Common Security Services Manager

Cryptographic Service Provider Interface (SPI) Specification

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, 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 Figures

List of Tables

1. Introduction

1.1 CDSA Overview

The Common Data Security Architecture (CDSA) defines the infrastructure for a comprehensive 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.

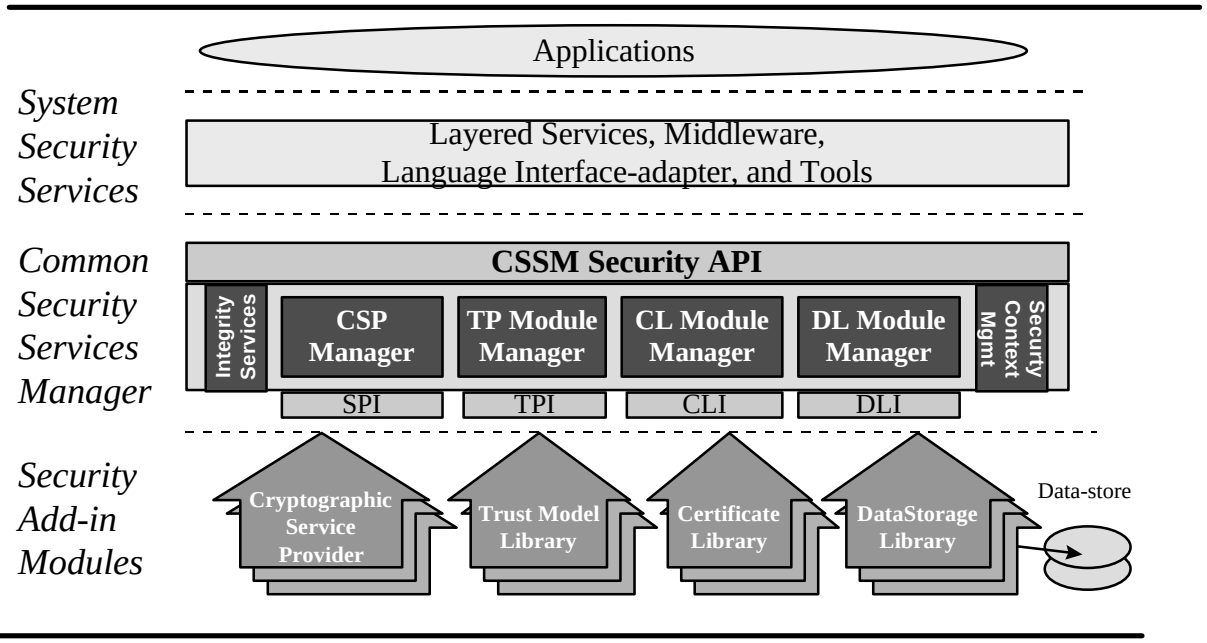

Figure 1. **The Common Data Security Architecture for all platforms.**

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 Cryptographic Service Provider Overview

The CSSM infrastructure doesn't implement any cryptography. It has been termed "crypto with a hole." The Cryptographic Services Manager provides applications with access to cryptographic functions that are implemented by Cryptographic Service Provider (CSP) modules. This achieves the objective of centralizing all the cryptography into exchangeable modules.

The Cryptographic Services Manager defines two categories of services:

- Module management - installation, feature registration, and query of CSP features
- Selection, initialization, and use of cryptographic operations, which are implemented by a CSP

The nature of the cryptographic functions contained in any particular CSP depends on what task the CSP was designed to perform. For example, a VISA* smart card* would be able to digitally sign credit card transactions on behalf of the card's owner, whereas a digital employee badge would be able to authenticate a user for physical or electronic access.

A CSP can perform one or more of these cryptographic functions:

- Bulk encryption
- Digital signature
- Cryptographic hash
- Unique identification number
- Random number generator

The Cryptographic Services Manager doesn't assume any particular form factor for a CSP. Indeed, CSPs can be instantiated in hardware, software or both. Operationally, the distinction must be transparent. The two visible distinctions between hardware and software implementations are the degree of trust the application receives by using a given CSP, and the cost of developing that CSP. A hardware implementation should be more tamper-resistant than a software implementation. Hence a higher level of trust is achieved by the application.

Software CSPs are the default and are portable in that they can be carried as an executable file. Additionally, the modules that implement a CSP must be digitally signed (to authenticate their origin and integrity), and they should be made as tamper-resistant as possible. This requirement extends to software

implementations and hardware. Multiple CSPs may be loaded and active within the CSSM at any time. A single application may use multiple CSPs concurrently. Interpreting the resulting level of trust and security is the responsibility of the application or the trust-policy module used by the application.

A small (yet significant) number of CSPs existed prior to the definition of CSSM Cryptographic API. These legacy CSPs have defined their own API for cryptographic services. These interfaces are CSP-specific, non-standard, and in general low-level, key-based interfaces. Low-level, key-based interfaces present a considerable development effort to the application developer attempting to secure an application by using those services.

The Cryptographic Services Manager defines a high-level, certificate-based API for cryptographic services to better support application development. In consideration of legacy and divergent CSPs, the Cryptographic Services Manager defines a lower-level Service Provider Interface (SPI) that more closely resembles typical CSP APIs, and provides CSP developers with a single interface to support. A CSP may or may not support multithreaded applications.

Acknowledging legacy CSPs, the CSSM architecture defines an optional adaptation layer between the Cryptographic Services Manager and a CSP. The adaptation layer allows the CSP vendor to implement a shim to map the CSSM SPI to the CSP's existing API, and to implement any additional management functions that are required for the CSP to function as an add-in module in the extensible CSSM architecture. New CSPs may support the CSSM SPI directly (without the aid of an adaptation layer).

1.3 CSSM Service Provider Interface Specification

1.3.1 Intended Audience

This document is intended for use by Independent Software Vendors (ISVs) who will develop their own CSPs to provide cryptographic 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.

1.3.2 Document Organization

This document is divided into the following sections:
Section 2, Service Provider Interface, describes the functions which a CSP makes available to applications via the CSSM.

Section 3, CSP Structure and Management, describes important considerations in developing a CSP. It also gives examples of how CSP functions might be implemented.

1.4 References

BSAFE*	*BSAFE Cryptographic Toolkit*, RSA Data Security, Inc., Redwood City, CA
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)
Cryptography	*Applied Cryptography, Second Edition Protocols, Algorithms, and Source Code in C*, Bruce Schneier: John Wiley & Sons, Inc., 1996
CDSA Spec	*Common Data Security Architecture Specification*, Intel Architecture Labs, 1997

CSSM API *CSSM Application Programming Interface* Intel Architecture Labs, 1997

2. Service Provider Interface

2.1 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.

2.1.1 Cryptographic Operations

CSSM_RETURN CSP_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 input size (encryption and decryption only) and output size for the specified algorithm.

CSSM_RETURN CSP_SignData
CSSM_RETURN CSP_SignDataInit
CSSM_RETURN CSP_SignDataUpdate
CSSM_RETURN CSP_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 CSP_VerifyData
CSSM_RETURN CSP_VerifyDataInit
CSSM_RETURN CSP_VerifyDataUpdate
CSSM_BOOL CSP_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 CSP_DigestData
CSSM_RETURN CSP_DigestDataInit
CSSM_RETURN CSP_DigestDataUpdate
CSSM_RETURN CSP_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 CSP_DigestDataClone accepts as input a handle to a cryptographic context describing the digest operation. A handle to another cryptographic context is created with similar information and intermediate result as described by the first context.

CSSM_RETURN CSP_GenerateMac
CSSM_RETURN CSP_GenerateMacInit
CSSM_RETURN CSP_GenerateMacUpdate
CSSM_RETURN CSP_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 CSP_EncryptData
CSSM_RETURN CSP_EncryptDataInit
CSSM_RETURN CSP_EncryptDataUpdate
CSSM_RETURN CSP_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 CSP_DecryptData
CSSM_RETURN CSP_DecryptDataInit
CSSM_RETURN CSP_DecryptDataUpdate
CSSM_RETURN CSP_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 CSP_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 CSP_GenerateKeyPair - accepts as input a handle to a cryptographic context describing the generate key operation. The keys are returned in a CSSM_KEY structures.

CSSM_RETURN CSP_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 CSP_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 CSP_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 CSP_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 CSP_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 CSP_KeyExchGenParam
CSSM_RETURN CSP_KeyExchPhase1
CSSM_RETURN CSP_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.

2.1.2 Cryptographic Sessions and Logon

CSSM_RETURN 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 CSP_Logout - the caller is logged out of the current login session with the designated CSP.

CSSM_RETURN 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.

2.1.3 Extensibility Functions

CSSM_RETURN CSP_PassThrough - This performs the CSP module-specific function indicated by the operation ID. The operation ID specifies an operation which the CSP has exported for use by an application or module. Such operations should be specific to the key format of the private keys stored in the CSP module.

2.1.4 Module Management Functions

CSSM_RETURN CSP_Initialize - Performs internal CSP initialization functions and version checking.

CSSM_RETURN CSP_Uninitialize - Performs any internal clean-up required by the CSP.

CSSM_CSPINFO_PTR CSP_GetCapabilities - Used by CSPs with dynamic capabilities to return a set of CSSM_CSPINFO structures describing itself.

CSSM_RETURN CSP_EventNotify - Called by the CSSM to notify the CSP that an important event has taken place.

2.2 Data Structures

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

2.2.1 CSSM_CSP_HANDLE

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

```
typedef uint32 CSSM_CSP_HANDLE  /* Cryptographic Service Provider Handle */
```

2.2.2 CSSM_DATA

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

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

Definition:

Length - length of the data buffer in bytes.

Data - pointer to a data buffer.

2.2.3 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 0x01.

```
#define CSSM_KEYHEADER_VERSION            (1)
```

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 vs permanent KeyData */
#define CSSM_BLOBDESC_TRANS_MASK         (0x40000000) /* Mask transient bit */
#define CSSM_BLOBDESC_PERMANENT          (0x00000000) /* KeyData constant across
                                                      attaches */
#define CSSM_BLOBDESC_TRANSIENT          (0x40000000) /* KeyData not constant
                                                      across attaches */

/* Data Type */
#define CSSM_BLOBDESC_TYPE_MASK          (0x30000000) /* Mask type value */
#define CSSM_BLOBDESC_DATA               (0x00000000) /* Actual key data */
#define CSSM_BLOBDESC_HANDLE             (0x10000000) /* Handle ref to key */
#define CSSM_BLOBDESC_LABEL                    (0x20000000) /* Label ref to ke   y
*/

/* 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              (0x00FFFF00) /* 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 0x01.

```
#define CSSM_DATAFORMAT_VERSION          (1)
```

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.

2.2.4 CSSM_KEYBLOB

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

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

Definition:

KeyHeader - Key header for the key.

KeyData - Data representation of the key.

2.2.5 CSSM_KEY

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

2.2.6 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.

2.2.7 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.

2.2.8 CSSM_CSP_TYPE

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

2.2.9 CSSM_CSP_SESSION_TYPE

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

2.2.10 CSSM_NOTIFY_CALLBACK

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

2.2.11 CSSM_HANDLEINFO

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

2.2.12 CSSM_CSPPININFO

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

2.2.13 CSSM_CSPMEMINFO

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

2.2.14 CSSM_CSPSESSIONINFO

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

2.2.15 CSSM_CSPINFO

```
typedef struct cssm_cspinfo {
       uint32 VerMajor;
       uint32 VerMinor;
       CSSM_BOOL ExportFlag;
       CSSM_BOOL MultiTasking;
       CSSM_BOOL SerialRequired;
       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 Ca pabilitiesInitialized;
       uint32 NumberOfCapabilities;
       CSSM_CONTEXT_PTR Capabilities;
       CSSM_CSPPININFO PinInfo;           /* CSP Pin information */
       CSSM_CSPMEMINFO MemInfo;           /* CSP memory information */
       CSSM_CSPSESSIONINFO SessionInfo;   /* 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 - True or false, if true CSP is not capable of executing parallel ops.

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 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.

2.2.16 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.

2.2.17 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.

2.3 Cryptographic Operations

2.3.1 CSP_QuerySize

CSSM_RETURN CSSMSPI CSP_QuerySize (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
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, CSP_EncryptDataUpdate and CSP_DecryptDataUpdate). There may be algorithm-specific and token-specific rules restricting the lengths of data following data update calls

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes associated with this context.

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_INVALID_CONTEXT_POINTER	Invalid context pointer
CSSM_CSP_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_NO_METHOD	Service not provided
CSSM_CSP_QUERY_SIZE_FAILED	Unable to query size

See Also

CSP_EncryptData, CSP_EncryptDataUpdate, CSP_DecryptData, CSP_DecryptDataUpdate, CSP_SignData, CSP_VerifyData, CSP_DigestData, CSP_GenerateMac

2.3.2 CSP_SignData

CSSM_RETURN CSSMSPI CSP_SignData (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
const CSSM_DATA_PTR DataBufs,
uint32 DataBufCount,
CSSM_DATA_PTR Signature)

This function signs data using the private key.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

DataBufs (input)
A pointer to one or more CSSM_DATA structures containing the data to be signed.

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_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
CSSM_CSP_INVALID_DATA_COUNT	Invalid data count
CSSM_CSP_INVALID_CALLBACK	Invalid call back function
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_PASSWORD_NO_PARAM	No password or callback function provided
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

CSP_VerifyData, CSP_SignDataInit, CSP_SignDataUpdate, CSP_SignDataFinal

2.3.3 CSP_SignDataInit

CSSM_RETURN CSSMSPI CSP_SignDataInit (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context)

This function initializes the staged sign data function.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes 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_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_CONTEXT_POINTER	Invalid context pointer
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

CSP_SignData, CSP_SignDataUpdate, CSP_SignDataFinal

2.3.4 CSP_SignDataUpdate

CSSM_RETURN CSSMSPI CSP_SignDataUpdate (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR DataBufs,
uint32 DataBufCount)

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

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

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 one or more CSSM_DATA structures containing the data be signed.

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_CSP_HANDLE	Invalid CSP handle
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

CSP_SignData, CSP_SignDataInit, CSP_SignDataFinal

2.3.5 CSP_SignDataFinal

CSSM_RETURN CSSMSPI CSP_SignDataFinal (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR Signature)

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

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

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_CSP_HANDLE	Invalid CSP handle
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

CSP_SignData, CSP_SignDataInit, CSP_SignDataUpdate

2.3.6 CSP_VerifyData

CSSM_BOOL CSSMSPI CSP_VerifyData (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
const CSSM_DATA_PTR DataBufs,
uint32 DataBufCount,
const CSSM_DATA_PTR Signature)

This function verifies the input data against the provided signature.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

DataBufs (input)
A pointer to one or more CSSM_DATA structures containing the data be verified.

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_CONTEXT_POINTER	Invalid context pointer
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_SIGNATURE_BAD	Signature is bad
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

CSP_SignData, CSP_VerifyDataInit, CSP_VerifyDataUpdate, CSP_VerifyDataFinal

2.3.7 CSP_VerifyDataInit

CSSM_RETURN CSSMSPI CSP_VerifyDataInit (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
const CSSM_DATA_PTR Signature)

This function initializes the staged verify data function.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

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_CONTEXT_POINTER	Invalid context pointer
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

CSP_VerifyDataUpdate, CSP_VerifyDataFinal, CSP_VerifyData

2.3.8 CSP_VerifyDataUpdate

CSSM_RETURN CSSMSPI CSP_VerifyDataUpdate (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR DataBufs,
uint32 DataBufCount)

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

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

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 one or more CSSM_DATA structures containing the data be verified.

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_CONTEXT_POINTER	Invalid context pointer
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

CSP_VerifyData, CSP_VerifyDataInit, CSP_VerifyDataFinal

2.3.9 CSP_VerifyDataFinal

CSSM_BOOL CSSMSPI CSP_VerifyDataFinal (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle)

This function finalizes the staged verify data function.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

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

CSP_VerifyData, CSP_VerifyDataInit, CSP_VerifyDataUpdate

2.3.10 CSP_DigestData

CSSM_RETURN CSSMSPI CSP_DigestData (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
const CSSM_DATA_PTR DataBufs,
uint32 DataBufCount,
CSSM_DATA_PTR Digest)

This function computes a message digest for the supplied data.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

DataBufs (input)
A pointer to one or more CSSM_DATA structures containing the supplied data.

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_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
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

CSP_DigestDataInit, CSP_DigestDataUpdate, CSP_DigestDataFinal, CSP_DigestDataClone

2.3.11 CSP_DigestDataInit

CSSM_RETURN CSSMSPI CSP_DigestDataInit (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context)

This function initializes the staged message digest function.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes 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_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_CONTEXT_POINTER	Invalid context pointer
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

CSP_DigestData, CSP_DigestDataUpdate, CSP_DigestDataClone, CSP_DigestDataFinal

2.3.12 CSP_DigestDataUpdate

CSSM_RETURN CSSMSPI CSP_DigestDataUpdate (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR DataBufs,
uint32 DataBufCount)

This function updates the staged message digest function.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

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 one or more CSSM_DATA structures containing the supplied data.

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

CSP_DigestData, CSP_DigestDataInit, CSP_DigestDataClone, CSP_DigestDataFinal

2.3.13 CSP_DigestDataClone

CSSM_CC_HANDLE CSSMSPI CSP_DigestDataClone (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE oldCCHandle,
CSSM_CC_HANDLE newCCHandle)

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

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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

newCCHandle (output)
The new handle that describes the cloned 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 CSP_DigestDataUpdate and CSP_DigestDataFinal functions.

See Also

CSP_DigestData, CSP_DigestDataInit, CSP_DigestDataUpdate, CSP_DigestDataFinal

2.3.14 CSP_DigestDataFinal

CSSM_RETURN CSSMSPI CSP_DigestDataFinal (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR Digest)

This function finalizes the staged message digest function.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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_CSP_HANDLE	Invalid CSP handle
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

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

CSP_DigestData, CSP_DigestDataInit, CSP_DigestDataUpdate, CSP_DigestDataClone

2.3.15 CSP_GenerateMac

CSSM_RETURN CSSMSPI CSP_GenerateMac (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
const CSSM_DATA_PTR DataBufs,
uint32 DataBufCount,
CSSM_DATA_PTR Mac)

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

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

DataBufs (input)
A pointer to one or more CSSM_DATA structures containing the supplied data.

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_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
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

CSP_GenerateMacInit, CSP_GenerateMacUpdate, CSP_GenerateMacFinal

2.3.16 CSP_GenerateMacInit

CSSM_RETURN CSSMSPI CSP_GenerateMacInit (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context)

This function initializes the staged message authentication code function.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes 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_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_CONTEXT_POINTER	Invalid context pointer
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

CSP_GenerateMac, CSP_GenerateMacUpdate, CSP_GenerateMacFinal

2.3.17 CSP_GenerateMacUpdate

CSSM_RETURN CSSMSPI CSP_GenerateMacUpdate (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR DataBufs,
uint32 DataBufCount)

This function updates the staged message authentication code function.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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 one or more CSSM_DATA structures containing the supplied data.

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

CSP_GenerateMac, CSP_GenerateMacInit, CSP_GenerateMacFinal

2.3.18 CSP_GenerateMacFinal

CSSM_RETURN CSSMSPI CSP_GenerateMacFinal (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR Mac)

This function finalizes the staged message authentication code function.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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

CSP_GenerateMac, CSP_GenerateMacInit, CSP_GenerateMacUpdate

2.3.19 CSP_EncryptData

CSSM_RETURN CSSM_SPI CSP_EncryptData (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
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. Th**€SP_QuerySize** function can be used to estimate the output buffer size required.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

ClearBufs (input)
A pointer to one or more CSSM_DATA structures containing the clear data.

ClearBufCount (input)
The number of *ClearBufs*.

CipherBufs (output)
A pointer to one or more CSSM_DATA structures for the encrypted 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_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
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	
CSSM_CSP_ENC_BAD_KEY_LENGTH	

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

CSP_QuerySize, CSP_DecryptData, CSP_EncryptDataInit, CSP_EncryptDataUpdate, CSP_EncryptDataFinal

2.3.20 CSP_EncryptDataInit

CSSM_RETURN CSSMSPI CSP_EncryptDataInit (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context)

This function initializes the staged encrypt function.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes 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_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_CONTEXT_POINTER	Invalid context pointer
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_ENC_UNKNOWN_ALGORITHM	Unknown algorithm
CSSM_CSP_ENC_NO_METHOD	
CSSM_CSP_ENC_INIT_FAILED	Unable to perform encrypt initialization
CSSM_CSP_ENC_BAD_IV_LENGTH	
CSSM_CSP_ENC_BAD_KEY_LENGTH	

See Also

CSP_EncryptData, CSP_EncryptDataUpdate, CSP_EncryptDataFinal

2.3.21 CSP_EncryptDataUpdate

CSSM_RETURN CSSMSPI CSP_EncryptDataUpdate (CSSM_CSP_HANDLE CSPHandle,
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 **CSP_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 **CSP_EncryptUpdate** calls.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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 one or more CSSM_DATA structures containing the clear data.

ClearBufCount (input)
The number of *ClearBufs*.

CipherBufs (output)
A pointer to one or more CSSM_DATA structures for the encrypted data.

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_CSP_HANDLE	Invalid CSP handle
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_ENC_UPDATE_FAILED	Unable to encrypt data
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. In-place encryption can be done by supplying the same input and output buffer.

See Also

CSP_QuerySize, CSP_EncryptData, CSP_EncryptDataInit, CSP_EncryptDataFinal

2.3.22 CSP_EncryptDataFinal

CSSM_RETURN CSSMSPI CSP_EncryptDataFinal (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR RemData)

This function finalizes the staged encrypt function.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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_CSP_HANDLE	Invalid CSP handle
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

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

CSP_EncryptData, CSP_EncryptDataInit, CSP_EncryptDataUpdate

2.3.23 CSP_DecryptData

CSSM_RETURN CSSMSPI CSP_DecryptData (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
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 **CSP_QuerySize** function can be used to estimate the output buffer size required.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

CipherBufs (input)
A pointer to one or more CSSM_DATA structures containing the encrypted data.

CipherBufCount (input)
The number of *CipherBufs*.

ClearBufs (output)
A pointer to one or more CSSM_DATA structures for the decrypted data.

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_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
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	
CSSM_CSP_DEC_BAD_KEY_LENGTH	

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

CSP_QuerySize, CSP_EncryptData, CSP_DecryptDataInit, CSP_DecryptDataUpdate, CSP_DecryptDataFinal

2.3.24 CSP_DecryptDataInit

CSSM_RETURN CSSMSPI CSSM_CSP_DecryptDataInit (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context)

This function initializes the staged decrypt function.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes 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_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_CONTEXT_POINTER	Invalid context pointer
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	
CSSM_CSP_DEC_BAD_KEY_LENGTH	

See Also

CSP_DecryptData, CSP_DecryptDataUpdate, CSP_DecryptDataFinal

2.3.25 CSP_DecryptDataUpdate

CSSM_RETURN CSSMSPI CSP_DecryptDataUpdate (CSSM_CSP_HANDLE CSPHandle,
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 **CSP_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 **CSP_DecryptUpdate** calls.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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 one or more CSSM_DATA structures containing the encrypted data.

CipherBufCount (input)
The number of *CipherBufs*.

ClearBufs (output)
A pointer to one or more CSSM_DATA structures for the decrypted data. The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate spaces; application has to free the memory in this case. If this is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

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_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
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

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

CSP_QuerySize, CSP_DecryptData, CSP_DecryptDataInit, CSP_DecryptDataFinal

2.3.26 CSP_DecryptDataFinal

CSSM_RETURN CSSMSPI CSP_DecryptDataFinal (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR RemData)

This function finalizes the staged decrypt function.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

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_CSP_HANDLE	Invalid CSP handle
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

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

CSP_DecryptData, CSP_DecryptDataInit, CSP_DecryptDataUpdate

2.3.27 CSP_GenerateKey

CSSM_RETURN CSSMSPI CSP_GenerateKey (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
CSSM_BOOL StoreKey,
CSSM_KEY_PTR Key)

This function generates a symmetric key.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

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_CSP_HANDLE	Invalid CSP handle
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

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

CSP_GenerateRandom, CSP_GenerateKeyPair

2.3.28 CSP_GenerateKeyPair

CSSM_RETURN CSSMSPI CSP_GenerateKeyPair(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
CSSM_BOOL StorePublicKey,
CSSM_KEY_PTR PublicKey,
CSSM_KEY_PTR PrivateKey)

This function generates an asymmetric key pair.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

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_CSP_HANDLE	Invalid CSP handle
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

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

CSP_GenerateRandom, CSP_GenerateKey

2.3.29 CSP_GenerateRandom

CSSM_RETURN CSSMSPI CSP_GenerateRandom (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
CSSM_DATA_PTR RandomNumber)

This function generates random data.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

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_CSP_HANDLE	Invalid CSP handle
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_CONTEXT_POINTER	Invalid context pointer
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 random number

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.

2.3.30 CSP_GenerateUniqueId

CSSM_RETURN CSSMSPI CSP_GenerateUniqueId (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
CSSM_DATA_PTR UniqueID)

This function generates unique identification code.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

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_CSP_HANDLE	Invalid csp handle
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_CONTEXT_POINTER	Invalid context pointer
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.

2.3.31 CSP_WrapKey

CSSM_RETURN CSSMSPI CSP_WrapKey (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
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

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

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

CSP_UnwrapKey

2.3.32 CSP_UnwrapKey

CSSM_RETURN CSSMSPI CSP_UnwrapKey
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
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

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

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

CSP_WrapKey

2.3.33 CSP_DeriveKey

CSSM_RETURN CSSMSPI CSP_DeriveKey (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
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

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

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

2.3.34 CSP_KeyExchGenParam

CSSM_RETURN CSSMSPI CSP_KeyExchGenParam (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
uint32 ParamBits,
CSSM_DATA_PTR Param)

This function generates key exchange parameter data for CSP_KeyExchPhase1.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

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_CSP_HANDLE	Invalid CSP handle
CSSM_CSP_INVALID_CONTEXT_HANDLE	Invalid context handle
CSSM_CSP_INVALID_CONTEXT_POINTER	Invalid context pointer
CSSM_CSP_MEMORY_ERROR	Not enough memory to allocate
CSSM_CSP_KEYEXCH_GENPARAM_FAIL	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

CSP_KeyExchPhase1, CSP_KeyExchPhase2

2.3.35 CSP_KeyExchPhase1

CSSM_RETURN CSSMSPI CSP_KeyExchPhase1 (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR Param,
CSSM_DATA_PTR Param1)

Phase 1 of the key exchange operation — generates data for CSP_KeyExchPhase2.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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 CSP_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_CSP_HANDLE	Invalid CSP handle
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

CSP_KeyExchGenParam, CSP_KeyExchPhase2

2.3.36 CSP_KeyExchPhase2

CSSM_RETURN CSSMSPI CSP_KeyExchPhase2 (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR Param1,
CSSM_KEY_PTR ExchangedKey)

Phase 2 of the key exchange operation.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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 CSP_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_CSP_HANDLE	Invalid CSP handle
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

CSP_KeyExchPhase1, CSP_KeyExchGenParam

2.4 Cryptographic Sessions and Logon

2.4.1 CSP_Login

CSSM_RETURN CSSMSPI 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

CSP_ChangeLoginPassword, CSP_Logout

2.4.2 CSP_Logout

CSSM_RETURN CSSMSPI 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

CSP_Login, CSP_ChangeLoginPassword

2.4.3 CSP_ChangeLoginPassword

CSSM_RETURN CSSMSPI 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

CSP_Login, CSP_Logout

2.5 Extensibility Functions

The 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. CSP developers may also define their own module-specific error codes, as described in Section 3.5.2.

2.5.1 CSP_PassThrough

CSSM_RETURN CSSMSPI CSP_PassThrough (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
uint32 PassThroughId,
const CSSM_DATA_PTR InData,
CSSM_DATA_PTR OutData)

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

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

Context (input)
Pointer to CSSM_CONTEXT structure that describes the attributes associated with this context.

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_FAILED	Unable to perform custom function

2.6 Module Management Functions

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

2.6.1 CSP_Initialize

CSSM_RETURN CSSMSPI CSP_Initialize (uint32 VerMajor,
uint32 VerMinor)

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

Parameters

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

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

Return Value

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

Error Codes

Value	Description
CSSM_CSP_INITIALIZE_FAIL	Unable to perform module initialization

See Also

CSP_Uninitialize

2.6.2 CSP_Uninitialize

CSSM_RETURN CSSMSPI CSP_Uninitialize (void)

This function performs any module-specific cleanup activities.

Parameters

None

Return Value

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

Error Codes

Value	Description
CSSM_CSP_UNINITIALIZE_FAIL	Unable to perform module cleanup

See Also

CSP_Initialize

2.6.3 CSP_GetCapabilities

CSSM_CSPINFO_PTR CSSMSPI CSP_GetCapabilities(CSSM_CSP_HANDLE CSPHandle,
CSSM_BOOL CompleteCapabilitiesOnly,
uint32 *CSPInfoCount)

This function is called by the CSSM when the registry indicates that capabilities information for a CSP is dynamic.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

CompleteCapabilitiesOnly (input)
Boolean flag that indicates whether all devices controlled by the CSP should be represented in the return list. If TRUE, all devices are listed regardless of availability. If FALSE, only devices that are available for use are listed.

CSPInfoCount (output)
The number of CSSM_CSPINFO structures returned. One structure should be returned for each device controlled by the CSP.

Return Value

The return value is an array of CSSM_CSPINFO structures, with the length returned in the CSPInfoCount parameter. If CSPInfoCount is zero, the return value will be NULL.

Error Codes

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

See Also

CSP_EventNotify

2.6.4 CSP_EventNotify

CSSM_RETURN CSSMSPI CSP_EventNotify (CSSM_CSP_HANDLE CSPHandle,
const CSSM_EVENT_TYPE Event,
const uint32 Param)

Called by the CSSM when an event that could impact the internal state of a CSP takes place.

Parameters

CSPHandle (input)
The handle that describes the add-in cryptographic service provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

Event (input)
One of the event types listed below.

Param (input)
This value will vary depending on the type of event. In the case where no parameter is required, this value will be zero.

Return Value

The return value from this function has varying effects based on the event type. In most cases the value CSSM_OK should be returned so indicate that the CSSM can continue. The value CSSM_FAIL should be returned in cases of fatal errors within the CSP.

Event Types

Event	Description
CSSM_EVENT_ATTACH	An attach to the token is taking place. The CSP handle passed to the function is the new handle that will be returned to the application. This event will take place after the initial call to CSP_Initialize. Returning CSSM_FAIL results in a failure of the CSSM_CSP_Attach call.
CSSM_EVENT_DETACH	A detach from the token is taking place. The CSP handle passed to the function is a handle that will have been the subject of a previous CSSM_EVENT_ATTACH event. This event will take place immediately before the call to CSP_Uninitialize when the handle being detached is the only handle associated with that CSP. Returning CSSM_FAIL has no effect.

CSSM_EVENT_INFOATTACH	An attach to the token is taking place in order to get the capabilities list for the CSP. The CSP handle passed to the function is a temporary handle created for the specific purpose of calling CSP_GetCapabilities. This event will take place without a call to CSP_Initialize. When this event is received, only the minimal amount of initialization required to successfully perform a CSP_GetCapabilities call should be performed. Returning CSSM_FAIL results in a failure of the attach.
CSSM_EVENT_INFODETACH	A detach from the token is taking place. The CSP handle passed to the function is a handle that will have been the subject of a previous CSSM_EVENT_INFOATTACH event. This event will never be followed by a call to CSP_Uninitialize when the handle being detached is the only handle associated with that CSP. Returning CSSM_FAIL has no effect.

See Also

CSP_GetCapabilities, CSSM_CSP_Attach, CSSM_CSP_Detach

3. CSP Structure and Management

3.1 Introduction

A CSP is an add-in module which can be used by applications via CSSM to perform cryptographic services.
There exists today a variety of cryptographic protocols, techniques, and algorithms. Even for the same cryptographic algorithm there exist variants based on key lengths, padding schemes, and algorithm modes. Because all algorithm and key-specific information is encapsulated in the CSP, the application can focus on interesting uses of cryptography, rather than the tedious details of algorithm variations and key formats. The availability of CSPs also allows CSP developers to easily customize and extend the cryptographic protocols to meet changing market requirements.

This section is provided to aid the CSP developers in creating a CSP module which will interface properly with CSSM. It covers the structure of a CSP, CSP installation, the expected behavior of a CSP on attach, and some behaviors expected of CSP modules. This section also includes examples of CSP function implementations as a reference for new CSP modules.

3.2 CSP Structure

A CSP is a dynamically-linkable library which contains routines which implement some or all of the CSSM SPI described in Section 2. The CSP should also contain functions which are called when the CSP is attached and detached. The attach function will be responsible for registering a function table with CSSM, accepting the memory management upcalls, and performing any module-specific setup. The detach function will be responsible for any cleanup required by the module. The attach and detach functions will vary depending on the target operating system. For example, DLLMain would be used to implement these functions for a CSP targeted to Windows NT*. _init and _fini would be used to implement these functions for a CSP targeted to SunOS*.

The CSP functionality can be broadly classified into the following categories:

- Registration with CSSM
- Token management
- Private key management
- Cryptographic services
- Other services

A CSP may implement all or some of the components listed above. A CSP need not expose all the functions for every component. A CSP vendor can expose other service functions through the CSP_PassThrough interface. A unique function ID is required to identify the custom function.

3.3 CSP Installation

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

3.3.1 Global Unique Identifiers (GUIDs)

Each CSP must have a globally-unique identifier (GUID) which will be used by CSSM, applications, and CSP modules to uniquely identify a CSP. The GUID will be used by the CSSM registry to expose add-in module availability to applications. The application will use this GUID to identify a targeted CSP in all cryptographic function calls. The CSP module will use this GUID to identify itself when it sets an error. GUID generators are publicly available for Windows 95*, Windows NT, and many UNIX* platforms.

A GUID is defined as:

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

3.4 Attaching a CSP

Before an application can use the functions of a specific CSP, it must attach the CSP to CSSM using the *CSSM_CSP_Attach* function. On attach, the CSP uses the *CSSM_CSP_RegisterServices* function to register its function table with CSSM. CSSM will use the CSP module's function table to direct calls from the application to the correct function in the CSP module. During the attach process, the CSP's *CSP_Initialize* function is called. At this time version compatibility is confirmed and a table of memory function upcalls is passed to the CSP. The CSP module uses the memory management upcalls to allocate any memory which will be returned to the calling application, and to free any memory that it received from the calling application.

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

3.4.1 The CSP module function table

The function table for a CSP module is a structure which contains pointers to the CSP module's implementation of the functions specified in the Service Provider Interface. This structure is specified as a part of the CSSM header file, cssm.h. If a CSP does not support some function in the SPI, the pointer to that function should be set to NULL.

3.4.2 Memory management upcalls

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

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

3.5 CSP Basic Services

3.5.1 Function Implementation

A CSP developer may choose to implement some or all of the functions specified in the SPI. The expected behavior of each function is detailed in Section 2, *Service Provider Interface*.

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

3.5.2 Error handling

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

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

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

3.6 CSP Utility Libraries

CSP Utility Libraries are software components that may be provided by a CSP developer for use by other CSP developers. They are expected to contain functions that may be useful to several CSP modules, such as BER and DER encoding and decoding.

A CSP may want its public/private key blobs to be PKCS-conformant. The following functions might be provided by the CSP utility library:

- Pkcs_MakePublicKeyBlob
- Pkcs_MakePrivateKeyBlob
- Pkcs_ConvPublicKeyBlob
- Pkcs_ConvPrivateKeyBlob

The CSP Utility Library developer is responsible for making the definition, interpretation, and usage of their library available to other CSP module developers.

3.7 Attach/Detach Example

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

3.7.1 DLLMain

```
#include "cssm.h"
CSSM_GUID csp_guid =
{ 0x83bafc39, 0xfac1, 0x11cf, { 0x81, 0x72, 0x0, 0xaa, 0x0, 0xb1, 0x99, 0xdd }
};

BOOL WINAPI DllMain ( HANDLE hInstance, DWORD dwReason, LPVOID lpReserved)
{
    switch (dwReason)
    {
        case DLL_PROCESS_ATTACH:
        {
            CSSM_SPI_MEMORY_FUNCS MemoryFunctions;
            CSSM_FUNCTIONTABLE FunctionTable;

            /* Fill in FunctionTable with function pointers */
            FunctionTable.QuerySize          = CSP_QuerySize;
            FunctionTable.GetCapabilities    = CSP_GetCapabilities;
            FunctionFuncTable.SignData       = CSP_SignData;
            FunctionTable.SignDataInit       = CSP_SignDataInit;
            FunctionTable.SignDataUpdate     = CSP_SignDataUpdate;
            FunctionTable.SignDataFinal      = CSP_SignDataFinal;
            FunctionTable.VerifyData         = CSP_VerifyData;
            FunctionTable.VerifyDataInit     = CSP_VerifyDataInit;
            FunctionTable.VerifyDataUpdate   = CSP_VerifyDataUpdate;
            FunctionTable.VerifyDataFinal    = CSP_VerifyDataFinal;
            FunctionTable.DigestData         = CSP_DigestData;
            FunctionTable.DigestDataInit     = CSP_DigestDataInit;
            FunctionTable.DigestDataUpdate   = CSP_DigestDataUpdate;
            FunctionTable.DigestDataClone    = CSP_DigestDataClone;
            FunctionTable.DigestDataFinal    = CSP_DigestDataFinal;
            FunctionTable.GenerateMac        = CSP_GenerateMac;
            FunctionTable.GenerateMacInit    = CSP_GenerateMacInit;
            FunctionTable.GenerateMacUpdate  = CSP_GenerateMacUpdate;
            FunctionTable.GenerateMacFinal   = CSP_GenerateMacFinal;
            FunctionTable.EncryptData        = CSP_EncryptData;
            FunctionTable.EncryptDataInit    = CSP_EncryptDataInit;
            FunctionTable.EncryptDataUpdate  = CSP_EncryptDataUpdate;
            FunctionTable.EncryptDataFinal   = CSP_EncryptDataFinal;
            FunctionTable.DecryptData        = CSP_DecryptData;
            FunctionTable.DecryptDataInit    = CSP_DecryptDataInit;
            FunctionTable.DecryptDataUpdate  = CSP_DecryptDataUpdate;
            FunctionTable.DecryptDataFinal   = CSP_DecryptDataFinal;
            FunctionTable.GenerateKey         = CSP_GenerateKey;
            FunctionTable.GenerateKeyPair     = CSP_GenerateKeyPair;
            FunctionTable.DeriveKey          = CSP_DeriveKey;
            FunctionTable.WrapKey            = CSP_WrapKey;
            FunctionTable.UnwrapKey          = CSP_UnwrapKey;
```

```
            FunctionTable.GenerateRandom     = CSP_GenerateRandom;
            FunctionTable.GenerateUniqueId   = CSP_GenerateUniqueId;
            FunctionTable.KeyExchGenParam    = CSP_KeyExchGenParam;
            FunctionTable.KeyExchPhase1      = CSP_KeyExchPhase1;
            FunctionTable.KeyExchPhase2      = CSP_KeyExchPhase2;
            FunctionTable.PassThrough        = CSP_PassThrough;
            FunctionTable.Initialize         = CSP_Initialize;
            FunctionTable.Uninitialize       = CSP_Uninitialize;
            FunctionTable.EventNotify        = CSP_EventNotify;

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

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

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

3.8 Cryptographic Operations Examples

```
CSSM_RETURN CSSMSPI CSP_GenerateKeyPair (CSSM_CSP_HANDLE CSPHandle,
                                         CSSM_CC_HANDLE CCHandle,
                                         const CSSM_CONTEXT_PTR Context,
                                          CSSM_BOOL StorePublicKey,
                                         CSSM_KEY_PTR PublicKey,
                                          CSSM_KEY_PTR PrivateKey)
{
    CSP_SESSION   session;
    uint32 rtn;

    rtn = l_ValidateContextParam(Context);
    if (rtn != CSSM_OK)
        return rtn;

    /* Create a temp session and fill the information */
    Token_InitSession(&session);
    Token_FillSession(&session, CSPHandle, CCHandle, Context);

    /* calls crypto func to generate key pair, return the key blobs,
       and save the wrapped prikey in the token (in the asymmetric
       key pair generation case) */
    return Cryp_GenerateKeyPair(session, PublicKey, PrivateKey);
}
```

4. Appendix A. Relevant CSSM API functions

4.1 Overview

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

4.2 Function Definitions

4.2.1 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

4.2.2 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

4.2.3 CSSM_CSP_RegisterServices

CSSM_RETURN CSSMAPI CSSM_CSP_RegisterServices
(const CSSM_GUID_PTR GUID,
const CSSM_SPI_CSP_FUNCS_PTR FunctionTable,
CSSM_SPI_MEMORY_FUNCS_PTR UpcallTable,
void *Reserved)

A CSP module uses this function to register its function table with CSSM and to receive a memory management upcall table from CSSM.

Parameters

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

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

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

Reserved (input)
A reserved input.

Return Value

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

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_INVALID_FUNCTION_TABLE	Invalid function table
CSSM_MEMORY_ERROR	Memory error
CSSM_REGISTRY_ERROR	Unable to register services

See Also

CSSM_CSP_DeregisterServices

4.2.4 CSSM_CSP_DeregisterServices

CSSM_RETURN CSSMAPI CSSM_CSP_DeregisterServices (const CSSM_GUID_PTR GUID)

A CSP module uses this function to deregister its services from the CSSM.

Parameters

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

Return Value

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

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer GUID
CSSM_MEMORY_ERROR	Unable to deregister services

See Also

CSSM_CSP_RegisterServices

4.2.5 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

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

4.2.6 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

4.2.7 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

4.2.8 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 specified 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

4.2.9 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

4.2.10 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

4.2.11 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 is CSSM_OK. A NULL pointer indicates the CSSM_InitError was not called by the CSSM Core or that CSSM Core made a call to CSSM_DestroyError. No error information is available.

See Also

CSSM_ClearError, CSSM_SetError

4.2.12 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 falls 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 the error number passed is not within a valid range, the GUID passed is invalid, CSSM_InitError was not called by the CSSM Core, or the CSSM core called CSSM_DestroyError. No error information is available.

See Also

CSSM_ClearError, CSSM_GetError

4.2.13 CSSM_ClearError

void CSSMAPI CSSM_ClearError (void)

This function sets the current error value to CSSM_OK. This is 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