IBM KeyWorks Toolkit

Cryptographic Service Provider Interface (SPI) Specification

June 11, 1999

001.001.004

Table of Contents

CHAPTER 1. INTRODUCTION 1

1.1 SERVICE PROVIDER MODULES 1
1.2 INTENDED AUDIENCE 2
1.3 DOCUMENTATION SET 2
1.4 REFERENCES 3

CHAPTER 2. SERVICE PROVIDER INTERFACE 5

2.1 CRYPTOGRAPHIC OPERATIONS 5
2.2 CRYPTOGRAPHIC LOGON AND SESSIONS 8
2.3 EXTENSIBILITY FUNCTIONS 8
2.4 DATA STRUCTURES 8
2.4.1 CSSM_BOOL 8
2.4.2 CSSM_CALLBACK 8
2.4.3 CSSM_CONTEXT 9
2.4.4 CSSM_CONTEXT_ATTRIBUTE 14
2.4.5 CSSM_CONTEXTINFO 16
2.4.6 CSSM_CRYPTO_DATA 16
2.4.7 CSSM_CSP_CAPABILITY 16
2.4.8 CSSM_CSP_FLAGS 16
2.4.9 CSSM_CSP_HANDLE 16
2.4.10 CSSM_CSPSUBSERVICE 17
2.4.11 CSSM_CSPTYPE 18
2.4.12 CSSM_CSP_WRAPPEDPRODUCTINFO 18
2.4.13 CSSM_DATA 19
2.4.14 CSSM_DATE 19
2.4.15 CSSM_HARDWARECSPSUBSERVICEINFO 20
2.4.16 CSSM_HEADERVERSION 22
2.4.17 CSSM_KEY 22
2.4.18 CSSM_KEYHEADER 23
2.4.19 CSSM_KEY_SIZE 26
2.4.20 CSSM_KEY_TYPE 26
2.4.21 CSSM_NOTIFY_CALLBACK 26
2.4.22 CSSM_PADDING 27
2.4.23 CSSM_QUERY_SIZE_DATA 27
2.4.24 CSSM_RANGE 27
2.4.25 CSSM_SOFTWARECSPSUBSERVICEINFO 28
2.4.26 CSSM_SPI_FUNC_TBL 28
2.5 CRYPTOGRAPHIC OPERATIONS 29
2.5.1 CSP_DecryptData 29
2.5.2 CSP_DecryptDataFinal 31
2.5.3 CSP_DecryptDataInit 32
2.5.4 CSP_DecryptDataUpdate 33
2.5.5 CSP_DeriveKey 34
2.5.6 CSP_DigestData 35
2.5.7 CSP_DigestDataClone 36
2.5.8 CSP_DigestDataFinal 37
2.5.9 CSP_DigestDataInit 38
2.5.10 CSP_DigestDataUpdate 39
2.5.11 CSP_EncryptData 40
2.5.12 CSP_EncryptDataFinal 42
2.5.13 CSP_EncryptDataInit 43

2.5.14 CSP_EncryptDataUpdate *44*
2.5.15 CSP_GenerateAlgorithmParams *45*
2.5.16 CSP_GenerateKey *46*
2.5.17 CSP_GenerateKeyPair *47*
2.5.18 CSP_GenerateMac *49*
2.5.19 CSP_GenerateMacFinal *50*
2.5.20 CSP_GenerateMacInit *51*
2.5.21 CSP_GenerateMacUpdate *52*
2.5.22 CSP_GenerateRandom *53*
2.5.23 CSP_QueryKeySizeInBits *54*
2.5.24 CSP_QuerySize *55*
2.5.25 CSP_SignData *56*
2.5.26 CSP_SignDataFinal *57*
2.5.27 CSP_SignDataInit *58*
2.5.28 CSP_SignDataUpdate *59*
2.5.29 CSP_UnwrapKey *60*
2.5.30 CSP_VerifyData *61*
2.5.31 CSP_VerifyDataFinal *62*
2.5.32 CSP_VerifyDataInit *63*
2.5.33 CSP_VerifyDataUpdate *64*
2.5.34 CSP_VerifyMac *65*
2.5.35 CSP_VerifyMacFinal *66*
2.5.36 CSP_VerifyMacInit *67*
2.5.37 CSP_VerifyMacUpdate *68*
2.5.38 CSP_WrapKey *69*
2.6 CRYPTOGRAPHIC SESSIONS AND LOGON 70
2.6.1 CSP_ChangeLoginPassword *70*
2.6.2 CSP_Login *71*
2.6.3 CSP_Logout *72*
2.7 EXTENSIBILITY FUNCTIONS 73
2.7.1 CSP_PassThrough *73*

CHAPTER 3. CRYPTOGRAPHIC SERVICE PROVIDER FUNCTION EXAMPLES 74

3.1 ATTACH/DETACH EXAMPLE 74
3.1.1 AddInAuthenticate *75*
3.2 EXTENSIBILITY FUNCTIONS EXAMPLES 76
3.2.1 CSP_PassThrough *76*

APPENDIX A. IBM KEYWORKS ERRORS 79

A.1 CRYPTOGRAPHIC SERVICE PROVIDER MODULE ERRORS 80

APPENDIX B. LIST OF ACRONYMS 86

APPENDIX C. GLOSSARY 87

List of Figures

Figure 1. IBM KeyWorks Toolkit Architecture 2

List of Tables

Table 1. Context Types 9
Table 2. Algorithms for a Session Context 10
Table 3. Modes of Algorithms 12
Table 4. Attribute Types 14
Table 5. CSSM Sessions 16
Table 6. CSP Flags 17
Table 7. CSP Information Type Identifiers and Associated Structure Types 18
Table 8. PKCS#11 CSP Reader Flags 21
Table 9. PKCS#11 CSP Token Flags 21
Table 10. Key Blob Type Identifiers 23
Table 11. Key Blob Format Identifiers 24
Table 12. Key Class Identifiers 24
Table 13. Key Attribute Flags 25
Table 14. Key Usage Flags 25
Table 15. CSSM_NOTIFY Reason Values 27
Table 16. CSP Module Error Numbers 79
Table 17. General CSP Messages and Errors 80
Table 18. CSP Memory Errors 80
Table 19. Invalid CSP Parameters 80
Table 20. File I/O Errors 81
Table 21. CSP Cryptographic Errors 81
Table 22. Missing or Invalid CSP Parameters 82
Table 23. Password Errors 82
Table 24. Key Management Messages and Errors 82
Table 25. Random Number Generation (RNG) Messages and Errors 83
Table 26. Unique ID Generation Messages and Errors 83
Table 27. Key Generation Messages and Errors 83
Table 28. Encryption/Decryption Messages 83
Table 29. Sign/Verify Messages and Errors 84
Table 30. Digest Function Errors 84
Table 31. MAC Function Errors 84
Table 32. Key Exchange Errors 84
Table 33. PassThrough Custom Errors 84
Table 34. Wrap/Unwrap Errors 85
Table 35. Hardware CSP Errors 85
Table 36. Query Size Errors 85

Chapter 1. Introduction

The IBM KeyWorks Toolkit defines the infrastructure for a complete set of security services. It is an extensible architecture that provides mechanisms to manage service provider 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 IBM KeyWorks Toolkit: Application Domains, System Security Services, KeyWorks Framework, and Service Providers. The KeyWorks Framework is the core of this architecture. It provides a means for applications to directly access security services through the KeyWorks security application programming interface (API), or to indirectly access security services via layered security services and tools implemented over the KeyWorks API. The IBM KeyWorks Framework manages the service provider security modules and directs application calls through the KeyWorks API to the selected service provider module that will service the request. The KeyWorks API defines the interface for accessing security services. The KeyWorks service provider interface (SPI) defines the interface for service providers who develop plug-able security service products.

Service providers perform various aspects of security services, including:

- Cryptographic Services
- Key Recovery Services
- Trust Policy Libraries
- Certificate Libraries
- Data Storage Libraries

Cryptographic Service Providers (CSPs) are service provider modules that perform cryptographic operations including encryption, decryption, digital signing, key pair generation, random number generation, and key exchange. Key Recovery Service Providers (KRSPs) generate and process Key Recovery Fields (KRFs) which can be used to retrieve the original session key if it is lost, or if an authorized party requires access to the decryption key. Trust Policy (TP) modules implement policies defined by authorities and institutions, such as VeriSign (as a Certificate Authority (CA)) or MasterCard (as an institution). Each TP 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 (CRLs). Data Storage Library (DL) modules provide persistent storage for certificates and CRLs.

1.1 Service Provider Modules

An IBM KeyWorks service provider module is a Dynamically Linked Library (DLL) composed of functions that implement some or all of the KeyWorks module interfaces. Applications directly or indirectly select the modules used to provide security services to the application. These service providers will be provided by Independent Software Vendors (ISVs) and hardware vendors. The functionality of the service providers may be extended beyond the services defined by the KeyWorks API, by exporting additional services to applications using an KeyWorks PassThrough mechanism.

The API calls defined for service provider 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 CRL 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 passthrough functions whose behavior and use is defined by the service provider module developer.

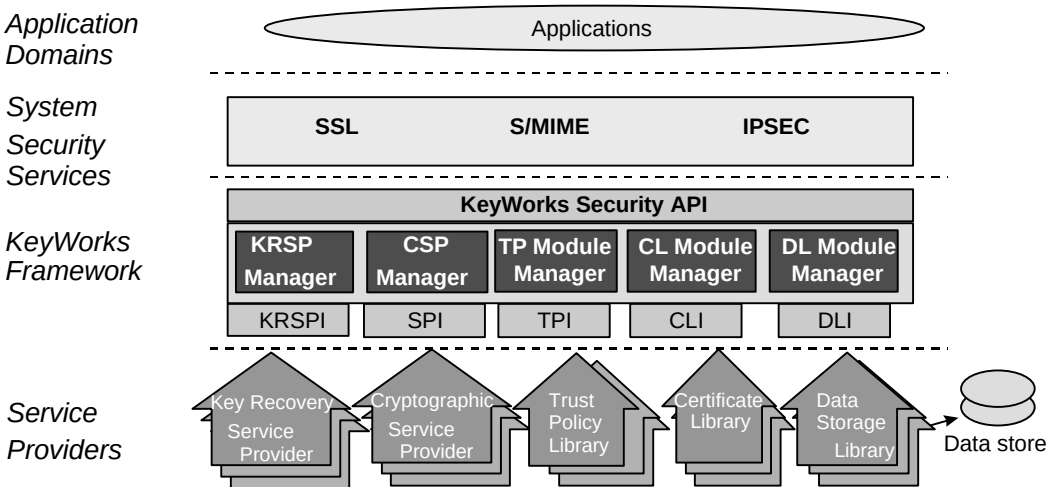

Figure 1. **IBM KeyWorks Toolkit Architecture**

Each module, regardless of the security services it offers, has the same set of module management responsibilities. Every module must expose functions that allow KeyWorks to indicate events such as module attach and detach. In addition, as part of the attach operation, every module must be able to verify its own integrity, verify the integrity of KeyWorks, and register with KeyWorks. Detailed information about service provider module structure, administration, and interfaces can be found in the *IBM KeyWorks Service Provider Module Structure & Administration Specification.*

1.2 Intended Audience

This document should be used by ISVs who want to develop their own TP service provider modules. These ISVs can be highly experienced software and security architects, advanced programmers, and sophisticated users. The intended audience of this document must be familiar with high-end cryptography and digital certificates. They must also be familiar with local and foreign government regulations on the use of cryptography and the implication of those regulations for their applications and products. We assume that this audience is familiar with the basic capabilities and features of the protocols they are considering.

1.3 Documentation Set

The IBM KeyWorks Toolkit documentation set consists of the following manuals. These manuals are provided in electronic format and can be viewed using the Adobe Acrobat Reader distributed with the IBM KeyWorks Toolkit. Both the electronic manuals and the Adobe Acrobat Reader are located in the IBM KeyWorks Toolkit `doc` subdirectory.

- *IBM KeyWorks Toolkit Developer's Guide*
  Document filename: kw_dev.pdf
  This document presents an overview of the IBM KeyWorks Toolkit. It explains how to integrate IBM KeyWorks into applications and contains a sample IBM KeyWorks application.

- *IBM KeyWorks Toolkit Application Programming Interface Specification*
  Document filename: kw_api.pdf
  This document defines the interface that application developers employ to access security services provided by IBM KeyWorks and service provider modules.

- *IBM KeyWorks Toolkit Service Provider Module Structure & Administration Specification*.
  Document filename: kw_mod.pdf
  This document describes the features common to all IBM KeyWorks service provider modules. It should be used in conjunction with the IBM KeyWorks service provider interface specifications in order to build a security service provider module.

- *IBM KeyWorks Toolkit Cryptographic Service Provider Interface Specification*
  Document filename: kw_spi.pdf
  This document defines the interface to which cryptographic service providers must conform in order to be accessible through IBM KeyWorks.

- *Key Recovery Service Provider Interface Specification*
  Document filename: kr_spi.pdf
  This document defines the interface to which key recovery service providers must conform in order to be accessible through IBM KeyWorks.

- *Key Recovery Server Installation and Usage Guide*
  Document filename: krs_gd.pdf
  This document describes how to install and use key recovery solutions using the components in the IBM Key Recovery Server.

- *IBM KeyWorks Toolkit Trust Policy Interface Specification*
  Document filename: kw_tp_spi.pdf
  This document defines the interface to which policy makers, such as certificate authorities, certificate issuers, and policy-making application developers, must conform in order to extend IBM KeyWorks with model or application-specific policies.

- *IBM KeyWorks Toolkit Certificate Library Interface Specification*
  Document filename: kw_cl_spi.pdf
  This document defines the interface to which library developers must conform to provide format-specific certificate manipulation services to numerous IBM KeyWorks applications and trust policy modules.

- *IBM KeyWorks Toolkit Data Storage Library Interface Specification*
  Document filename: kw_dl_spi.pdf
  This document defines the interface to which library developers must conform to provide format-specific or format-independent persistent storage of certificates.

1.4 References

Cryptography	*Applied Cryptography*, Schneier, Bruce**,** 2nd Edition, John Wiley and Sons, Inc., 1996.
	Handbook of Applied Cryptography, Menezes, A., Van Oorschot, P., and Vanstone, S., CRC Press, Inc., 1997.
	SDSI - A Simple Distributed Security Infrastructure, R. Rivest and B. Lampson, 1996.
	Microsoft CryptoAPI, Version 0.9, Microsoft Corporation, January 17, 1996.
CDSA Spec	*Common Data Security Architecture Specification*, Intel Architecture Labs, 1997.

CSSM API	*Common Security Services Manager Application Programming Interface Specification,* Intel Architecture Labs, 1997.
Key Escrow	*A Taxonomy for Key Escrow Encryption Systems,* Denning, Dorothy E. and Branstad, Dennis, Communications of the ACM, Vol. 39, No. 3, March 1996.
PKCS	*The Public-Key Cryptography Standards*, RSA Laboratories, Redwood City, CA: RSA Data Security, Inc.
IBM KeyWorks CLI	*Certificate Library Interface Specification,* Intel Architecture Labs, 1997.
IBM KeyWorks DLI	*Data Storage Library Interface Specification,* Intel Architecture Labs, 1997.
IBM KeyWorks KRI	*Key Recovery Service Provider Interface Specification,* Intel Architecture Labs, 1997.
IBM KeyWorks SPI	*Cryptographic Service Provider Interface Specification,* Intel Architecture Labs, 1997.
IBM KeyWorks TPI	*Trust Policy Interface Specification*, Intel Architecture Labs, 1997.
X.509	*CCITT. Recommendation X.509: The Directory – Authentication Framework,* 1988. CCITT stands for Comite Consultatif Internationale Telegraphique et Telephonique (International Telegraph and Telephone Consultative Committee)

Chapter 2. Service Provider Interface

Cryptographic Service Providers (CSPs) are add-in modules which perform cryptographic operations including encryption, decryption, digital signing, key pair generation, random number generation, message digest, and key exchange. Besides the traditional cryptographic functions, CSPs may provide other vendor-specific services.

The range and types of services a CSP supports are at the discretion of the vendor. A registry and query mechanism is available through the IBM KeyWorks for CSPs to disclose the services and details about the services. All cryptographic services requested by applications will be channeled to one of the CSPs via the KeyWorks. CSP vendors only need target their modules to KeyWorks for all security-conscious applications to have access to their product.

Calls made to a 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 also are 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 are available 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 cryptographic operations. Staged encryption/decryption are an exception in that each update call generates a portion of the result.

2.1 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_RETURN 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 Message Authentication Code (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_VerifyMac
CSSM_RETURN CSP_VerifyMacInit
CSSM_RETURN CSP_VerifyMacUpdate
CSSM_RETURN CSP_VerifyMacFinal

Accepts as input a handle to a cryptographic context describing the MAC operation and the data to operate on. The result of the completed verify operation is a CSSM_RETURN value.

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_QueryKeySizeInBits

Accepts as input a handle to a cryptographic context and the key. This function returns a pointer to a data structure containing the keysize and effective keysize in bits.

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 CSSM_GenerateKeyPair

Accepts as input a handle to a cryptographic context describing the generate key pair operation. The keys returned are in 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_GenerateAlgorithmParams

Accepts as input a handle to a cryptographic context describing an algorithm and returns a set of algorithm parameters appropriate for that algorithm.

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.

2.2 Cryptographic Logon and Sessions

CSSM_RETURN CSP_Login

Accepts as input a login password and a flag indicating the persistent or nonpersistent 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.

2.3 Extensibility Functions

void * CSP_PassThrough

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.

2.4 Data Structures

This section describes the data structures that 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 KeyWorks API function calls, which 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 through KeyWorks. These data structures are defined in the header file, cssmtype.h, which is distributed with the IBM KeyWorks Toolkit.

2.4.1 CSSM_BOOL

```
typedef uint32 CSSM_BOOL;

#define CSSM_TRUE   1
#define CSSM_FALSE  0
```

2.4.2 CSSM_CALLBACK

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

Definitions:

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

ID - Input data to identify the callback.

2.4.3 CSSM_CONTEXT

```
typedef struct cssm_context {
    uint32 ContextType;
    uint32 AlgorithmType;
    uint32 Reserve;
    uint32 NumberOfAttributes;
    CSSM_CONTEXT_ATTRIBUTE_PTR ContextAttributes;
    CSSM_BOOL Privileged;
    uint32 EncryptionProhibited;
    uint32 WorkFactor;
} CSSM_CONTEXT, *CSSM_CONTEXT_PTR
```

Definitions:

ContextType - An identifier describing the type of services for this context. Table 1 provides the context types.

Table 1. 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
CSSM_ALGCLASS_KEY_RECOVERY_ENABLEMENT	Key Recovery Enablement algorithms
CSSM_ALGCLASS_KEY_RECOVERY_REGISTRATION	Key Recovery Registration algorithms
CSSM_ALGCLASS_KEY_RECOVERY_REQUEST	Key Recovery Request algorithms

AlgorithmType - An ID number describing the algorithm to be used. Table 2 provides the algorithms for a session context.

Table 2. 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	MD2hash algorithm
CSSM_ALGID_MD4	MD4hash algorithm
CSSM_ALGID_MD5	MD5hash 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	International Data Encryption Algorithm (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

Value	Description
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-40, K-64, K-128 block cipher
CSSM_ALGID_CRAB	CRAB block cipher
CSSM_ALGID_RSA	RSA public key cipher
CSSM_ALGID_DSA	Digital Signature Algorithm (DSA)
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 Public-Key Cryptographic Standard (PKCS#11)
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 Secure Sockets Layer (SSL) 3 premaster key

Value	Description
CSSM_ALGID_SSL3MasterDerive	Derive an SSL 3 key from a premaster 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	Secure Electronic Transaction (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
CSSM_ALGID_3DES	Generic 3DES
CSSM_ALGID_SSL3MD5	SSL3MD5
CSSM_ALGID_SSL3SHA1	SSL3SHA1
CSSM_ALGID_FortezzaTimestamp	FortezzaTimestamp
CSSM_ALGID_SHA1WithDSA	SHA1WithDSA
CSSM_ALGID_SHA1WithECDSA	SHA1WithECDSA
CSSM_ALGID_DSA_BSAFE	BSAFE Key format

Some of the algorithms above in Table 2 operate in a variety of modes. The desired mode is specified using an attribute of type CSSM_ATTRIBUTE_MODE. The valid values for the mode attribute are as follows in Table 3.

Table 3. Modes of Algorithms

Value	Description
CSSM_ALGMODE_NONE	Null Algorithm mode
CSSM_ALGMODE_CUSTOM	Custom mode
CSSM_ALGMODE_ECB	Electronic Code Book (ECB)
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

Value	Description
CSSM_ALGMODE_CFB	Cipher FeedBack
CSSM_ALGMODE_CFB_IV8	CFB with Initialization Vector of 8 bytes
CSSM_ALGMODE_CFBPadIV8	CFB with Initialization Vector of 8 bytes and padding
CSSM_ALGMODE_OFB	Output FeedBack
CSSM_ALGMODE_OFB_IV8	OFB with Initialization Vector of 8 bytes
CSSM_ALGMODE_OFBPadIV8	OFB with Initialization Vector of 8 bytes and padding
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
CSSM_ALGMODE_ECB64	Electronic Code Book (64 bits)
CSSM_ALGMODE_CBC64	Cipher Block Chaining (64 bits)
CSSM_ALGMODE_OFB64	Output FeedBack (64 bits)
CSSM_ALGMODE_CFB64	Cipher FeedBack (64 bits)
CSSM_ALGMODE_CFB32	Cipher FeedBack (32 bits)
CSSM_ALGMODE_CFB16	Cipher FeedBack (16 bits)
CSSM_ALGMODE_CFB8	Cipher FeedBack (8 bits)
CSSM_ALGMODE_WRAP	SKIPJACK Wrap mechanism
CSSM_ALGMODE_PRIVATE_WRAP	SKIPJACK Private Wrap mechanism
CSSM_ALGMODE_RELAYX	SKIPJACK RELAYX mechanism
CSSM_ALGMODE_ECB128	Electronic Code Book (128 bits)
CSSM_ALGMODE_ECB96	Electronic Code Book (96 bits)
CSSM_ALGMODE_CBC128	Cipher Block Chaining (128 bits)
CSSM_ALGMODE_OAEP_HASH	Optimal Asymmetric Encryption Padding (OAEP) for RSA

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.

Privileged - When this flag is CSSM_TRUE, the context can perform cryptographic operations without being forced to follow the key recovery policy.

EncryptionProhibited - An integer indicating whether encryption is allowed. If encryption is allowed, this field is zero. Otherwise, the flags indicate which policy disallowed encryption.

WorkFactor - WorkFactor is the maximum number of bits that can be left out of Key Recovery Fields (KRFs) when they are generated. The recoverer of the key must then search this number of bits to recover the key.

2.4.4 CSSM_CONTEXT_ATTRIBUTE

```
typedef struct cssm_context_attribute{
    uint32 AttributeType;
    uint32 AttributeLength;
    union {
        char *String;
        uint32 Uint32;
        CSSM_CRYPTO_DATA_PTR Crypto;
        CSSM_KEY_PTR Key;
        CSSM_DATA_PTR Data;
        CSSM_DATE_PTR Date;
        CSSM_RANGE_PTR Range;
        CSSM_VERSION_PTR Version;
        CSSM_KR_PROFILE_PTR KRProfile;
    } Attribute;
} CSSM_CONTEXT_ATTRIBUTE, *CSSM_CONTEXT_ATTRIBUTE_PTR;
```

Definitions:

AttributeType - An identifier describing the type of attribute. Valid attribute types are as follows in Table 4.

Table 4. Attribute Types

Value	Description	Data Type
CSSM_ATTRIBUTE_NONE	No attribute	None
CSSM_ATTRIBUTE_CUSTOM	Custom data	Opaque pointer
CSSM_ATTRIBUTE_DESCRIPTION	Description of attribute	String
CSSM_ATTRIBUTE_KEY	Key Data	CSSM_KEY
CSSM_ATTRIBUTE_INIT_VECTOR	Initialization vector	CSSM_DATA
CSSM_ATTRIBUTE_SALT	Salt	CSSM_DATA
CSSM_ATTRIBUTE_PADDING	Padding information	uint32
CSSM_ATTRIBUTE_RANDOM	Random data	CSSM_DATA
CSSM_ATTRIBUTE_SEED	Seed	CSSM_CRYPTO_DATA

Value	Description	Data Type
CSSM_ATTRIBUTE_PASSPHRASE	Passphrase	CSSM_CRYPTO_DATA
CSSM_ATTRIBUTE_KEY_LENGTH	Key length specified in bits	uint32
CSSM_ATTRIBUTE_KEY_LENGTH_RANGE	Key length range specified in bits	CSSM_RANGE
CSSM_ATTRIBUTE_BLOCK_SIZE	Block size	uint32
CSSM_ATTRIBUTE_OUTPUT_SIZE	Output size	uint32
CSSM_ATTRIBUTE_ROUNDS	Number of runs or rounds	uint32
CSSM_ATTRIBUTE_IV_SIZE	Size of initialization vector	uint32
CSSM_ATTRIBUTE_ALG_PARAMS	Algorithm parameters	CSSM_DATA
CSSM_ATTRIBUTE_LABEL	Label placed on an object when it is created	CSSM_DATA
CSSM_ATTRIBUTE_KEY_TYPE	Type of key to generate or derive	uint32
CSSM_ATTRIBUTE_MODE	Algorithm mode to use for encryption	uint32
CSSM_ATTRIBUTE_EFFECTIVE_BITS	Number of effective bits used in the RC2 cipher	uint32
CSSM_ATTRIBUTE_START_DATE	Starting date for an object's validity	CSSM_DATE
CSSM_ATTRIBUTE_END_DATE	Ending date for an object's validity	CSSM_DATE
CSSM_ATTRIBUTE_KEYUSAGE	Key usage	uint32
CSSM_ATTRIBUTE_KEYATTR	Key attributes	uint32
CSSM_ATTRIBUTE_VERSION	Object version	CSSM_VERSION
CSSM_ATTRIBUTE_ALG_ID	Algorithm ID	uint32
CSSM_ATTRIBUTE_ITERATION_COUNT	Number of iterations	uint32
CSSM_ATTRIBUTE_ROUNDS_RANGE	Minimum and maximum number of rounds	CSSM_RANGE
CSSM_ATTRIBUTE_KRPROFILE_LOCAL	Key Recovery Profile for the local user	CSSM_KR_PROFILE
CSSM_ATTRIBUTE_KRPROFILE_REMOTE	Key Recovery Profile for the remote user	CSSM_KR_PROFILE

The data referenced by a CSSM_ATTRIBUTE_CUSTOM attribute must be a single continuous memory block. This allows the KeyWorks to appropriately release all dynamically allocated memory resources.

AttributeLength - Length of the attribute data.

Attribute - Union representing the attribute data. The union member used is named after the type of data contained in the attribute. See Table 4 for the data types associated with each attribute type.

2.4.5 CSSM_CONTEXTINFO

```
typedef CSSM_CONTEXT CSSM_CONTEXTINFO;
```

2.4.6 CSSM_CRYPTO_DATA

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

Definitions:

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

Callback - An optional callback routine for the service provider modules to obtain the parameter.

ID - A tag that identifies the callback.

2.4.7 CSSM_CSP_CAPABILITY

```
typedef CSSM_CONTEXT CSSM_CSP_CAPABILITY, *CSSM_CSP_CAPABILITY_PTR;
```

2.4.8 CSSM_CSP_FLAGS

```
typedef uint32 CSSM_CSP_FLAGS;
```

2.4.9 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 KeyWorks to attach to a CSP. It is freed when an application causes KeyWorks 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 (see Table 5).

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

Table 5. CSSM Sessions

CSSM_CSP_Session Values		**Description**
CSSM_CSP_SESSION_EXCLUSIVE	0x0001	Single user CSP.
CSSM_CSP_SESSION_READWRITE	0x0002	Caller can read and write objects such as keys in the CSP.
CSSM_CSP_SESSION_SERIAL	0x0004	Multiuser, reentrant CSP that requires serial access.

2.4.10 CSSM_CSPSUBSERVICE

Three structures are used to contain all of the static information that describes a CSP module: the cssm_moduleinfo, cssm_serviceinfo, and cssm_cspsubservice structure. This descriptive information is securely stored in the KeyWorks registry when the CSP module is installed with CSSM. A CSP module may implement multiple types of services and organize them as subservices.

The descriptive information stored in these structures can be queried using the function CSSM_GetModuleInfo and specifying the cryptographic service provider module GUID.

```
typedef struct cssm_cspsubservice {
    uint32 SubServiceId;
    CSSM_STRING Description;
    CSSM_CSP_FLAGS CspFlags;
    uint32 CspCustomFlags;
    uint32 AccessFlags;
    CSSM_CSPTYPE CspType;
    union {
        CSSM_SOFTWARE_CSPSUBSERVICE_INFO SoftwareCspSubService;
        CSSM_HARDWARE_CSPSUBSERVICE_INFO HardwareCspSubService;
    };
    CSSM_CSP_WRAPPEDPRODUCT_INFO WrappedProduct;
} CSSM_CSPSUBSERVICE, *CSSM_CSPSUBSERVICE_PTR;
```

Definitions:

SubServiceId - The subservice ID required for an attach call to connect a CSP to an individual subservice within a CSP.

Description - A NULL-terminated character string containing a text description of the subservice.

CspFlags - A bit-mask containing general flags defined by KeyWorks for CSPs. The mask may contain a combination of the following in Table 6.

Table 6. CSP Flags

CSSM_CSP_FLAGS Values	Description
CSSM_CSP_STORES_PRIVATE_KEYS	CSP can store private keys.
CSSM_CSP_STORES_PUBLIC_KEYS	CSP can store public keys.
CSSM_CSP_STORES_SESSION_KEYS	CSP can store session/secret keys.

CspCustomFlags - Flags defined by the vendor. Consult the individual CSP documentation for the list of valid flags.

AccessFlags - Flags that are required to be provided by the application during an attach call when specifying the subservice ID given in *SubServiceId*.

CspType - Identifier that determines the type of CSP information structure referenced by *CspInfo*. The following values in Table 7 and their corresponding CSP information structures are currently defined.

Table 7. CSP Information Type Identifiers and Associated Structure Types

CSP Information Structure Identifier	Structure Type
CSSM_CSP_TYPE_SOFTWARE	CSSM_CSP_TYPE_SOFTWARE_INFO
CSSM_CSP_TYPE_PKCS11	CSSM_CSP_TYPE_PKCS11_INFO

SoftwareCspSubService/HardwareCspSubService - A CSP information structure of the type specified by *CspType*.

WrappedProduct - Pointer to a CSSM_CSP_WRAPPEDPRODUCTINFO structure describing a product that is wrapped by the CSP.

2.4.11 CSSM_CSPTYPE

```
typedef uint32 CSSM_CSPTYPE;
#define CSSM_CSP_SOFTWARE         1
#define CSSM_CSP_HARDWARE         2
```

2.4.12 CSSM_CSP_WRAPPEDPRODUCTINFO

```
typedef struct cssm_csp_wrappedproductinfo {
    CSSM_VERSION StandardVersion;
    CSSM_STRING StandardDescription;
    CSSM_VERSION ProductVersion;
    CSSM_STRING ProductDescription;
    CSSM_STRING ProductVendor;
    uint32 ProductFlags;
} CSSM_CSP_WRAPPEDPRODUCT_INFO, *CSSM_CSP_WRAPPEDPRODUCT_INFO_PTR;
```

Definitions:

StandardVersion - Version of the standard to which the wrapped product complies.

StandardDescription - A NULL-terminated character string containing a text description of the standard to which the wrapped product complies.

ProductVersion - Version of the product wrapped by the CSP.

ProductDescription - A NULL-terminated character string containing a text description of the product wrapped by the CSP.

ProductVendor - A NULL-terminated character string containing the name of the wrapped product's vendor.

ProductFlags - This version of KeyWorks has no flags defined. This field must be set to zero.

2.4.13 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 KeyWorks.

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

Definitions:

Length - Length of the data buffer in bytes.

Data - Pointer to a data buffer.

2.4.14 CSSM_DATE

```
typedef struct cssm_date {
    uint8 Year[4];
    uint8 Month[2];
    uint8 Day[2];
} CSSM_DATE, *CSSM_DATE_PTR;
```

Definitions:

Year - Four-digit ASCII representation of the year.

Month - Two-digit representation of the month.

Day - Two-digit representation of the day.

2.4.15 CSSM_HARDWARECSPSUBSERVICEINFO

```
typedef struct cssm_hardwarecspsubserviceinfo {
    uint32 NumberOfCapabilities;
    CSSM_CSP_CAPABILITY_PTR CapabilityList;
    void * Reserved;

    /* Reader/Slot Info */
    CSSM_STRING ReaderDescription;
    CSSM_STRING ReaderVendor;
    CSSM_STRING ReaderSerialNumber;
    CSSM_VERSION ReaderHardwareVersion;
    CSSM_VERSION ReaderFirmwareVersion;
    uint32 ReaderFlags;
    uint32 ReaderCustomFlags;

    CSSM_STRING TokenDescription;
    CSSM_STRING TokenVendor;
    CSSM_STRING TokenSerialNumber;
    CSSM_VERSION TokenHardwareVersion;
    CSSM_VERSION TokenFirmwareVersion;

    uint32 TokenFlags;
    uint32 TokenCustomFlags;
    uint32 TokenMaxSessionCount;
    uint32 TokenOpenedSessionCount;
    uint32 TokenMaxRWSessionCount;
    uint32 TokenOpenedRWSessionCount;
    uint32 TokenTotalPublicMem;
    uint32 TokenFreePublicMem;
    uint32 TokenTotalPrivateMem;
    uint32 TokenFreePrivateMem;
    uint32 TokenMaxPinLen;
    uint32 TokenMinPinLen;
    char TokenUTCTime[16];

    char *UserLabel;
    CSSM_DATA UserCACertificate;
} CSSM_HARDWARE_CSPSUBSERVICE_INFO, *CSSM_HARDWARE_CSPSUBSERVICE_INFO_PTR;
```

Definitions:

NumberOfCapabilities - Number of capabilities in list.

CapabilityList - A context list that specifies the capabilities of the CSP.

Reserved - This field is reserved for future use and must always be set to NULL.

ReaderDescription - A NULL-terminated character string that contains a text description of the device reader.

ReaderVendor - A NULL-terminated string that contains the name of the reader vendor.

ReaderSerialNumber - A NULL-terminated string that contains the serial number of the reader.

ReaderHardwareVersion - Hardware version of the reader.

ReaderFirmwareVersion - Firmware version of the reader.

ReaderFlags - Bit-mask containing information about the reader. The flags specified in the mask are as follows in Table 8.

Table 8. PKCS#11 CSP Reader Flags

Reader Flag	Description
CSSM_CSP_RDR_TOKENPRESENT	Token is present in the reader.
CSSM_CSP_RDR_REMOVABLE	Reader supports removable tokens.
CSSM_CSP_RDR_HW	Reader is a hardware device.

ReaderCustomFlags - Flags defined by the vendor. Consult the individual CSP documentation for the list of valid flags.

The following fields may not be valid if the CSSM_CSP_RDR_TOKENPRESENT flag is not set in the *ReaderFlags* field. Unknown string and CSSM_DATA fields will be set to NULL, integer and date fields will be set to zero, and flag fields will have all flags set to false.

TokenDescription - A NULL-terminated character string that contains a text description of the token. This value may be NULL or equal to *ReaderDescription* if the token is not removable.

TokenVendor - A NULL-terminated string that contains the name of the token vendor. This value may be NULL or equal to *ReaderVendor* if the token is not removable.

TokenSerialNumber - A NULL-terminated string that contains the serial number of the token. This value may be NULL or equal to *ReaderSerialNumber* if the token is not removable.

TokenHardwareVersion - Hardware version of the token.

TokenFirmwareVersion - Firmware version of the token.

TokenFlags - Bit-mask containing information about the token. The flags specified in the mask are as follows in Table 9.

Table 9. PKCS#11 CSP Token Flags

Token Flags	Description
CSSM_CSP_TOK_RNG	Token has random number generator.
CSSM_CSP_TOK_WRITE_PROTECTED	Token is write-protected.
CSSM_CSP_TOK_LOGIN_REQUIRED	User must login to access private objects.
CSSM_CSP_TOK_USER_PIN_INITIALIZED	User's PIN has been initialized.
CSSM_CSP_TOK_EXCLUSIVE_SESSION	An exclusive session currently exists.
CSSM_CSP_TOK_CLOCK_EXISTS	Token has built-in clock.
CSSM_CSP_TOK_ASYNC_SESSION	Token supports asynchronous operations.
CSSM_CSP_TOK_PROT_AUTHENTICATION	Token has protected authentication path.
CSSM_CSP_TOK_DUAL_CRYPTO_OPS	Token supports dual cryptographic operations.

TokenCustomFlags - Flags defined by the vendor. Consult the individual CSP documentation for the list of valid flags.

TokenMaxSessionCount - Maximum number of CSP handles referencing the token that may exist simultaneously.

TokenOpenedSessionCount - Number of CSP handles referencing the token that currently exists.

TokenTotalPublicMem - Amount of public storage space in the CSP. This value will be set to CSSM_VALUE_NOT_AVAILABLE if the CSP does not want to expose this information.

TokenFreePublicMem - Amount of public storage space available for use in the CSP. This value will be set to CSSM_VALUE_NOT_AVAILABLE (-1) if the CSP does not want to expose this information.

TokenTotalPrivateMem - Amount of private storage space in the CSP. This value will be set to CSSM_VALUE_NOT_AVAILABLE (-1) if the CSP does not want to expose this information.

TokenFreePrivateMem - Amount of private storage space available for use in the CSP. This value will be set to CSSM_VALUE_NOT_AVAILABLE if the CSP does not want to expose this information.
TokenMaxPinLen - Maximum length of passwords that can be used for authentication to the CSP.

TokenMinPinLen - Minimum length of passwords that can be used for authentication to the CSP.

TokenUTCTime - Character array containing the current Coordinated Universal Time (UTC) value in the CSP. The value is valid if the CSSM_CSP_TOK_CLOCK_EXISTS flag is true. The time is represented in the format YYYYMMDDhhmmssxx (4 characters for the year; 2 characters each for the month, day, hour, minute, and second; and 2 additional reserved '0'characters).

UserLabel - A NULL-terminated string containing the label of the token.

UserCACertificate - Certificate of the Certificate Authority (CA).

2.4.16 CSSM_HEADERVERSION

This data structure represents the version number of a key header structure. This version number is an integer that increments with each format revision of CSSM_KEYHEADER. The current revision number is represented by CSSM_KEYHEADER_VERSION, which equals 2 in this release of KeyWorks.

```
typedef uint32 CSSM_HEADERVERSION

#define CSSM_KEYHEADER_VERSION (2)
```

2.4.17 CSSM_KEY

This structure is used to represent keys in KeyWorks.

```
typedef struct cssm_key{
    CSSM_KEYHEADER KeyHeader;
    CSSM_DATA KeyData;
} CSSM_KEY, *CSSM_KEY_PTR;

typedef CSSM_KEY CSSM_WRAP_KEY, *CSSM_WRAP_KEY_PTR;
```

Definitions:

KeyHeader - Header describing the key, fixed length.

KeyData – Data representation of the key, variable length.

2.4.18 CSSM_KEYHEADER

The key header contains meta-data about a key. It contains information used by a CSP or application when using the associated key data. The service provider module is responsible for setting the appropriate values.

```
typedef struct cssm_keyheader {
    CSSM_HEADERVERSION HeaderVersion;
    CSSM_GUID CspId;
    uint32 BlobType;
    uint32 Format;
    uint32 AlgorithmId;
    uint32 KeyClass;
    uint32 KeySizeInBits;
    uint32 KeyAttr;
    uint32 KeyUsage;
    CSSM_DATE StartDate;
    CSSM_DATE EndDate;
    uint32 WrapAlgorithmId;
    uint32 WrapMode;
    uint32 Reserved;
} CSSM_KEYHEADER, *CSSM_KEYHEADER_PTR;
```

Definitions:

HeaderVersion - This is the version of the key header structure.

CspId - If known, the Globally Unique ID (GUID) of the CSP that generated the key. This value will not be known if a key is received from a third party or extracted from a certificate.

BlobType - Describes the basic format of the key data. It can be any one of the following values in Table 10.

Table 10. Key Blob Type Identifiers

Key Blob Type Identifier	Description
CSSM_KEYBLOB_RAW	The blob is a clear, raw key.
CSSM_KEYBLOB_RAW_BERDER	The blob is a clear key, DER encoded.
CSSM_KEYBLOB_REFERENCE	The blob is a reference to a key.
CSSM_KEYBLOB_WRAPPED	The blob is a wrapped RAW key.
CSSM_KEYBLOB_WRAPPED_BERDER	The blob is a wrapped DER-encoded key.
CSSM_KEYBLOB_OTHER	The blob is a wrapped DER-encoded key.

Format - Describes the detailed format of the key data based on the value of the *BlobType* field. If the blob type has a nonreference basic type, then a CSSM_KEYBLOB_RAW_FORMAT identifier must be used, otherwise a CSSM_KEYBLOB_REF_FORMAT identifier is used. Any of the following values in Table 11 are valid as format identifiers.

Table 11. Key Blob Format Identifiers

Key Blob Format Identifier	Description
CSSM_KEYBLOB_RAW_FORMAT_NONE	No further conversion needs to be done.
CSSM_KEYBLOB_RAW_FORMAT_PKCS1	RSA PKCS1 V1.5
CSSM_KEYBLOB_RAW_FORMAT_PKCS3	RSA PKCS3 V1.5
CSSM_KEYBLOB_RAW_FORMAT_MSCAPI	Microsoft CAPI V2.0
CSSM_KEYBLOB_RAW_FORMAT_PGP	PGP
CSSM_KEYBLOB_RAW_FORMAT_FIPS186	U.S. Gov. FIPS 186 - DSS V
CSSM_KEYBLOB_RAW_FORMAT_BSAFE	RSA BSAFE V3.0
CSSM_KEYBLOB_RAW_FORMAT_PKCS11	RSA PKCS11 V2.0
CSSM_KEYBLOB_RAW_FORMAT_CDSA	Intel CDSA
CSSM_KEYBLOB_RAW_FORMAT_OTHER	Other, CSP defined
CSSM_KEYBLOB_REF_FORMAT_INTEGER	Reference is a number or handle.
CSSM_KEYBLOB_REF_FORMAT_STRING	Reference is a string or name.
CSSM_KEYBLOB_REF_FORMAT_OTHER	Other, CSP defined

AlgorithmId - The algorithm for which the key was generated. This value does not change when the key is wrapped. Any of the defined KeyWorks algorithm IDs may be used.

KeyClass - Class of key contained in the key blob. Valid key classes are as follows in Table 12.

Table 12. Key Class Identifiers

Key Class Identifier	Description
CSSM_KEYCLASS_PUBLIC_KEY	Key is a public key.
CSSM_KEYCLASS_PRIVATE_KEY	Key is a private key.
CSSM_KEYCLASS_SESSION_KEY	Key is a session or symmetric key.
CSSM_KEYCLASS_SECRET_PART	Key is part of secret key.
CSSM_KEYCLASS_OTHER	Other.

KeySizeInBits - This is the logical size of the key in bits. The logical size is the value referred to when describing the length of the key. For instance, an RSA key would be described by the size of its modulus and a DSA key would be represented by the size of its prime. Symmetric key sizes describe the actual number of bits in the key. For example, Data Encryption Standard (DES) keys would be 64 bits and an RC4 key could range from 1 to 128 bits.

KeyAttr - Attributes of the key represented by the data. These attributes are used by CSPs to convey information about stored or referenced keys. The attributes are represented as a bit-mask (see Table 13).

Table 13. Key Attribute Flags

Attribute	Description
CSSM_KEYATTR_PERMANENT	Key is stored persistently in the CSP, i.e., PKCS11 token object.
CSSM_KEYATTR_PRIVATE	Key is a private object and protected by either user login, a password, or both.
CSSM_KEYATTR_MODIFIABLE	Key or its attributes can be modified.
CSSM_KEYATTR_SENSITIVE	Key is sensitive. It may only be extracted from the CSP in a wrapped state. It will always be false for raw keys.
CSSM_KEYATTR_ALWAYS_SENSITIVE	Key has always been sensitive. It will always be false for raw keys.
CSSM_KEYATTR_EXTRACTABLE	Key is extractable from the CSP. If this bit is not set, the key is either not stored in the CSP or cannot be extracted from the CSP under any circumstances. It will always be false for raw keys.
CSSM_KEYATTR_NEVER_EXTRACTABLE	Key has never been extractable. It will always be false for raw keys.

KeyUsage - A bit-mask representing the valid uses of the key. Any of the following values are valid in Table 14.

Table 14. Key Usage Flags

Usage Mask	Description
CSSM_KEYUSE_ANY	Key may be used for any purpose supported by the algorithm.
CSSM_KEYUSE_ENCRYPT	Key may be used for encryption.
CSSM_KEYUSE_DECRYPT	Key may be used for decryption.
CSSM_KEYUSE_SIGN	Key can be used to generate signatures. For symmetric keys, this represents the ability to generate MACs.
CSSM_KEYUSE_VERIFY	Key can be used to verify signatures. For symmetric keys, this represents the ability to verify MACs.
CSSM_KEYUSE_SIGN_RECOVER	Key can be used to perform signatures with message recovery. This form of a signature is generated using the CSSM_EncryptData API with the algorithm mode set to CSSM_ALGMODE_PRIVATE_KEY. This attribute is only valid for asymmetric algorithms.
CSSM_KEYUSE_VERIFY_RECOVER	Key can be used to verify signatures with message recovery. This form of a signature verified using the CSSM_DecryptData API with the algorithm mode set to CSSM_ALGMODE_PRIVATE_KEY. This attribute is only valid for asymmetric algorithms.
CSSM_KEYUSE_WRAP	Key can be used to wrap another key.
CSSM_KEYUSE_UNWRAP	Key can be used to unwrap a key.
CSSM_KEYUSE_DERIVE	Key can be used as the source for deriving other keys.

StartDate - Date from which the corresponding key is valid. All fields of the CSSM_DATA structure will be set to zero if the date is unspecified or unknown. This date is not enforced by the CSP.

EndDate - Data that the key expires and can no longer be used. All fields of the CSSM_DATA structure will be set to zero if the date is unspecified or unknown. This date is not enforced by the CSP.

WrapAlgorithmId - If the key data contains a wrapped key, this field contains the algorithm used to create the wrapped blob. This field will be set to CSSM_ALGID_NONE if the key is not wrapped.

WrapMode - If the wrapping algorithm supports multiple wrapping modes, this field contains the mode used to wrap the key. This field is ignored if the *WrapAlgorithmId* is CSSM_ALGID_NONE.

Reserved - This field is reserved for future use. It should always be set to zero.

2.4.19 CSSM_KEY_SIZE

This structure holds the key size and the effective key size for a given key. The metric used is bits. The number of effective bits is the number of key bits that can be used in a cryptographic operation compared with the number of bits that may be present in the key. When the number of effective bits is less than the number of actual bits, this is known as *dumbing down*.

```
typedef struct cssm_key_size {
    uint32 KeySizeInBits; /* Key size in bits */
    uint32 EffectiveKeySizeInBits; /* Effective key size in bits */
} CSSM_KEYZIZE, *CSSM_KEYSIZE_PTR;
```

Definitions:

KeySizeInBits - The actual number of bits in a key.

EffectiveKeySizeInBits - The number of key bits that can be used for cryptographic operations.

2.4.20 CSSM_KEY_TYPE

```
typedef uint32 CSSM_KEY_TYPE, *CSSM_KEY_TYPE_PTR;
```

2.4.21 CSSM_NOTIFY_CALLBACK

This data structure defines a pointer to a function that applications can use to invoke an application-supplied function.

```
typedef CSSM_RETURN (CSSMAPI *CSSM_NOTIFY_CALLBACK)
    (CSSM_MODULE_HANDLE ModuleHandle,
        uint32 Application,
        uint32 Reason,
        void * Param)
```

Definitions:

ModuleHandle - Handle of the module to which the notification applies.

Application - Application-specific context indicator. This value is specified when a service provider module is attached.

Reason - One of the values specified below in Table 15.

Table 15. CSSM_NOTIFY Reason Values

Reason	Value
CSSM_NOTIFY_SURRENDER	0
CSSM_NOTIFY_COMPLETE	1
CSSM_NOTIFY_DEVICE_REMOVED	2
CSSM_NOTIFY_DEVICE_INSERTED	3

Param - Used by the module that triggers the notification to pass relevant information about the notification to the application.

2.4.22 CSSM_PADDING

```
typedef enum cssm_padding {
    CSSM_PADDING_NONE           = 0,
    CSSM_PADDING_CUSTOM         = CSSM_PADDING_NONE+1,
    CSSM_PADDING_ZERO           = CSSM_PADDING_NONE+2,
    CSSM_PADDING_ONE            = CSSM_PADDING_NONE+3,
    CSSM_PADDING_ALTERNATE      = CSSM_PADDING_NONE+4,
    CSSM_PADDING_FF             = CSSM_PADDING_NONE+5,
    CSSM_PADDING_PKCS5          = CSSM_PADDING_NONE+6,
    CSSM_PADDING_PKCS7          = CSSM_PADDING_NONE+7,
    CSSM_PADDING_CipherStealing = CSSM_PADDING_NONE+8,
    CSSM_PADDING_RANDOM         = CSSM_PADDING_NONE+9
} CSSM_PADDING;
```

2.4.23 CSSM_QUERY_SIZE_DATA

```
typedef struct cssm_query_size_data {
    uint32 SizeInputBlock;
    uint32 SizeOutputBlock;
} CSSM_QUERY_SIZE_DATA, *CSSM_QUERY_SIZE_DATA_PTR
```

Definitions:

SizeInputBlock - The size of the input block in bytes.

SizeOutputBlock - The size of the output block in bytes.

2.4.24 CSSM_RANGE

```
typedef struct cssm_range {
uint32 Min;/* inclusive minimium value */
uint32 Max;/* inclusive maximium value */
} CSSM_RANGE, *CSSM_RANGE_PTR
```

Definitions:

Min - Minimum value in the range.

Max - Maximum value in the range.

2.4.25 CSSM_SOFTWARECSPSUBSERVICEINFO

```
typedef struct cssm_softwarecspsubserviceinfo {
    uint32 NumberOfCapabilities;
    CSSM_CSP_CAPABILITY_PTR CapabilityList;
    void* Reserved;
} CSSM_SOFTWARE_CSPSUBSERVICE_INFO, *CSSM_SOFTWARE_CSPSUBSERVICE_INFO_PTR;
```

Definitions:

NumberOfCapabilities - Number of capabilities available from the CSP.

CapabilityList - Pointer to an array of CSSM_CSP_CAPABILITY structures that represent the capabilities available from the CSP.

Reserved - Reserved for future use.

2.4.26 CSSM_SPI_FUNC_TBL

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

```
typedef struct cssm_spi_func_tbl {
    void *(*malloc_func) (CSSM_HANDLE AddInHandle, uint32 Size);
    void (*free_func) (CSSM_HANDLE AddInHandle, void *MemPtr);
    void  *(*realloc_func) (CSSM_HANDLE AddInHandle, void *MemPtr, uint32 Size);
    void  *(*calloc_func) (CSSM_HANDLE AddInHandle, uint32 Num, uint32 Size);
} CSSM_SPI_MEMORY_FUNCS, *CSSM_SPI_MEMORY_FUNCS_PTR;
```

2.5 Cryptographic Operations

2.5.1 CSP_DecryptData

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Comments

The output can be obtained by either filling the caller-supplied buffer or using the application's memory allocation functions to allocate space. The 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

2.5.2 CSP_DecryptDataFinal

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Comments

The output can be obtained by either filling the caller-supplied buffer or using the application's memory allocation functions to allocate space. The 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

2.5.3 CSP_DecryptDataInit

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

See Also

CSP_DecryptData, CSP_DecryptDataUpdate, CSP_DecryptDataFinal

2.5.4 CSP_DecryptDataUpdate

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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 by either filling the caller-supplied buffer or using the application's memory allocation functions to allocate spaces. The 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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Comments

The output can be obtained by either filling the caller-supplied buffer or using the application's memory allocation functions to allocate space. The 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

2.5.5 CSP_DeriveKey

CSSM_RETURN CSP_DeriveKey (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
const CSSM_KEY_PTR BaseKey,
void * Param,
uint32 KeyUsage,
uint32 KeyAttr,
const CSSM_DATA_PTR KeyLabel,
CSSM_KEY_PTR DerivedKey)

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

Parameters

CSPHandle (input)
The handle that describes the add-in CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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 initialization vector. Concatenation algorithms will use this parameter to get the second item to concatenate.

KeyUsage (input)
A bit-mask representing the valid uses of the key.

KeyAttr (input)
A bit-mask representing the attributes of the key represented by the data.

KeyLabel (input/optional)
Pointer to a byte string that will be used as the label for the derived key.

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

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

2.5.6 CSP_DigestData

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Comments

The output can be obtained by either filling the caller-supplied buffer or using the application's memory allocation functions to allocate space. The 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

2.5.7 CSP_DigestDataClone

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

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

2.5.8 CSP_DigestDataFinal

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Comments

The output can be obtained by either filling the caller-supplied buffer or using the application's memory allocation functions to allocate space. The 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

2.5.9 CSP_DigestDataInit

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

See Also

CSP_DigestData, CSP_DigestDataUpdate, CSP_DigestDataClone, CSP_DigestDataFinal

2.5.10 CSP_DigestDataUpdate

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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 KeyWorks return value. This function returns CSSM_OK if successful and returns an error code if an error has occurred.

See Also

CSP_DigestData, CSP_DigestDataInit, CSP_DigestDataClone, CSP_DigestDataFinal

2.5.11 CSP_EncryptData

CSSM_RETURN 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. The CSP_QuerySize function can be used to estimate the output buffer size required.

Parameters

CSPHandle (input)
The handle that describes the add-in CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Comments

The output can be obtained by either filling the caller-supplied buffer or using the application's memory allocation functions to allocate space. The 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

2.5.12 CSP_EncryptDataFinal

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Comments

The output can be obtained by either filling the caller-supplied buffer or using the application's memory allocation functions to allocate space. The 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

2.5.13 CSP_EncryptDataInit

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

See Also

CSP_EncryptData, CSP_EncryptDataUpdate, CSP_EncryptDataFinal

2.5.14 CSP_EncryptDataUpdate

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Comments

The output can be obtained by either filling the caller-supplied buffer or using the application's memory allocation functions to allocate space. The 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

2.5.15 CSP_GenerateAlgorithmParams

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

This function generates algorithm parameters for the specified context. These parameters include Diffie-Hellman key agreement parameters and DSA key generation parameters.

Parameters

CSPHandle (input)
The handle that describes the add-in CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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 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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Comments

The output can be obtained by either filling the caller-supplied buffer or using the application's memory allocation functions to allocate space. The 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.

2.5.16 CSP_GenerateKey

CSSM_RETURN CSP_GenerateKey (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
uint32 KeyUsage,
uint32 KeyAttr,
const CSSM_DATA_PTR KeyLabel,
CSSM_KEY_PTR Key)

This function generates a symmetric key.

Parameters

CSPHandle (input)
The handle that describes the add-in CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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.

KeyUsage (input)
A bit-mask representing the valid uses of the key.

KeyAttr (input)
A bit-mask representing the attributes of the key represented by the data.

KeyLabel (input)
Pointer to a byte string that will be used as the label for the key.

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

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Comments

The output can be obtained by either filling the caller-supplied buffer or using the application's memory allocation functions to allocate space. The 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, CSSM_GenerateKeyPair

2.5.17 CSP_GenerateKeyPair

CSSM_RETURN CSP_GenerateKeyPair (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
uint32 PublicKeyUsage,
uint32 PublicKeyAttr,
const CSSM_DATA_PTR PublicKeyLabel,
CSSM_KEY_PTR PublicKey,
uint32 PrivateKeyUsage,
uint32 PrivateKeyAttr,
const CSSM_DATA_PTR PrivateKeyLabel,
CSSM_KEY_PTR PrivateKey)

This function generates an asymmetric key pair.

Parameters

CSPHandle (input)
The handle that describes the add-in CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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.

PublicKeyUsage (input)
A bit-mask representing the valid uses of the public key.

PublicKeyAttr (input)
A bit-mask representing the attributes of the public key represented by the data. These attributes can be used to convey information about stored or referenced keys.

PublicKeyLabel(input)
Pointer to a byte string that will be used as the label for the public key.

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

PrivateKeyUsage (input)
A bit-mask representing the valid uses of the private key.

PrivateKeyAttr (input)
A bit-mask representing the attributes of the private key represented by the data. These attributes can be used to convey information about stored or referenced keys.

PrivateKeyLabel(input)
Pointer to a byte string that will be used as the label for the private key.

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

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Comments

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

See Also

CSSM_GenerateRandom, CSSM_GenerateKey

2.5.18 CSP_GenerateMac

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Comments

The output can be obtained by either filling the caller-supplied buffer or using the application's memory allocation functions to allocate space. The 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

2.5.19 CSP_GenerateMacFinal

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Comments

The output can be obtained by either filling the caller-supplied buffer or using the application's memory allocation functions to allocate space. The 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

2.5.20 CSP_GenerateMacInit

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

See Also

CSP_GenerateMac, CSP_GenerateMacUpdate, CSP_GenerateMacFinal

2.5.21 CSP_GenerateMacUpdate

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

See Also

CSP_GenerateMac, CSP_GenerateMacInit, CSP_GenerateMacFinal

2.5.22 CSP_GenerateRandom

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Comments

The output can be obtained by either filling the caller-supplied buffer or using the application's memory allocation functions to allocate space. The 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.

2.5.23 CSP_QueryKeySizeInBits

CSSM_RETURN CSP_QueryKeySizeInBits (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
CSSM_KEY_SIZE_PTR KeySize)

This function queries a CSP for the effective and real size of a key in bits.

Parameters

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

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

KeySize (output)
Pointer to a CSSM_KEYSIZE data structure to receive the size of the key in bits.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

2.5.24 CSP_QuerySize

CSSM_RETURN CSP_QuerySize (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
CSSM_BOOL Encrypt,
uint32 QuerySizeCount,
CSSM_QUERY_SIZE_DATA_PTR DataBlock)

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. For encryption, the total size of all output buffers must always be a multiple of the output block size. This function also can 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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.

Encrypt (input)
This parameter describes whether the SizeInputBlock in DataBlock is for encryption (CSSM_TRUE) or decryption (CSSM_FALSE).

QuerySizeCount (input)
This parameter describes the number of DataBlocks.

DataBlock (input/output)
Pointer to a CSSM_QUERY_SIZE_DATA structure which contains one SizeInputBlock and one SizeOutputBlock. The function returns the size of the output in bytes in SizeOutputBlock for the size of the input.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

See Also

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

2.5.25 CSP_SignData

CSSM_RETURN 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 associated with the public key specified in the context.

Parameters

CSPHandle (input)
The handle that describes the add-in CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Comments

The output can be obtained by either filling the caller-supplied buffer or using the application's memory allocation functions to allocate space. The 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

2.5.26 CSP_SignDataFinal

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Comments

The output can be obtained by either filling the caller-supplied buffer or using the application's memory allocation functions to allocate space. The 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

2.5.27 CSP_SignDataInit

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

See Also

CSP_SignData, CSP_SignDataUpdate, CSP_SignDataFinal

2.5.28 CSP_SignDataUpdate

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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 to be signed.

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

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

See Also

CSP_SignData, CSP_SignDataInit, CSP_SignDataFinal

2.5.29 CSP_UnwrapKey

CSSM_RETURN 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,
uint32 StorageMask,
const CSSM_DATA_PTR KeyLabel,
CSSM_KEY_PTR UnwrappedKey)

This function unwraps the data using the context.

Parameters

CSPHandle (input)
The handle that describes the add-in CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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.

NewPassPhrase (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 NewPassPhrase 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.

StorageMask (input)
A storage mask that is used by the CSP to determine how to store the unwrapped key and how to return that key to the application.

KeyLabel (input/optional)
Pointer to a byte string that will be used as the label for the unwrapped key.

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

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

See Also

CSP_WrapKey

2.5.30 CSP_VerifyData

CSSM_BOOL 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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 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.

See Also

CSP_SignData, CSP_VerifyDataInit, CSP_VerifyDataUpdate, CSP_VerifyDataFinal

2.5.31 CSP_VerifyDataFinal

CSSM_BOOL CSP_VerifyDataFinal (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle
const CSSM_DATA_PTR Signature)

This function finalizes the staged verify data function.

Parameters

CSPHandle (input)
The handle that describes the add-in CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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

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

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. The use CSSM_GetError to obtain the error code.

See Also

CSP_VerifyData, CSP_VerifyDataInit, CSP_VerifyDataUpdate

2.5.32 CSP_VerifyDataInit

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

This function initializes the staged verify data function.

Parameters

CSPHandle (input)
The handle that describes the add-in CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

See Also

CSP_VerifyDataUpdate, CSP_VerifyDataFinal, CSP_VerifyData

2.5.33 CSP_VerifyDataUpdate

CSSM_RETURN 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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 to be verified.

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

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

See Also

CSP_VerifyData, CSP_VerifyDataInit, CSP_VerifyDataFinal

2.5.34 CSP_VerifyMac

CSSM_RETURN CSP_VerifyMac (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 verifies a message authentication code for the supplied data.

Parameters

CSPHandle (input)
The handle that describes the add-in CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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 a vector of CSSM_DATA structures that contain the data to be operated on.

DataBufCount (input)
The number of DataBufs.

Mac (input)
A pointer to the CSSM_DATA structure containing the MAC to verify.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

See Also

CSSM_VerifyMacInit, CSSM_VerifyMacUpdate, CSSM_VerifyMacFinal

2.5.35 CSP_VerifyMacFinal

CSSM_RETURN CSP_VerifyMacFinal (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR Mac)

This function finalizes the staged message authentication code verification function.

Parameters

CSPHandle (input)
The handle that describes the add-in CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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

Mac (input)
A pointer to the CSSM_DATA structure containing the MAC to verify.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

See Also

CSSM_VerifyMac, CSSM_VerifyMacInit, CSSM_VerifyMacUpdate

2.5.36 CSP_VerifyMacInit

CSSM_RETURN CSP_VerifyMacInit (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context)

This function initializes the staged message authentication code verification function.

Parameters

CSPHandle (input)
The handle that describes the add-in CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

See Also

CSSM_VerifyMac, CSSM_VerifyMacUpdate, CSSM_VerifyMacFinal

2.5.37 CSP_VerifyMacUpdate

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

This function updates the staged message authentication code verification function.

Parameters

CSPHandle (input)
The handle that describes the add-in CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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

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

DataBufCount (input)
The number of DataBufs.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

See Also

CSSM_VerifyMac, CSSM_VerifyMacInit, CSSM_VerifyMacFinal

2.5.38 CSP_WrapKey

CSSM_RETURN CSP_WrapKey (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
const CSSM_CRYPTO_DATA_PTR PassPhrase,
const 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 CSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

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_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

See Also

CSP_UnwrapKey

2.6 Cryptographic Sessions and Logon

2.6.1 CSP_ChangeLoginPassword

CSSM_RETURN 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 if login fails. Use CSSM_GetError to determine the exact error.

See Also

CSP_Login, CSP_Logout

2.6.2 CSP_Login

CSSM_RETURN CSP_Login (CSSM_CSP_HANDLE CSPHandle,
const CSSM_CRYPTO_DATA_PTR Password,
const CSSM_DATA_PTR Reserved)

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

Parameters

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

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

Reserved (input)
This field is reserved for future use. The value NULL should always be given.

Return Value

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

See Also

CSP_ChangeLoginPassword, CSP_Logout

2.6.3 CSP_Logout

CSSM_RETURN 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 occurred. Use CSSM_GetError to determine the exact error.

See Also

CSP_Login, CSP_ChangePassword

2.7 Extensibility Functions

The CSP_PassThrough function is provided to allow CSP developers to extend the cryptographic functionality of the KeyWorks API. Because it is only exposed to KeyWorks 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 in Section 2.7.1.

2.7.1 CSP_PassThrough

void * CSP_PassThrough (CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT_PTR Context,
uint32 PassThroughId,
const void * InData)

Parameters

CSPHandle (input)
The handle that describes the add-in CSP module used to perform this function.

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 a module, implementation-specific structure containing parameters to be interpreted in a function-specific manner by the requested CSP module. This parameter can be used as a pointer to an array of void pointers.

Return Value

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

Chapter 3. Cryptographic Service Provider Function Examples

3.1 Attach/Detach Example

The CSP module is responsible for performing certain operations when KeyWorks attaches to and detaches from it. These operations should be performed in a function called AddInAuthenticate, which must be exported by the CSP module. The AddInAuthenticate function will be called by the framework when the module is loaded. The steps shown in Section 3.1.1 must be performed in order for the attach process to work properly.

In the code example in Section 3.1.1, it is assumed that the CSSM entry points, such as CSSM_RegisterServices, have been resolved at link time. If not, the module may call GetProcAddress to resolve the entry points. Also, this AddInAuthenticate indicates a CSP module which implements only the DecryptData and EncryptData functions. The unimplemented functions in the function table are initialized to NULL, and not reassigned.

3.1.1 AddInAuthenticate

```
#include “cssm.h”

CSSM_SPI_MEMORY_FUNCS      CssmMemFuncs;
CSSM_GUID CspGuid =
{ 0x83badc39, 0xfac1, 0x11cf, { 0x81, 0x72, 0x0, 0xaa, 0x0, 0xb1, 0x99, 0xdd } };

CSSM_RETURN CSSMAPI AddInAuthenticate(char* cssmCredentialPath, char*
cssmSection)
{
       CSSM_SPI_CSP_FUNCS         CssmCSPFuncs;
       CSSM_REGISTRATION_INFO     CssmRegInfo;
       CSSM_MODULE_FUNCS          CssmModuleFuncs[1];
       CSSM_RETURN                retcode;

       // initialize tables
       memset(&CssmCSPFuncs, 0, sizeof(CSSM_SPI_CSP_FUNCS));
       memset(&CssmRegInfo, 0, sizeof(CSSM_REGISTRATION_INFO));

       // Now register services
       CssmCSPFuncs.DecryptData         = DecryptData;
       CssmCSPFuncs.EncryptData         = EncryptData;

       CssmRegInfo.Initialize           = Initialize;
       CssmRegInfo.Terminate            = Uninitialize;
       CssmRegInfo.EventNotify          = EventNotify;
       CssmRegInfo.ThreadSafe           = CSSM_TRUE;
       CssmRegInfo.ServiceSummary       = CSSM_SERVICE_CSP;
       CssmRegInfo.NumberOfServiceTables = 1;
       CssmRegInfo.Services             = CssmModuleFuncs;

       CssmModuleFuncs[0].ServiceType = CSSM_SERVICE_CSP;
       CssmModuleFuncs[0].CspFuncs = &CssmCSPFuncs;

       retcode = CSSM_RegisterServices(&CspGuid, &CssmRegInfo, &CssmMemFuncs,
       NULL);

       return retcode;
}
```

3.2 Extensibility Functions Examples

This section contains a sample implementation of the passthrough function in the CSP library.

3.2.1 CSP_PassThrough

Some CSP vendors may need to provide functionality that is not part of the KeyWorks API. These functions are called private functions. Applications access the CSP private functions by using the CSSM_PassThrough API. The following is an example CSP_PrivateFunctions function.

```
/* PassThrough IDs */
typedef enum csp_custom_function_id {
    CSP_CUSTOMID_CHANGE_PASSWORD = 0,
    CSP_CUSTOMID_IMPORT_PRIKEY = 1,
    CSP_CUSTOMID_EXPORT_PRIKEY = 2,
} CSP_CUSTOM_FUNCTION_ID;


/*---------------------------------------------------------------------------
* Name: CSP_PassThrough
*
* Description:
* This function allows applications to call KeyWorks CSP module-specific
operations.
* Examples of KeyWorks CSP module-specific operations include:
*     csp_ChangePassword
*     csp_ImportPrivateKey
*     csp_ExportPrivateKey
*
* Parameters:
* CSPHandle (input)     : The handle that describes the add-in CSP module used by
*                         the passthrough function.
* CCHandle (input)      : Handle identifying a Cryptographic Context which
*                         may be used by the passthrough function
* Context               : Pointer to CSSM_CONTEXT structure that describes
*                         the attributes associated with this context.
* PassThroughId (input) : An identifier assigned by the KeyWorks CSP module
*                         to indicate the exported function to perform.
* InData (input)        : Parameters to be interpreted in a
*                         function-specific manner by the KeyWorks CSP module.
*
* Return value:
* Output from the passthrough function.
* The output data must be interpreted by the calling application
* based on externally available information.
*
* Error Codes:
* CSSM_CSP_INVALID_CSP_HANDLE
* CSSM_CSP_INVALID_CC_HANDLE
* CSSM_CSP_INVALID_DATA_POINTER
* CSSM_CSP_INVALID_PASSTHROUGH_ID
* CSSM_CSP_INVALID_PASSTHROUGH_PARAMS
* CSSM_CSP_UNSUPPORTED_OPERATION
* CSSM_CSP_PASS_THROUGH_FAIL
*---------------------------------------------------------------------------*/
void * CSSMAPI CSP_PassThrough  (CSSM_CSP_HANDLE CSPHandle,
                                 CSSM_CC_HANDLE CCHandle,
                                 const CSSM_CONTEXT_PTR Context,
                                 uint32 PassThroughId,
                                 void * InData)
{
    /* Initializations */
    /* Check inputs */
    /* Check that this is a recognized PassThroughId */

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

```
    return csp_ChangePassword( InData );
case CSP_CUSTOMID_IMPORT_PRIKEY:
    return csp_ImportPrivateKey( InData );
case CSP_CUSTOMID_EXPORT_PRIKEY:
    return csp_ExportPrivateKey( InData );
default:
    CSSM_SetError(&my_csp_guid, CSSM_CSP_UNSUPPORTED_OPERATION);
return NULL;
}
return NULL;
};
```

Appendix A. IBM KeyWorks Errors

This section describes the error handling features in KeyWorks that provide a consistent mechanism across all layers of KeyWorks for returning errors to the caller. All Cryptographic Service Provider (CSP) service provider interface (SPI) functions return variables of the following types:

- CSSM_RETURN - An enumerated type consisting of CSSM_OK and CSSM_FAIL. If it is CSSM_FAIL, an error code indicating the reason for failure can be obtained by calling CSSM_GetError.

- CSSM_BOOL - KeyWorks functions returning this data type return either CSSM_TRUE or CSSM_FALSE. If the function returns CSSM_FALSE, an error code may be available (but not always) by calling CSSM_GetError.

- A pointer to a data structure, a handle, a file size, or whatever is logical for the function to return. An error code may be available (but not always) by calling CSSM_GetError.

The information returned from CSSM_GetError includes both the error number and a Globally Unique ID (GUID) that associates the error with the module that set it. Each module must have a mechanism for reporting their errors to the calling application. In general, there are two types of errors a module can return:

- Errors defined by KeyWorks that are common to a particular type of service provider module
- Errors reserved for use by individual service provider modules

Since some errors are predefined by KeyWorks, those errors have a set of predefined numeric values that are reserved by KeyWorks, and cannot be redefined by modules. For errors that are particular to a module, a different set of predefined values has been reserved for their use. Table 16 lists the range of error numbers defined by KeyWorks for CSP modules and those available for use with individual CSP modules.

Table 16. CSP Module Error Numbers

Error Number Range	Description
1000 – 1999	CSP errors defined by KeyWorks
2000 - 2999	CSP errors reserved for individual CSP modules

The calling application must determine how to handle the error returned by CSSM_GetError. Detailed descriptions of the KeyWorks error values are documented in the *IBM KeyWorks Toolkit Application Programming Interface Specification* and the cssmerr.h header file. Errors specific to individual CSP modules are defined in the CSP's documentation. If a routine does not know how to handle the error, it may choose to pass the error to its caller.

A.1 Cryptographic Service Provider Module Errors

Table 17. General CSP Messages and Errors

Error Code	Error Name
1001	CSSM_CSP_UNKNOWN_ERROR
1002	CSSM_CSP_REGISTER_ERROR
1003	CSSM_CSP_VERSION_ERROR
1004	CSSM_CSP_CONVERSION_ERROR
1005	CSSM_CSP_NO_TOKENINFO
1006	CSSM_CSP_INTERNAL_ERROR
1007	CSSM_CSP_SERIAL_REQUIRED
1008	CSSM_CSP_NOT_IMPLEMENTED

Table 18. CSP Memory Errors

Error Code	Error Name
1010	CSSM_CSP_MEMORY_ERROR
1011	CSSM_CSP_NOT_ENOUGH_BUFFER
1012	CSSM_CSP_ERR_OUTBUF_LENGTH
1013	CSSM_CSP_NO_OUTBUF
1014	CSSM_CSP_ERR_INBUF_LENGTH
1015	CSSM_CSP_ERR_KEYBUF_LENGTH
1016	CSSM_CSP_NO_SLOT

Table 19. Invalid CSP Parameters

Error Code	Error Name
1020	CSSM_CSP_INVALID_CSP_HANDLE
1021	CSSM_CSP_INVALID_POINTER
1022	CSSM_CSP_INVALID_CERTIFICATE
1023	CSSM_CSP_INVALID_ALGORITHM
1024	CSSM_CSP_INVALID_WINDOW_HANDLE
1025	CSSM_CSP_INVALID_CALLBACK
1026	CSSM_CSP_INVALID_CONTEXT
1027	CSSM_CSP_INVALID_CONTEXT_HANDLE
1028	CSSM_CSP_INVALID_CONTEXT_POINTER
1029	CSSM_CSP_INVALID_DATA_POINTER
1030	CSSM_CSP_INVALID_DATA_COUNT
1031	CSSM_CSP_INVALID_KEY_LENGTH
1032	CSSM_CSP_INVALID_KEY
1033	CSSM_CSP_INVALID_KEY_POINTER
1034	CSSM_CSP_INVALID_ALGORITHM_MODE
1035	CSSM_CSP_INVALID_PADDING
1036	CSSM_CSP_INVALID_KEY_ATTRIBUTE
1037	CSSM_CSP_INVALID_PARAM_LENGTH
1038	CSSM_CSP_INVALID_IV_SIZE
1039	CSSM_CSP_INVALID_SIGNATURE
1040	CSSM_CSP_INVALID_DEVICE_ID
1041	CSSM_CSP_INVALID_KEYCLASS
1042	CSSM_CSP_INVALID_MODULE_HANDLE
1043	CSSM_CSP_INVALID_KEY_TYPE
1044	CSSM_CSP_INVALID_ITERATION_COUNT

Table 20. File I/O Errors

Error Code	Error Name
1050	CSSM_CSP_FILE_NOT_EXISTS
1051	CSSM_CSP_FILE_NOT_OPEN
1052	CSSM_CSP_FILE_OPEN_FAILED
1053	CSSM_CSP_FILE_CREATE_FAILED
1054	CSSM_CSP_FILE_READ_FAILED
1055	CSSM_CSP_FILE_WRITE_FAILED
1056	CSSM_CSP_FILE_CLOSE_FAILED
1057	CSSM_CSP_FILE_COPY_FAILED
1058	CSSM_CSP_FILE_DELETE_FAILED
1059	CSSM_CSP_FILE_FORMAT_ERROR

Table 21. CSP Cryptographic Errors

Error Code	Error Name
1065	CSSM_CSP_PUBKEY_GET_ERROR
1066	CSSM_CSP_QUERY_SIZE_FAILED
1067	CSSM_CSP_UNKNOWN_ALGORITHM
1068	CSSM_CSP_OPERATION_UNSUPPORTED
1069	CSSM_CSP_VECTOROFBUFS_UNSUPPORTED
1070	CSSM_CSP_STAGED_OPERATION_UNSUPPORTED
1071	CSSM_CSP_KEY_MODULUS_UNSUPPORTED
1072	CSSM_CSP_KEY_LENGTH_UNSUPPORTED
1073	CSSM_CSP_PADDING_UNSUPPORTED
1074	CSSM_CSP_IV_SIZE_UNSUPPORTED
1075	CSSM_CSP_GET_APIMEMFUNC_ERROR
1076	CSSM_CSP_INPUT_LENGTH_OVERSIZE
1077	CSSM_CSP_INPUT_LENGTH_ERROR
1078	CSSM_CSP_INPUT_DATA_ERROR
1079	CSSM_CSP_UNSUPPORTED_STORAGE_MASK
1080	CSSM_CSP_OPERATION_IN_PROGRESS
1081	CSSM_CSP_NO_WRITE_PERMISSIONS
1082	CSSM_CSP_EXCLUSIVE_UNAVAILABLE
1083	CSSM_CSP_UPDATE_WITHOUT_INIT
1084	CSSM_CSP_LOGIN_FAILED
1085	CSSM_CSP_ALREADY_LOGGED_IN
1086	CSSM_CSP_NOT_LOGGED_IN
1087	CSSM_CSP_KEY_PROTECTED
1088	CSSM_CSP_CALLBACK_FAILED
1089	CSSM_CSP_ROUNDS_UNSUPPORTED
1090	CSSM_CSP_EFFECTIVE_BITS_UNSUPPORTED
1091	CSSM_CSP_INCOMPATIBLE_VERSION
1092	CSSM_CSP_INCOMPATIBLE_KEY_VERSION
1093	CSSM_CSP_ALGORITHM_UNSUPPORTED
1094	CSSM_CSP_OPERATION_FAILED

Table 22. Missing or Invalid CSP Parameters

Error Code	Error Name
1100	CSSM_CSP_PARAM_NO_PARAM
1101	CSSM_CSP_PARAM_NO_PASSWORD
1102	CSSM_CSP_PARAM_NO_SEED
1103	CSSM_CSP_PARAM_NO_KEY
1104	CSSM_CSP_PARAM_NO_SALT
1105	CSSM_CSP_PARAM_NO_MODULUS
1106	CSSM_CSP_PARAM_NO_OUTPUT_SIZE
1108	CSSM_CSP_PARAM_NO_KEY_LENGTH
1109	CSSM_CSP_PARAM_NO_MODE
1110	CSSM_CSP_PARAM_NO_DATA
1111	CSSM_CSP_PARAM_NO_INIT_VECTOR
1112	CSSM_CSP_PARAM_NO_PADDING
1113	CSSM_CSP_PARAM_NO_ROUNDS
1114	CSSM_CSP_PARAM_NO_RANDOM
1115	CSSM_CSP_PARAM_NO_REMAINDATA
1116	CSSM_CSP_PARAM_NO_ALG_PARAMS
1117	CSSM_CSP_PARAM_INVALID_VALUE
1118	CSSM_CSP_PARAM_NO_EFFECTIVE_BITS
1119	CSSM_CSP_PARAM_NO_PRIME
1120	CSSM_CSP_PARAM_NO_BASE
1121	CSSM_CSP_PARAM_NO_SUBPRIME
1122	CSSM_CSP_PARAM_NO_ALG_ID
1123	CSSM_CSP_PARAM_NO_KEY_TYPE
1124	CSSM_CSP_PARAM_NO_ITERATION_COUNT

Table 23. Password Errors

Error Code	Error Name
1130	CSSM_CSP_PASSWORD_INCORRECT
1131	CSSM_CSP_PASSWORD_SAME
1132	CSSM_CSP_PASSWORD_LENGTH_ERROR
1133	CSSM_CSP_PASSWORD_INVALID

Table 24. Key Management Messages and Errors

Error Code	Error Name
1140	CSSM_CSP_PRIKEY_LOAD_ERROR
1141	CSSM_CSP_PRIKEY_NOT_FOUND
1142	CSSM_CSP_PRIKEY_ALREADY_EXIST
1143	CSSM_CSP_PRIKEY_GET_ERROR
1144	CSSM_CSP_PRIKEY_PUBKEY_INCONSISTENT
1150	CSSM_CSP_KEY_DUPLICATE
1151	CSSM_CSP_KEY_BAD_KEY
1152	CSSM_CSP_KEY_BAD_LENGTH
1153	CSSM_CSP_KEY_NO_PARAM
1154	CSSM_CSP_KEY_ALGID_NOTMATCH
1155	CSSM_CSP_KEY_BLOBTYPE_INCORRECT
1156	CSSM_CSP_KEY_CLASS_INCORRECT
1157	CSSM_CSP_KEY_DELETE_FAILED
1158	CSSM_CSP_KEY_USAGE_INCORRECT

Error Code	Error Name
1159	CSSM_CSP_KEY_NOT_PROTECTED
1160	CSSM_CSP_KEY_FORMAT_INCORRECT

Table 25. Random Number Generation (RNG) Messages and Errors

Error Code	Error Name
1200	CSSM_CSP_RNG_FAILED
1201	CSSM_CSP_RNG_UNKNOWN_ALGORITHM
1202	CSSM_CSP_RNG_NO_METHOD

Table 26. Unique ID Generation Messages and Errors

Error Code	Error Name
1220	CSSM_CSP_UIDG_FAILED
1221	CSSM_CSP_UIDG_UNKNOWN_ALGORITHM
1222	CSSM_CSP_UIDG_NO_METHOD

Table 27. Key Generation Messages and Errors

Error Code	Error Name
1210	CSSM_CSP_KEYGEN_FAILED
1211	CSSM_CSP_KEYGEN_UNKNOWN_ALGORITHM
1212	CSSM_CSP_KEYGEN_NO_METHOD

Table 28. Encryption/Decryption Messages

Error Code	Error Name
1230	CSSM_CSP_ENC_UNKNOWN_ALGORITHM
1231	CSSM_CSP_ENC_NO_METHOD
1232	CSSM_CSP_ENC_FAILED
1233	CSSM_CSP_ENC_INIT_FAILED
1234	CSSM_CSP_ENC_UPDATE_FAILED
1235	CSSM_CSP_ENC_FINAL_FAILED
1236	CSSM_CSP_ENC_BAD_IV_LENGTH
1237	CSSM_CSP_ENC_IV_ERROR
1238	CSSM_CSP_ENC_BAD_KEY_LENGTH
1239	CSSM_CSP_ENC_UNKNOWN_MODE
1250	CSSM_CSP_DEC_UNKNOWN_ALGORITHM
1251	CSSM_CSP_DEC_NO_METHOD
1253	CSSM_CSP_DEC_FAILED
1254	CSSM_CSP_DEC_INIT_FAILED
1255	CSSM_CSP_DEC_UPDATE_FAILED
1256	CSSM_CSP_DEC_FINAL_FAILED
1257	CSSM_CSP_DEC_BAD_IV_LENGTH
1258	CSSM_CSP_DEC_IV_ERROR
1259	CSSM_CSP_DEC_BAD_KEY_LENGTH
1260	CSSM_CSP_DEC_UNKNOWN_MODE

Table 29. Sign/Verify Messages and Errors

Error Code	Error Name
1350	CSSM_CSP_SIGN_UNKNOWN_ALGORITHM
1351	CSSM_CSP_SIGN_NO_METHOD
1352	CSSM_CSP_SIGN_FAILED
1353	CSSM_CSP_SIGN_INIT_FAILED
1354	CSSM_CSP_SIGN_UPDATE_FAILED
1355	CSSM_CSP_SIGN_FINAL_FAILED
1360	CSSM_CSP_VERIFY_FAILED
1361	CSSM_CSP_VERIFY_INIT_FAILED
1362	CSSM_CSP_VERIFY_UPDATE_FAILED
1363	CSSM_CSP_VERIFY_FINAL_FAILED
1365	CSSM_CSP_VERIFY_UNKNOWN_ALGORITHM
1366	CSSM_CSP_VERIFY_NO_METHOD

Table 30. Digest Function Errors

Error Code	Error Name
1380	CSSM_CSP_DIGEST_UNKNOWN_ALGORITHM
1382	CSSM_CSP_DIGEST_NO_METHOD
1383	CSSM_CSP_DIGEST_FAILED
1384	CSSM_CSP_DIGEST_INIT_FAILED
1385	CSSM_CSP_DIGEST_UPDATE_FAILED
1386	CSSM_CSP_DIGEST_CLONE_FAILED
1387	CSSM_CSP_DIGEST_FINAL_FAILED

Table 31. MAC Function Errors

Error Code	Error Name
1390	CSSM_CSP_MAC_UNKNOWN_ALGORITHM
1392	CSSM_CSP_MAC_NO_METHOD
1393	CSSM_CSP_MAC_FAILED
1394	CSSM_CSP_MAC_INIT_FAILED
1395	CSSM_CSP_MAC_UPDATE_FAILED
1396	CSSM_CSP_MAC_CLONE_FAILED
1397	CSSM_CSP_MAC_FINAL_FAILED

Table 32. Key Exchange Errors

Error Code	Error Name
1410	CSSM_CSP_KEYEXCH_GENPARAM_FAILED
1411	CSSM_CSP_KEYEXCH_PHASE1_FAILED
1412	CSSM_CSP_KEYEXCH_PHASE2_FAILED
1413	CSSM_CSP_KEYEXCH_UNKNOWN_ALGORITHM
1414	CSSM_CSP_KEYEXCH_NO_METHOD

Table 33. PassThrough Custom Errors

Error Code	Error Name
1420	CSSM_CSP_INVALID_PASSTHROUGH_ID
1421	CSSM_CSP_INVALID_PASSTHROUGH_PARAMS

Table 34. Wrap/Unwrap Errors

Error Code	Error Name
1450	CSSM_CSP_WRAP_UNKNOWN_ALGORITHM
1451	CSSM_CSP_WRAP_NO_METHOD
1452	CSSM_CSP_WRAP_FAILED
1456	CSSM_CSP_UNWRAP_UNKNOWN_ALGORITHM
1457	CSSM_CSP_UNWRAP_NO_METHOD
1458	CSSM_CSP_UNWRAP_FAILED

Table 35. Hardware CSP Errors

Error Code	Error Name
1470	CSSM_CSP_DEVICE_ERROR
1471	CSSM_CSP_DEVICE_MEMORY_ERROR
1472	CSSM_CSP_DEVICE_REMOVED
1473	CSSM_CSP_DEVICE_NOT_PRESENT
1474	CSSM_CSP_DEVICE_UNKNOWN
1490	CSSM_CSP_PERMISSIONS_READ_ONLY
1491	CSSM_CSP_PERMISSIONS_WRITE_PROTECT
1492	CSSM_CSP_PERMISSIONS_NOT_EXCLUSIVE

Table 36. Query Size Errors

Error Code	Error Name
1500	CSSM_CSP_QUERY_SIZE_UNKNOWN
1501	CSSM_CSP_QUERY_KEYSIZEINBITS_UNKNOWN

Appendix B. List of Acronyms

API	Application Programming Interface
CA	Certificate Authority
CL	Certificate Library
CRL	Certificate Revocation List
CSP	Cryptographic Service Provider
DES	Data Encryption Standard
DL	Data Storage Library
DLL	Dynamic Link Library
DSA	Digital Signature Algorithm
ECB	Electronic Code Book
GUID	Globally Unique ID
IDEA	International Data Encryption Algorithm
ISO	International Organization for Standardization
ISV	Independent Software Vendor
KRF	Key Recovery Field
KRSP	Key Recovery Service Provider
MAC	Message Authentication Code
OAEP	Optimal Asymmetric Encryption Padding
PKCS	Public-Key Cryptographic Standard
SET	Secure Electronic Transaction
SPI	Service Provider Interface
SSL	Secure Sockets Layer
TP	Trust Policy
UTC	Coordinated Universal Time

Appendix C. Glossary

Asymmetric algorithms	Cryptographic algorithms, where one key is used to encrypt and a second key is used to decrypt. They are often called public-key algorithms. One key is called the public key, and the other is called the private key or secret key. RSA (Rivest-Shamir-Adelman) is the most commonly used public-key algorithm. It can be used for encryption and for signing.
Authentication Information	Information that is verified for authentication. For example, a Key Recovery Officer (KRO) selects a password which will be used for authentication with the Key Recovery Coordinator (KRC). A KRO operator who has identification information must search the Authentication Information (AI) database to locate an AI value that corresponds to the individual who generated the information.
Certificate	See Digital certificate.
Certificate Authority	An entity that guarantees or sponsors a certificate. For example, a credit card company signs a cardholder's certificate to assure that the cardholder is who he or she claims to be. The credit card company is a Certificate Authority (CA). CAs issue, verify, and revoke certificates.
Certificate chain	The hierarchical chain of all the other certificates used to sign the current certificate. This includes the CA who signs the certificate, the CA who signed that CA's certificate, and so on. There is no limit to the depth of the certificate chain.
Certificate signing	The CA can sign certificates it issues or co-sign certificates issued by another CA. In a general signing model, an object signs an arbitrary set of one or more objects. Hence, any number of signers can attest to an arbitrary set of objects. The arbitrary objects could be, for example, pieces of a document for libraries of executable code.
Certificate validity date	A start date and a stop date for the validity of the certificate. If a certificate expires, the CA may issue a new certificate.
Cryptographic algorithm	A method or defined mathematical process for implementing a cryptography operation. A cryptographic algorithm may specify the procedure for encrypting and decrypting a byte stream, digitally signing an object, computing the hash of an object, generating a random number, etc. IBM KeyWorks accommodates Data Encryption Standard (DES), RC2, RC4, International Data Encryption Algorithm (IDEA), and other encryption algorithms.
Cryptographic Service Provider	Cryptographic Service Providers (CSPs) are modules that provide secure key storage and cryptographic functions. The modules may be software only or hardware with software drivers. The cryptographic functions provided may include: • Bulk encryption and decryption • Digital signing • Cryptographic hash

- Random number generation
- Key exchange

Cryptography

The science for keeping data secure. Cryptography provides the ability to store information or to communicate between parties in such a way that prevents other non-involved parties from understanding the stored information or accessing and understanding the communication. The encryption process takes understandable test and transforms it into an unintelligible piece of data (called ciphertext); the decryption process restores the understandable text from the unintelligible data. Both involve a mathematical formula or algorithm and a secret sequence of data called a key. Cryptographic services provide confidentiality (keeping data secret), integrity (preventing data from being modified), authentication (proving the identity of a resource or a user), and non-repudiation (providing proof that a message or transaction was sent and/or received).

There are two types of cryptography:

- In shared/secret key (symmetric) cryptography there is only one key that is shared secret between the two communicating parties. The same key is used for encryption and decryption.

- In public key (asymmetric) cryptography different keys are used for encryption and decryption. A party has two keys: a public key and a private key. The two keys are mathematically related, but it is virtually impossible to derive the private key from the public key. A message that is encrypted with someone's public key (obtained from some public directory) can only be decrypted with the associated private key. Alternately, the private key can be used to "sign" a document; the public key can be used as verification of the source of the document.

Cryptoki

Short for cryptographic token interface. See Token.

Data Encryption Standard

In computer security, the National Institute of Standards and Technology (NIST) Data Encryption Standard (DES), adopted by the U.S. Government as Federal Information Processing Standard (FIPS) Publication 46, which allows only hardware implementations of the data encryption algorithm.

Digital certificate

The binding of some identification to a public key in a particular domain, as attested to directly or indirectly by the digital signature of the owner of that domain. A digital certificate is an unforgettable credential in cyberspace. The certificate is issued by a trusted authority, covered by that party's digital signature. The certificate may attest to the certificate holder's identity, or may authorize certain actions by the certificate holder. A certificate may include multiple signatures and may attest to multiple objects or multiple actions.

Digital signature

A data block that was created by applying a cryptographic signing algorithm to some other data using a secret key. Digital signatures may be used to:

- Authenticate the source of a message, data, or document
- Verify that the contents of a message has not been modified since it was signed by the sender
- Verify that a public key belongs to a particular person

	Typical digital signing algorithms include MD5 with RSA encryption, and DSS, the proposed Digital Signature Standard defined as part of the U.S. Government Capstone project.
Enterprise	A company or individual who is authorized to submit on-line requests to the Key Recovery Officer (KRO). In the enterprise key recovery scenario, a process at the enterprise called the KRO is responsible for preparing key recovery requests and communicating them to the KRC. The KRO, acting on behalf of an enterprise or individual, sends an on-line request to the Key Recovery Coordinator (KRC) to recover a key from a Key Recovery Block (KRB).
Graphical User Interface	A type of display format that enables the user to choose commands, start programs, and see lists of files and other options by pointing to pictorial representations (icons) and lists of menu items on the screen. Graphical User Interfaces (GUIs) are used by the Microsoft Windows program for IBM-compatible microcomputers and by other systems.
Hash algorithm	A cryptographic algorithm used to hash a variable-size input stream into a unique, fixed-sized output value. Hashing is typically used in digital signing algorithms. Example hash algorithms include MD and MD2 from RSA Data Security. MD5, also from RSA Data Security, hashes a variable-size input stream into a 128-bit output value. SHA, a Secure Hash Algorithm published by the U.S. Government, produces a 160-bit hash value from a variable-size input stream.
IBM KeyWorks Architecture	A set of layered security services that address communications and data security problems in the emerging PC business space.
IBM KeyWorks Framework	The IBM KeyWorks Framework defines five key service components: • Cryptographic Module Manager • Key Recovery Module Manager • Trust Policy Module Manager • Certificate Library Module Manager • Data Storage Library Module Manager IBM KeyWorks binds together all the security services required by PC applications. In particular, it facilitates linking digital certificates to cryptographic actions and trust protocols.
Key Escrow	The storing of a key (or parts of a key) with a trusted party or trusted parties in case of loss or destruction of the key.
Key Recovery Agent	The Key Recovery Agent (KRA) acts as the back end for a key recovery operation. The KRA can only be accessed through an on-line communication protocol via the Key Recovery Coordinator (KRC). KRAs are considered outside parties involved in the key recovery process; they are analogous to the neighbors who each hold one digit of the combination of the lock box containing the key. The authorized parties (i.e., enterprise or law enforcement) have the freedom to choose the number of specific KRAs that they want to use. The authorized party requests that each KRA decrypt its section of the Key Recovery Fields (KRFs) that is associated with the transmission. Then those pieces of information are used in the process that derives the session key. The KRA will only be able to recover a portion of the key, and reading the original message

will require searching the remaining key space in order to find the key that will decrypt the message. The number of KRAs on each end of the communication does not have to be equal.

Key Recovery Block

The Key Recovery Block (KRB) is a piece of encrypted information that is contained within a block. The KRS components (i.e., KRO, KRC, KRA) work collectively to recover a session key from a provided KRB. In the enterprise scenario, the KRO has both the KRB and the credentials that authenticate it to receive the recovered key. This information will be transmitted over the network to the KRC. In the law enforcement scenario, the KRB is presented on a 3.5-inch diskette, and the credentials are in the physical form of a legal warrant. This warrant will specify any information available to the law enforcement agents which can be used to tie the warrant to the identity of the user for whom

KRBs were generated (i.e., username, hostname, IP address). The KRC has the ability to check credentials and derive the original encryption key from the KRB with the help of its KRAs.

Key Recovery Coordinator

The Key Recovery Coordinator (KRC) acts as the front end for the key recovery operation. The KRO, acting on behalf of an enterprise or individual, sends an on-line request to the KRC to recover a key from a KRB. The KRC receives the on-line request and services it by interacting with the appropriate set of KRAs as specified within the KRB. The recovered key is then sent back to the KRO by the KRC using an on-line protocol. The KRC consists of one main application which, when started, behaves as a server process. The system, which serves as the KRC, may be configured to start the KRC application as part of system services; alternatively, the KRC operator can start up the KRC application manually. The KRC application performs the following operations:

- Listens for on-line recovery requests from KRO
- Can be used to launch an embedded application that allows manual key recovery for law enforcement
- Monitors and displays the status of the recovery requests being serviced

Key Recovery Field

A Key Recovery Field (KRF) is a block of data that is created from a symmetric key and key recovery profile information. The Key Recovery Service Provider (KRSP) is invoked from the IBM KeyWorks framework to create KRFs. There are two major pieces of the KRFs: block 1 contains information that is unrelated to the session key of the transmitted message, and encrypted with the public keys of the selected key recovery agents; block 2 contains information that is related to the session key of the transmission. The KRSP generates the KRFs for the session key. This information is *not* the key or any portion of the key, but is information that can be used to recover the key. The KRSP has access to location-unique jurisdiction policy information that controls and modifies some of the steps in the generation of the KRFs. Only once the KRFs are generated, and both the client and server sides have access to them, can the encrypted message flow begin. KRFs are generated so that they can be used by a KRA to recover the original symmetric key, either because the user who generated the message has lost the key, or at the warranted request of law enforcement agents.

Key Recovery Module Manager

The Key Recovery Module Manager enables key recovery for cryptographic services obtained through the KeyWorks. It mediates all cryptographic services provided by the KeyWorks and applies the appropriate key recovery policy on all such operations. The Key Recovery Module Manager contains a Key Recovery Policy Table (KRPT) that defines the applicable key recovery policy for all cryptographic products. The Key Recovery Module Manager routes the KR-API function calls made by an application to the appropriate KR-SPI functions. The Key Recovery Module Manager also enforces the key recovery policy on all cryptographic operations that are obtained through the KeyWorks. It maintains key recovery state in the form of key recovery contexts.

Key Recovery Officer

An entity called the Key Recovery Officer (KRO) is the focal point of the key recovery process. In the enterprise key recovery scenario, the KRO is responsible for preparing key recovery requests and communicating them to the KRC. The KRO has both the KRB and the credentials that authenticate it to receive the recovered key. The KRO is the entity that acts on behalf of an enterprise to initiate a key recovery request operation. An employee within an enterprise who desires key recovery will send a request to the KRO with the KRB that is to be recovered. The actual key recovery phase begins when the KRO operator uses the KRO application to initiate a key recovery request to the appropriate KRC. At this time, the operator selects a KRB to be sent for recovery, enters the Authentication Information (AI) information that can be used to authenticate the request to the KRC, and submits the request.

Key Recovery Policy

Key recovery policies are mandatory policies that are typically derived from jurisdiction-based regulations on the use of cryptographic products for data confidentiality. Often, the jurisdictions for key recovery policies coincide with the political boundaries of countries in order to serve the law enforcement and intelligence needs of these political jurisdictions. Political jurisdictions may choose to define key recovery policies for cryptographic products based on export, import, or use controls. Enterprises may define internal and external jurisdictions, and may mandate key recovery policies on the cryptographic products within their own jurisdictions.

Key recovery policies come in two flavors: *key recovery enablement policies* and *key recovery interoperability policies.* Key recovery enablement policies specify the exact cryptographic protocol suites (e.g., algorithms, modes, key lengths, etc.) and perhaps usage scenarios, where key recovery enablement is mandated. Furthermore, these policies may also define the number of bits of the cryptographic key that may be left out of the key recovery enablement operation; this is typically referred to as the *workfactor*. Key recovery interoperability policies specify to what degree a key recovery enabled cryptographic product is allowed to interoperate with other cryptographic products.

Key Recovery Server

The Key Recovery Server (KRS) consists of three major entities: Key Recovery Coordinator (KRC), Key Recovery Agent (KRA), and Key Recovery Officer (KRO). The KRS is intended to be used by enterprise employees and security personnel, law enforcement personnel, and KRSF personnel. The KRS interacts with one or more local or remote KRAs to reconstruct the secret key that can be used to decrypt the ciphertext.

Key Recovery Server Facility	The Key Recovery Server Facility (KRSF) is a facility room that houses the KRS component facilities ensuring they operate within a secure environment that is highly resistant to penetration and compromise. Several physical and administrative security procedures must be followed at the KRSF such as a combination keyed lock, limited personnel, standalone system, operating system with security features (Microsoft NT Workstation 4.0), NTFS (Windows NT Filesystem), and account and auditing policies.
Key Recovery Service Provider	Key Recovery Service Providers (KRSPs) are modules that provide key ecovery enablement functions. The cryptographic functions provided may include: • Key recovery field generation • Key recovery field processing
Law Enforcement	A type of scenario where key recovery is mandated by the jurisdictional law enforcement authorities in the interest of national security and law enforcement. In the law enforcement scenario, the KRB is presented on a 3.5-inch diskette, and the credentials are in the physical form of a legal warrant. This warrant will specify any information available to the law enforcement agents which can be used to tie the warrant to the identity of the user for whom KRBs were generated (i.e., username, hostname, IP address).
Leaf certificate	The certificate in a certificate chain that has not been used to sign another certificate in that chain. The leaf certificate is signed directly or transitively by all other certificates in the chain.
Message digest	The digital fingerprint of an input stream. A cryptographic hash function is applied to an input message arbitrary length and returns a fixed-size output, which is called the digest value.
Owned certificate	A certificate whose associated secret or private key resides in a local Cryptographic Service Provider (CSP). Digital-signing algorithms require using owned certificates when signing data for purposes of authentication and non-repudiation. A system may use certificates it does not own for purposes other than signing.
Private key	The cryptographic key is used to decipher messages in public-key cryptography. This key is kept secret by its owner.
Public key	The cryptographic key is used to encrypt messages in public-key cryptography. The public key is available to multiple users (i.e., the public).
Random number generator	A function that generates cryptographically strong random numbers that cannot be easily guessed by an attacker. Random numbers are often used to generate session keys.
Root certificate	The prime certificate, such as the official certificate of a corporation or government entity. The root certificate is positioned at the top of the certificate hierarchy in its domain, and it guarantees the other certificates in its certificate chain. Each Certificate Authority (CA) has a self-signed root certificate. The root certificate's public key is the foundation of signature verification in its domain.

Secure Electronic Transaction	A mechanism for securely and automatically routing payment information among users, merchants, and their banks. Secure Electronic Transaction (SET) is a protocol for securing bankcard transactions on the Internet or other open networks using cryptographic services. SET is a specification designed to utilize technology for authenticating parties involved in payment card purchases on any type of on-line network, including the Internet. SET was developed by Visa and MasterCard, with participation from leading technology companies, including Microsoft, IBM, Netscape, SAIC, GTE, RSA, Terisa Systems, and VeriSign. By using sophisticated cryptographic techniques, SET will make cyberspace a safer place for conducting business and is expected to boost consumer confidence in electronic commerce. SET focuses on maintaining confidentiality of information, ensuring message integrity, and authenticating the parties involved in a transaction. The significance of SET, over existing Internet security protocols, is found in the use of digital certificates. Digital certificates will be used to authenticate all the parties involved in a transaction. SET will provide those in a virtual world with the same level of trust and confidence a consumer has today when making a purchase at any of the 13 million Visa-acceptance locations in the physical world. The SET specification is open and free to anyone who wishes to use it to develop SET-compliant software for buying or selling in cyberspace.
Security Context	A control structure that retains state information shared between a CSP and the application agent requesting service from the CSP. Only one context can be active for an application at any given time, but the application is free to switch among contexts at will, or as required. A security context specifies CSP and application-specific values, such as required key length and desired hash functions.
Security-relevant event	An event where a CSP-provided function is performed, a security module is loaded, or a breach of system security is detected.
Session key	A cryptographic key used to encrypt and decrypt data. The key is shared by two or more communicating parties, who use the key to ensure privacy of the exchanged data.
Signature	See Digital signature.
Signature chain	The hierarchical chain of signers, from the root certificate to the leaf certificate, in a certificate chain.
Smart Card	A device (usually similar in size to a credit card) that contains an embedded microprocessor that could be used to store information. Smart cards can store credentials used to authenticate the holder.

S/MIME	Secure/Multipurpose Internet Mail Extensions (S/MIME) is a protocol that adds digital signatures and encryption to Internet MIME messages. MIME is the official proposed standard format for extended Internet electronic mail. Internet e-mail messages consist of two parts, the header and the body. The header forms a collection of field/value pairs structured to provide information essential for the transmission of the message. The body is normally unstructured unless the e-mail is in MIME format. MIME defines how the body of an e-mail message is structured. The MIME format permits e-mail to include enhanced text, graphics, audio, and more in a standardized manner via MIME-compliant mail systems. However, MIME itself does not provide any security services. The purpose of S/MIME is to define such services, following the syntax given in PKCS #7 for digital signatures and encryption. The MIME body part carries a PKCS #7 message, which itself is the result of cryptographic processing on other MIME body parts.
Symmetric algorithms	Cryptographic algorithms that use a single secret key for encryption and decryption. Both the sender and receiver must know the secret key. Well-known symmetric functions include Data Encryption Standard (DES) and International Data Encryption Algorithm (IDEA). The U.S. Government endorsed DES as a standard in 1977. It is an encryption block cipher that operates on 64-bit blocks with a 56-bit key. It is designed to be implemented in hardware, and works well for bulk encryption. IDEA, one of the best known public algorithms, uses a 128-bit key.
Token	The logical view of a cryptographic device, as defined by a CSP's interface. A token can be hardware, a physical object, or software. A token contains information about its owner in digital form, and about the services it provides for electronic-commerce and other communication applications. A token is a secure device. It may provide a limited or a broad range of cryptographic functions. Examples of hardware tokens are smart cards and Personal Computer Memory Card International Association (PCMCIA) cards.
Verification	The process of comparing two message digests. One message digest is generated by the message sender and included in the message. The message recipient computes the digest again. If the message digests are exactly the same, it shows or proves there was no tampering of the message contents by a third party (between the sender and the receiver).
Web of trust	A trust network among people who know and communicate with each other. Digital certificates are used to represent entities in the web of trust. Any pair of entities can determine the extent of trust between the two, based on their relationship in the web. Based on the trust level, secret keys may be shared and used to encrypt and decrypt all messages exchanged between the two parties. Encrypted exchanges are private, trusted communications.