSSM_DL_CRL_INSERT_FAIL	Add caused an exception
CSSM_DL_CRL_NOT_IN_DB	CRL not in DB
CSSM_DL_CRL_DELETE_FAIL	Delete caused an exception
CSSM_DL_INVALID_SELECTION_PTR	Invalid selection predicate pointer
CSSM_DL_NO_CRL_FOUND	No CRLs that match the selection predicate
CSSM_DL_CRL_GET_FIRST_FAIL	Get first caused an exception
CSSM_DL_NO_MORE_CRLS	No more CRLs for that selection handle
CSSM_DL_CRL_GET_NEXT_FAIL	Opening the records caused an exception
CSSM_DL_CRL_ABORT_QUERY_FAIL	Unable to abort query

7.3.5.4 Data Storage Library Module Management Function Errors

CSSM_INVALID_POINTER	Invalid pointer
CSSM_REGISTRY_ERROR	Error in writing registry
CSSM_MEMORY_ERROR	Error in memory allocation
CSSM_INCOMPATIBLE_VERSION	Incompatible version
CSSM_ATTACH_FAIL	Unable to attach to DL module
CSSM_INVALID_ADDIN_HANDLE	Invalid DL handle
CSSM_INVALID_GUID	No known module with specified GUID
CSSM_INVALID_DLINFO_ POINTER	Invalid CSSM_DL_INFO pointer
CSSM_DL_NO_DATA_SOURCES	No known data store names
CSSM_DL_GET_DB_NAMES_FAIL	Get DB Names failed
CSSM_DL_INVALID_DL_HANDLE	Invalid DL Handle
CSSM_DL_INVALID_PTR	Invalid pointer to the name list

7.3.5.5 Data Storage Library Extensibility Function Errors

CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_INVALID_PASSTHROUGH_ID	Invalid passthrough ID
CSSM_DL_INVALID_ PTR	Invalid pointer
CSSM_DL_ PASS_THROUGH_FAIL	DB exception doing passthrough function
CSSM_DL_MEMORY_ERROR	Error in allocating memory

7.4 Error Handling Functions

7.4.1 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 no valid error, the error number will be CSSM_OK. A NULL pointer indicates that the CSSM_InitError was not called or that a call to CSSM_DestroyError has been made. No error information is available.

See Also

CSSM_InitError, CSSM_DestroyError, CSSM_ClearError, CSSM_SetError, CSSM_IsCSSMError, CSSM_IsCLError, CSSM_IsTPError, CSSM_IsDLError, CSSM_IsCSPError

7.4.2 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, or CSSM_DestroyError has been called. No error information is available.

See Also

CSSM_InitError, CSSM_DestroyError, CSSM_ClearError, CSSM_GetError

7.4.3 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

7.4.4 CSSM_InitError

CSSM_RETURN CSSMAPI CSSM_InitError (void)

This function initializes the error information for that thread/process and allocates any necessary memory. Should be called by the thread/process initialization function.

Parameters

None

Return Value

CSSM_OK if the error information was successfully initialized. If CSSM_FAIL is returned, no error information will be available.

Notes

CSSM_InitError does not need to be called if you have loaded the CSSM DLL.

See Also

CSSM_DestroyError

7.4.5 CSSM_DestroyError

CSSM_RETURN CSSMAPI CSSM_DestroyError (void)

This function destroys the error information for a thread/process and frees any necessary memory. It should be called by the function performing clean up before a thread/process exits.

Parameters

None

Return Value

CSSM_OK if the error information was successfully destroyed. If CSSM_FAIL is returned, no error information will be available.

Notes

CSSM_DestroyError does not need to be called if you have loaded the CSSM DLL.

See Also

CSSM_InitError

7.4.6 CSSM_IsCSSMError

CSSM_BOOL CSSMAPI CSSM_IsCSSMError (uint32 error_number)

This function determines if*error_number*is within the CSSM range of errors.

Parameters

error_number (input)
An error number.

Return Value

CSSM_TRUE if the error is a CSSM error; otherwise CSSM_FALSE.

See Also

CSSM_IsCLError, CSSM_IsDLError, CSSM_IsTPError, CSSM_IsCSPError

7.4.7 CSSM_IsCLError

CSSM_BOOL CSSMAPI CSSM_IsCLError (uint32 error_number)

This function determines if*error_number*is within the CL range of errors.

Parameters

error_number (input)
An error number.

Return Value

CSSM_TRUE if the error is a CL error; otherwise CSSM_FALSE.

See Also

CSSM_IsCSSMError, CSSM_IsDLError, CSSM_IsTPError, CSSM_IsCSPError

7.4.8 CSSM_IsDLError

CSSM_BOOL CSSMAPI CSSM_IsDLError (uint32 error_number)

This function determines if*error_number*is within the DL range of errors.

Parameters

error_number (input)
An error number.

Return Value

CSSM_TRUE if the error is a DL error; otherwise CSSM_FALSE.

See Also

CSSM_IsCLError, CSSM_IsCSSMError, CSSM_IsTPError, CSSM_IsCSPError

7.4.9 CSSM_IsTPError

CSSM_BOOL CSSMAPI CSSM_IsTPError (uint32 error_number)

This function determines if*error_number*is within the TP range of errors.

Parameters

error_number (input)
An error number.

Return Value

CSSM_TRUE if the error is a TP error; otherwise CSSM_FALSE.

See Also

CSSM_IsCLError, CSSM_IsDLError, CSSM_IsCSSMError, CSSM_IsCSPError

7.4.10 CSSM_IsCSPError

CSSM_BOOL CSSMAPI CSSM_IsCSPError (uint32 error_number)

This function determines if*error_number*is within the CSP range of errors.

Parameters

error_number (input)
An error number.

Return Value

CSSM_TRUE if the error is a CSP error; otherwise CSSM_FALSE.

See Also

CSSM_IsCLError, CSSM_IsDLError, CSSM_IsTPError, CSSM_IsCSSMError

7.4.11 CSSM_CompareGuids

CSSM_BOOL CSSMAPI CSSM_CompareGuids (CSSM_GUID guid1,
CSSM_GUID guid2)

This function determines if two GUIDs are equal.

Parameters

guid1 (input)
A GUID.

guid1 (input)
A GUID.

Return Value

CSSM_TRUE if the two GUIDs are equal, CSSM_FALSE otherwise.

Notes

GUIDs are returned in the error information of CSSM_GetError. Once you know which type of error is returned (CSP, CL, TP, DL), you can call CSSM_XX_ListModules to get a list of all the modules that are registered and their GUIDs in order to determine which module set the error. This can be useful for debugging purposes if there is more than one type of module for each add-in type installed on the system.

See Also

CSSM_GetError, CSSM_CSP_ListModules, CSSM_CL_ListModules, CSSM_TP_ListModules, CSSM_DL_ListModules.

8. Appendix B. Application Memory Functions

8.1 Introduction

When CSSM or add-in modules return memory structures to applications, that memory is maintained by the application. Instead of using a model where the application passes memory blocks to the add-in modules to work on, the CSSM model requires the application to supply memory functions. This has the advantage for applications not requiring to know the sizes of memory blocks to supply to the CSSM and the add-ins. The memory that the application receives is in its process space, and this prevents the application from walking through the memory of the CSSM or the add-in modules. An application that has access to secure memory could supply functions to the cryptographic service provider for managing that memory. All data returned from the cryptographic service provider will be through that secure memory. When the application no longer requires the memory, it is responsible for freeing it.

Applications will register memory functions with the add-in modules during attach time and with CSSM during initialization. A memory function table will be passed from the application to add-in modules through the CSSM_xxx_Attach functions associated with each add-in. The CSSM_Init function is where the CSSM will receive the application's memory function.

8.1.1 CSSM_API_MEMORY_FUNCS Data Structure

This structure is used by applications to supply memory functions for the CSSM and the add-in modules. The functions are used when memory needs to be allocated by the CSSM or add-ins for returning data structures to the applications.

```
typedef struct cssm_api_memory_funcs {
    void * (*malloc_func) (uint32 size, void *allocRef);
    void (*free_func) (void *memblock, void *allocRef);
    void * (*realloc_func) (void *memblock, uint32 size, void *allocRef);
    void * (*calloc_func) (uint32 num, uint32 size, void *allocRef);
} CSSM_API_MEMORY_FUNCS, *CSSM_API_MEMORY_FUNCS_PTR
```

Definition:

malloc_func - pointer to function that returns a void pointer to the allocated memory block of at least *size* bytes from heap allocRef

free_func - pointer to function that deallocates a previously-allocated memory block (*memblock*) from heap allocRef

realloc_func - pointer to function that returns a void pointer to the reallocated memory block (*memblock*) of at least *size* bytes from heap allocRef

calloc_func - pointer to function that returns a void pointer to an array of *num* elements of length *size* initialized to zero from heap allocRef

8.1.2 Initialization of Memory Structure

The memory structure CSSM_API_MEMORY_FUNCS requires pointers to functions that implement the memory routines. Below is an example of an application supplying the C runtime utilities malloc, realloc and free to the memory structure. The memory structure is then used by the CSSM_Init call.

```
/* Allocating the structure */
MemoryFuncs = (CSSM_API_MEMORY_FUNCS_PTR)malloc (
                          sizeof (CSSM_API_MEMORY_FUNCS));

/* Initialize the memory function structure */
MemoryFuncs->malloc_func = HeapMalloc;
MemoryFuncs->realloc_func = HeapRealloc;
MemoryFuncs->free_func = HeapFree;
MemoryFuncs->calloc_func = HeapCalloc;

/* Initialize the CSSM */
CSSM_Init (CSSM_MAJOR, CSSM_MINOR, MemoryFuncs, NULL);
```

9. Appendix C. Acronyms

For a complete glossary of terms, see the *CDSA Specification*.

API	Application Programming Interface
CBC	Cipher Block Chaining (cryptographic algorithm context)
CDSA	Common Data Security Architecture
CLI	Certificate Library Interface
CSP	Cryptographic Service Provider
CSSM	Common Security Services Manager
DLI	Data Storage Library Interface
DLL	Dynamic Link Library
GUID	Global Unique Identifier
IV	Initialization Vector (cryptographic algorithm context)
SPI	Cryptographic Service Provider Interface
TC	Test and Check used to ensure CSSM self-integrity
TPI	Trust Policy Interface