IBM KeyWorks Toolkit

Trust Policy Interface (TPI) Specification

001.001.003

Table of Contents

List of Figures

List of Tables

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, IBM KeyWorks Framework, and Service Providers. The IBM 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. Independent Software Vendors (ISVs) and hardware vendors will provide these service providers. The functionality of the service providers may be extended beyond the services defined by the KeyWorks API, by exporting additional services to applications using a 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.

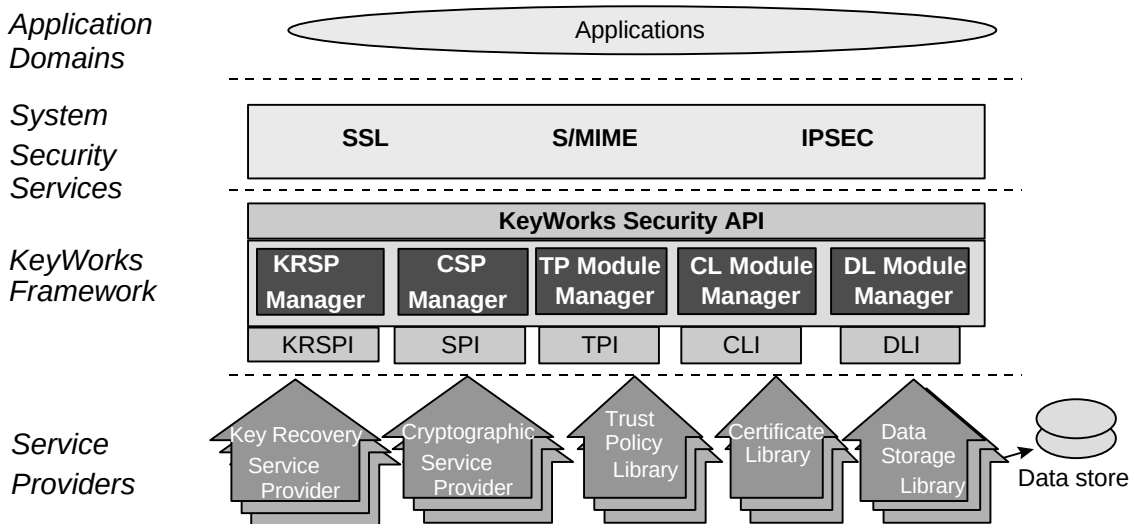

Figure 1. IBM KeyWorks Toolkit Architecture

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 also must 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. Trust Policy Interface

A digital certificate is the binding of some identification to a public key in a particular domain. When a trust domain authority issues (creates and signs) a certificate to a subject, it binds the subject's public key to the identity. This binding obviously can be verified through the signature verification process. The issuing authority also associates a level of trust with the certificate. The actions of the user, whose identity is bound to the certificate, are constrained by the Trust Policy (TP) governing the usage domain of the certificate.
A digital certificate is a subject's credential in cyberspace that cannot be forged.

The use of digital certificates is the basic premise of KeyWorks design. The KeyWorks assumes the concept of digital certificates in its broadest sense. Applications use digital certificates as credential for:

- Identification
- Authentication
- Authorization

The applications interpret and manipulate the contents of certificates to achieve these ends based on the real-world trust model they chose as their model for trust and security. The primary purpose of a TP module is to answer the question, “Is this certificate trusted for this action”? The KeyWorks TP application programming interface (API) determines the generic operations that should be defined for certificate-based trust in every application domain. The specific semantics of each operation is defined by the following:

- Application domain
- Trust model
- Policy statement for a domain
- Certificate type
- Real-world operation the user is trying to perform within the application domain

The trust model is expressed as an executable policy that is used by all applications that ascribe to that policy and the trust model it represents. As an infrastructure, KeyWorks is policy neutral; it does not incorporate any single policy. For example, the verification procedure for a credit card certificate should be defined and implemented by the credit company issuing the certificate. Employee access to a lab housing a critical project should be defined by the company whose intellectual property is at risk. Rather than defining policies, KeyWorks provides the infrastructure for installing and managing policy-specific modules. This ensures complete extensibility of certificate-based trust on every platform hosting KeyWorks.

Different TPs define different actions that an application may request. Some of these actions are common to every TP, and are operations on objects that all trust models use. The objects common to all trust models are certificates and Certificate Revocation Lists (CRLs). The basic operations on these objects are sign, verify, and revoke.

KeyWorks defines a set of API calls that should be implemented by TP modules. These calls allow an application to perform basic operations such as verify, sign-on certificates, and CRLs. More extensible operations can be embedded in the implementation of these APIs.

Application developers and trust domain authorities benefit from the ability to define and implement policy-based modules. Application developers are freed from the burden of implementing a policy description and certifying that their implementation conforms. Instead, the application needs only to build in a list of the authorities and certificate issuers it uses.

Trust domain authorities also benefit from an infrastructure that supports TP modules. Trust domain authorities are ensured that applications using their modules adhere to the policies of the domain. In addition, dynamic download of trust modules (possibly from remote systems) ensures timely and accurate propagation of policy changes. Individual functions within the module may combine local and remote processing. This flexibility allows the module developer to implement policies based on the ability to communicate with a remote authority system. This also allows the policy implementation to be decomposed in any convenient distributed manner.

Implementing a TP module may or may not be tightly coupled with one or more Certificate Library (CL) modules or one or more Data Storage Library (DL) modules. The TP embodies the semantics of the domain. The CL and the DL embody the syntax of a certificate format and operations on that format. A TP can be completely independent of certificate format, or it may be defined to operate with one or a small number of certificate formats. A TP implementation may invoke a CL module and/or a DL module to manipulate certificates.

2.1 Trust Policy Services API

KeyWorks defines eight API calls that TP modules can implement. These calls implement various categories of operations that can be performed on trust objects.

Signing Certificates and Certificate Revocation Lists. Every system should be capable of being a Certificate Authority (CA), if so authorized. CAs are applications that issue and validate certificates and CRLs. Issuing certificates and CRLs include initializing their attributes and digitally signing the result using the private key of the issuing authority. The private key used for signing is associated with the signer's certificate. The TP module must evaluate the trustworthiness of the signer's certificate before performing this operation. Some policies may require that multiple authorities sign an issued certificate. If the TP trusts the signer's certificate, then the TP module may perform the cryptographic signing algorithm by invoking the signing function in a CL module, or by directly invoking the data signing function in a Cryptographic Service Provider (CSP) module. The CL functions that can be used to carry out some of the TP operations are documented in the *IBM KeyWorks Toolkit Certificate Library Interface Specification*.

Verifying Certificates and Certificate Revocation Lists. The TP module determines the trustworthiness of a CRL received from a remote system. The test focuses on the trustworthiness of the agent who signed the CRL. The TP module may need to perform operations on the certificate or CRL to determine trustworthiness. If these operations depend on the data format of the certificate or CRL, the TP module uses the services of a CL module to perform these checks.

Revoking Certificates. When revoking a certificate, the identity of the revoking agent is presented in the form of another certificate. The TP module must determine trustworthiness of the revoking agent's certificate to perform revocation. If the requesting agent's certificate is trustworthy, the TP module carries out the operation directly by invoking a CL module to add a new revocation record to a CRL, marking the certificate as revoked. The KeyWorks API also defines a reason parameter that is passed to the TP module. The TP may use this parameter as part of its trust evaluation.

PassThrough Function. For operations not defined in the TPI, the passthrough function allows the TP module to provide support for these services to clients. These private services are identified by operation identifiers. TP module developers must provide documentation of these services.

2.2 Trust Operations

TP_CertSign

Determines whether the signer's certificate is authorized to perform the signing operation. If so, the TP module carries out the operation. The *scope* of a signature may be used to identify which certificate field should be signed. An example is the case of multiple signatures on a certificate. Should signatures be applied to just the certificate, or to the certificate and all currently existing signatures, as a notary public would do?

TP_CertRevoke

Determines whether the revoker's certificate is trusted to perform/sign the revocation. If so, the TP module carries out the operation by adding a new revocation record to the CRL.

TP_CrlVerify

Determines whether the CRL is trusted. This test may include verifying the correctness of the signature associated with the CRL, determining whether the CRL has been tampered with, and determining if the agent who signed the CRL was trusted to do so.

TP_CrlSign

Determines whether the certificate is trusted to sign the CRL. If so, the TP module carries out the operation.

TP_ApplyCrlToDb

Determines whether the memory-resident CRL is trusted and should be applied to a persistent database, which could result in designating certificates as revoked.

TP_CertGroupConstruct

Constructs a collection of certificates that forms a semantically related trust-relationship.

TP_CertGroupPrune

Removes from a collection of certificates those that do not participate in a semantically related trust-relationship outside of the local system.

TP_CertGroupVerify

Verifies the signatures on each certificate in a group of certificates.

2.3 Extensibility Functions

TP_PassThrough

Executes TP module custom operations. This function accepts as input an operation ID and an arbitrary set of input parameters. The operation ID may specify any type of operation the TP wishes to export. Such operations may include queries or services specific to the domain represented by the TP module.

2.4 Data Structures

This section describes the data structures that may be passed to or returned from a TP function. They will be used by applications to prepare data to be passed as input parameters into KeyWorks API function calls that will be passed without modification to the appropriate TP. The TP 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 KeyWorks.

2.4.1 Basic Data Types

```
typedef unsigned char uint8;
typedef unsigned short uint16;
typedef short sint16;
typedef unsigned int uint32;
typedef int sint32;

#define CSSM_MODULE_STRING_SIZE  64
typedef char CSSM_STRING [CSSM_MODULE_STRING_SIZE + 4];
```

2.4.2 CSSM_BOOL

This data type is used to indicate a true or false condition.

```
typedef uint32 CSSM_BOOL;

#define CSSM_TRUE 1
define CSSM_FALSE 0
```

Definitions:

CSSM_TRUE - Indicates a true result or a true value.

CSSM_FALSE - Indicates a false result or a false value.

2.4.3 CSSM_CERTGROUP

This structure contains a set of certificates. It is assumed that the certificates are related based on the signature hierarchy. A typical group is a chain of certificates. The certificate group is a syntactic representation of a trust model. All certificates in the group must be of the same type and issued for the same trust domain.

```
typedef struct cssm_certgroup{
    uint32 NumCerts;
    CSSM_DATA_PTR CertList;
    void *reserved;
} CSSM_CERTGROUP, *CSSM_CERTGROUP_PTR;
```

Definitions:

NumCerts - Number of certificates in the group.

CertList - List of certificates.

reserved - Reserved for future use.

2.4.4 CSSM_DATA

The CSSM_DATA structure associates 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;
    uint8* Data;
} CSSM_DATA, *CSSM_DATA_PTR
```

Definitions:

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

Data - A pointer to a contiguous block of memory.

2.4.5 CSSM_DL_DB_HANDLE

This data structure holds a pair of handles, one for a DL and another for a data store opened and being managed by the DL.

```
typedef struct cssm_dl_db_handle {
    CSSM_DL_HANDLE DLHandle;
    CSSM_DB_HANDLE DBHandle;
} CSSM_DL_DB_HANDLE, *CSSM_DL_DB_HANDLE_PTR;
```

Definitions:

DLHandle - Handle of an attached module that provides DL services.

DBHandle - Handle of an open data store that is currently under the management of the DL module specified by the DLHandle.

2.4.6 CSSM_DL_DB_LIST

This data structure defines a list of handle pairs (DL handle, data store handle).

```
typedef struct cssm_dl_db_list {
    uint32 NumHandles;
    CSSM_DL_DB_HANDLE_PTR DLDBHandle;
} CSSM_DB_LIST, *CSSM_DB_LIST_PTR;
```

Definitions:

NumHandles - Number of pairs in the list (DL handle, data store handle).

DLDBHandle - List of pairs (DL handle, data store handle).

2.4.7 CSSM_FIELD

This structure contains the tag/data pair for a single field of a certificate or CRL.

This structure contains the object identifier (OID)/value pair for any item that can be identified by an OID. A CL module uses this structure to hold an OID/value pair for a field in a certificate or CRL.

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

Definitions:

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

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

2.4.8 CSSM_OID

The OID is used to hold an identifier for the data types and data structures that comprise the fields of a certificate or CRL. The underlying representation and meaning of the identifier is defined by the CL module. For example, a CL module can choose to represent its identifiers in any of the following forms:

- A character string in a character set native to the platform
- A DER-encoded X.509 OID that must be parsed
- An S-expression that must be evaluated
- An enumerated value that is defined in header files supplied by the CL module

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

2.4.9 CSSM_RETURN

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

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

Definitions:

CSSM_OK - Indicates operation was successful.

CSSM_FAIL - Indicates operation was unsuccessful.

2.4.10 CSSM_REVOKE_REASON

This structure represents the reason a certificate is being revoked.

```
typedef enum cssm_revoke_reason {
    CSSM_REVOKE_CUSTOM = 0,
    CSSM_REVOKE_UNSPECIFIC = 1,
    CSSM_REVOKE_KEYCOMPROMISE = 2,
    CSSM_REVOKE_CACOMPROMISE = 3,
    CSSM_REVOKE_AFFILIATIONCHANGED = 4,
    CSSM_REVOKE_SUPERCEDED = 5,
    CSSM_REVOKE_CESSATIONOFOPERATION = 6,
    CSSM_REVOKE_CERTIFICATEHOLD = 7,
    CSSM_REVOKE_CERTIFICATEHOLDRELEASE = 8,
    CSSM_REVOKE_REMOVEFROMCRL = 9
} CSSM_REVOKE_REASON;
```

2.4.11 CSSM_TP_ACTION

This data structure represents a descriptive value defined by the TP module. A TP can define application-specific actions for the application domains over which the TP applies. Given a set of credentials, the TP module verifies authorizations to perform these actions.

```
typedef uint32 CSSM_TP_ACTION
```

2.4.12 CSSM_TP_HANDLE

This data structure represents the TP module handle. The handle value is a unique pairing between a TP module and an application that has attached that module. TP handles can be returned to an application as a result of the CSSM_ModuleAttach function.

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

2.4.13 CSSM_TP_STOP_ON

This enumerated list defines the conditions controlling termination of the verification process by the TP module when a set of policies/conditions must be tested.

```
typedef enum cssm_tp_stop_on {
    CSSM_TP_STOP_ON_POLICY = 0,/* use the pre-defined stopping criteria */
    CSSM_TP_STOP_ON_NONE = 1,/* evaluate all condition whether T or F */
    CSSM_TP_STOP_ON_FIRST_PASS = 2, /* stop evaluation at first TRUE */
    CSSM_TP_STOP_ON_FIRST_FAIL = 3/* stop evaluation at first FALSE */
} CSSM_TP_STOP_ON;
```

2.5 Trust Policy Operations

2.5.1 TP_CertSign

CSSM_DATA_PTR CSSMTPI TP_CertSign (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DB_LIST_PTR DBList,
const CSSM_DATA_PTR CertToBeSigned,
const CSSM_CERTGROUP_PTR SignerCertGroup,
const CSSM_FIELD_PTR SignScope,
uint32 ScopeSize)

The TP module decides first whether the signer certificate is trusted to sign the subject certificate. Once the trust is established, the TP signs the certificate when given the signer's certificate and the *scope* of the signing process.

Parameters

TPHandle (input)
The handle that describes the TP module used to perform this function.

CLHandle (input)
The handle that describes the CL module used to perform this function.

CCHandle (input)
The cryptographic context specifies the handle of the CSP that must be used to perform the operation.

DBList (input)
A list of handle pairs specifying a DL module and a data store managed by that module. These data stores can be used to store or retrieve objects (such as certificate and CRLs) related to the signer's certificate or a data store for storing a resulting signed CRL.

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

SignerCertGroup (input)
A pointer to the CSSM_CERTGROUP structure containing one or more related certificates used to sign the certificate.

SignScope (input)
A pointer to the CSSM_FIELD array containing the tags of the certificate fields to be included in the signing process.

ScopeSize (input)
The number of entries in the sign scope list. If the signing scope is not specified, the input parameter value for scope size must be zero.

Return Values

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

See Also

CSSM_TP_CertVerify, CSSM_CL_CertSign

2.5.2 TP_CertRevoke

CSSM_DATA_PTR CSSMTPI TP_CertRevoke
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DB_LIST_PTR DBList,
const CSSM_DATA_PTR OldCrl,
const CSSM_CERTGROUP_PTR CertGroupToBeRevoked,
const CSSM_CERTGROUP_PTR RevokerCertGroup,
CSSM_REVOKE_REASON Reason)

The TP module determines whether the revoking certificate can revoke the subject certificate. The revoker certificate group is first authenticated and its applicability to perform this operation is determined. Once the trust is established, the TP revokes the subject certificate by adding it to the CRL. The revoker certificate and passphrase is used to sign the resultant CRL.

Parameters

TPHandle (input)
The handle that describes the TP module used to perform this function.

CLHandle (input)
The handle that describes the CL module that can be used to manipulate the certificates targeted for revocation and the revoker's certificates. If no CL module is specified, the TP module uses an assumed CL module, if required.

CCHandle (input)
The handle that describes the context for a cryptographic operation. The cryptographic context specifies the handle of the CSP that must be used to perform the operation.

DBList (input)
A list of certificate databases containing certificates that may be used to construct the trust structure of the subject and revoker certificate group.

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

CertGroupToBeRevoked (input)
A group of one or more certificates that partially or fully represent the certificate to be revoked by this operation. The first certificate in the group is the target certificate. The use of subsequent certificates is specific to the trust domain. For example, in a hierarchical trust model subsequent members are intermediate certificates of a certificate chain.

RevokerCertGroup (input)
A group of one or more certificates that partially or fully represent the revoking entity for this operation. The first certificate in the group is the target certificate representing the revoker. The use of subsequent certificates is specific to the trust domain.

Reason (input)
The reason for revoking the target certificates.

Return Value

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

See Also

CSSM_CL_CrlAddCert

2.5.3 TP_CrlVerify

CSSM_BOOL CSSMTPI TP_CrlVerify (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CSP_HANDLE CSPHandle,
const CSSM_DB_LIST_PTR DBList,
const CSSM_DATA_PTR CrlToBeVerified,
const CSSM_CERTGROUP_PTR SignerCertGroup,
const CSSM_FIELD_PTR VerifyScope,
uint32 ScopeSize)

This function verifies the integrity of the CRL and determines whether it is trusted. Some of the checks that may be performed include verifying the signatures on the signer's certificate group, establishing the authorization of the signer to issue CRLs, verification of the signature on the CRL, verifying validity period of the CRL and the date the CRL was issued, etc.

Parameters

TPHandle (input)
The handle that describes the TP module used to perform this function.

CLHandle (input/optional)
The handle that describes the CL module that can be used to manipulate the certificates to be verified. If no CL module is specified, the TP module uses an assumed CL module, if required.

CSPHandle (input)
The handle referencing a CSP to be used to verify signatures on the signer's certificate and on the CRL. The TP module is responsible for creating the cryptographic context structure required to perform the verification operation. If no CSP is specified, the TP module uses an assumed CSP to perform the operations.

DBList (input)
A list of handle pairs specifying a DL module and a data store managed by that module. These data stores can be used to store or retrieve objects (such as certificate and CRLs) related to the signer's certificate. If no DL and database (DB) handle pairs are specified, the TP module can use an assumed DL module and an assumed data store, if required.

CrlToBeVerified (input)
A pointer to the CSSM_DATA structure containing a signed CRL to be verified.

SignerCertGroup (input)
A group of one or more certificates that partially or fully represent the signer of the CRL. The first certificate in the group is the target certificate representing the CRL signer. Use of subsequent certificates is specific to the trust domain. For example, in a hierarchical trust model subsequent members are intermediate certificates of a certificate chain.

VerifyScope (input)
A pointer to the CSSM_FIELD array indicating the CRL fields to be included in the CRL signature verification process. A NULL input verifies the signature assuming the module's default set of fields was used in the signaturing process (this can include all fields in the CRL).

ScopeSize (input)
The number of entries in the verify scope list. If the verification scope is not specified, the input parameter value for scope size must be zero.

Return Value

A CSSM_TRUE return value means the CRL can be trusted. If CSSM_FALSE is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

See Also

CSSM_CL_CrlVerify

2.5.4 TP_CrlSign

CSSM_DATA_PTR CSSMTPI TP_CrlSign (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DB_LIST_PTR DBList,
const CSSM_DATA_PTR CrlToBeSigned,
const CSSM_CERTGROUP_PTR SignerCertGroup,
const CSSM_FIELD_PTR SignScope,
uint32 ScopeSize)

The TP module decides whether the signer certificate is trusted to sign CRL. The signer certificate group is first authenticated and its applicability to perform this operation is determined. Once the trust is established, this operation signs the CRL.

Parameters

TPHandle (input)
The handle that describes the TP module used to perform this function.

CLHandle (input)
The handle that describes the CL module used to perform this function.

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

DBList (input)
A list of handle pairs specifying a DL module and a data store managed by that module. These data stores can be used to store or retrieve objects (such as certificate and CRLs) related to the signer's certificate or a data store for storing a resulting signed CRL. If no DL and DB handle pairs are specified, the TP module can use an assumed DL module and an assumed data store, if required.

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

SignerCertGroup (input)
A group of one or more certificates that partially or fully represent the signer for this operation. The first certificate in the group is the target certificate representing the signer. Use of subsequent certificates is specific to the trust domain. For example, in a hierarchical trust model subsequent members are intermediate certificates of a certificate chain.

SignScope (input)
A pointer to the CSSM_FIELD array containing the tags of the fields to be signed. A NULL input signs a default set of fields in the CRL.

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

Return Value

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

See Also

CSSM_CL_CrlSign

2.5.5 TP_ApplyCrlToDb

CSSM_RETURN CSSMTPI TP_ApplyCrlToDb (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CSP_HANDLE CSPHandle,
const CSSM_DB_LIST_PTR DBList,
const CSSM_DATA_PTR Crl)

This function first determines whether the memory-resident CRL is trusted. The CRL is authenticated, its signer is verified, and its authority to update the data sources is determined. If trust is established, this function updates persistent storage to reflect entries in the CRL. This results in designating persistent certificates as revoked.

Parameters

TPHandle (input)
The handle that describes the TP module used to perform this function.

CLHandle (input/optional)
The handle that describes the certificate library module that can be used to manipulate the CRL as it is applied to the data store and to manipulate the certificates effected by the CRL, if required. If no certificate library module is specified, the TP module uses an assumed CL module, if required. If optional, the caller will set this value to 0.

CSPHandle (input/optional)
The handle referencing a Cryptographic Service Provider to be used to verify signatures on the CRL determining whether to trust the CRL and apply it to the data store. The TP module is responsible for creating the cryptographic context structures required for verification operation. If no CSP is specified, the TP module uses an assumed CSP to perform these operations. If optional, the caller will set this value to 0.

DBList (input/optional)
A list of handle pairs specifying a DL module and a data store managed by that module. These data stores can contain certificates that might be effected by the CRL, they may contain CRLs, or both. If no DL and DB handle pairs are specified, the TP module must use an assumed DL module and an assumed data store for this operation. If optional, the caller will set this value to NULL.

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

Return Value

A CSSM_TRUE return value means the CRL has been used to update the revocation status of certificates in the specified database. If CSSM_FALSE is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

See Also

CSSM_CL_CrlGetFirstItem, CSSM_CL_CrlGetNextItem, CSSM_DL_CertRevoke

2.5.6 TP_CertGroupConstruct

CSSM_CERTGROUP_PTR CSSMTPI TP_CertGroupConstruct
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CSP_HANDLE CSPHandle
CSSM_CERTGROUP_PTR CertGroupFrag,
CSSM_DB_LIST_PTR DBList)

This function builds a collection of certificates that together make up a meaningful credential for a given trust domain. For example, in a hierarchical trust domain, a certificate group is a chain of certificates from an end entity to a top-level CA. The constructed certificate group format (such as ordering) is implementation-specific. However, the subject or end-entity is always the first certificate in the group.

A partially constructed certificate group is specified in *CertGroupFrag*. The first certificate is interpreted to be the subject or end-entity certificate. Subsequent certificates in the *CertGroupFrag* structure may be used during the construction of a certificate group in conjunction with certificates found in *DBList*. The TP defines the certificates that will be included in the resulting set.

The constructed certificate group can be consistent locally or globally. Consistency can be limited to the local system if locally defined anchor certificates are inserted into the group.

Parameters

TPHandle (input)
The handle to the TP module to perform this operation.

CLHandle (input)
The handle to the CL module that can be used to manipulate and parse values in stored in the certgroup certificates. If no CL module is specified, the TP module uses an assumed CL module.

CSPHandle (input)
The handle referencing a CSP to be used to perform this operation.

CertGroupFrag (input)
The first certificate in the group represents the target certificate for which a group of semantically related certificates will be assembled. Subsequent intermediate certificates can be supplied by the caller. They need not be in any particular order.

DBList (input)
A list of handle pairs specifying a DL module and a data store managed by that module. These data stores should contain certificates (and possibly, other security object also). The data stores should be searched to complete construction of a semantically related certificate group.

Return Value

A list of certificates that form a complete certificate group based on the original subset of certificates and the certificate data stores. A NULL list indicates an error.

See Also

CSSM_TP_CertGroupPrune, CSSM_TP_CertGroupVerify

2.5.7 TP_CertGroupPrune

CSSM_CERTGROUP_PTR CSSMTPI TP_CertGroupPrune
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CERTGROUP_PTR OrderedCertGroup,
CSSM_DB_LIST_PTR DBList)

This function removes certificates from a certificate group. The prune operation can remove those certificates that have been signed by any local CA, as it is possible that these certificates will not be meaningful on other systems.

This operation can also remove additional certificates that can be added to the certificate group, again using the *CertGroupConstruct* operation. The pruned certificate group should be suitable for transmission to external hosts, which can in turn reconstruct and verify the certificate group.

Parameters

TPHandle (input)
The handle to the TP module used to perform this operation.

CLHandle (input/optional)
The handle to the CL module that can be used to manipulate and parse the certgroup certificates and the certificates in the specified data stores. If no CL module is specified, the TP module uses an assumed CL module.

OrderedCertGroup (input)
The initial, complete set of certificates from which certificates will be selectively removed.

DBList (input)
A list of handle pairs specifying a DL module and a data store managed by that module. These data stores should contain certificates (and possibly, other security object also). The data stores are searched for certificates semantically related to those in the certificate group to determine whether they should be removed from the certificate group.

Return Value

Returns a certificate group containing those certificates which are verifiable credentials outside of the local system. If the list is NULL, an error has occurred.

See Also

CSSM_TP_CertGroupConstruct, CSSM_TP_CertGroupVerify

2.5.8 TP_CertGroupVerify

CSSM_BOOL CSSMTPI TP_CertGroupVerify (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_DB_LIST_PTR DBList,
CSSM_CSP_HANDLE CSPHandle,
const CSSM_FIELD_PTR PolicyIdentifiers,
uint32 NumberofPolicyIdentifiers,
CSSM_TP_STOP_ON VerificationAbortOn,
const CSSM_CERTGROUP_PTR CertToBeVerified,
const CSSM_DATA_PTR AnchorCerts,
uint32 NumberofAnchorCerts,
const CSSM_FIELD_PTR VerifyScope,
uint32 ScopeSize,
CSSM_TP_ACTION Action,
const CSSM_DATA_PTR Data,
CSSM_DATA_PTR *Evidence,
uint32 *EvidenceSize)

This function verifies the signatures on each certificate in the group. Each certificate in the group has an associated signing certificate that was used to sign the subject certificate. Determination of the associated signing certificate is implied by the certificate model. For example, when verifying an X.509 certificate chain, the signing certificate for a certificate C is known to be the certificate of the issuers of certificate C. In a multisignature, web-of-trust model, the signing certificates can be any certificates in the CertGroup or unknown certificates.

Signature verification is performed on the *VerifyScope* fields for all certificates in the *CertGroup*. Additional validation tests can be performed on the certificates in the group depending on the certificate model supported by the TP. For example, certificate expiration dates can be checked and appropriate CRLs can be searched as part of the verification process.

Parameters

TPHandle (input)
The handle to the TP module to perform this operation.

CLHandle (input/optional)
The handle to the CL module that can be used to manipulate and parse the certgroup certificates and the certificates in the specified data stores. If no CL module is specified, the TP module uses an assumed CL module.

DBList (input/optional)
A list of handle pairs specifying a DL module and a data store managed by that module. These data stores should contain zero or more trusted certificates. If no data stores are specified, the TP module can assume a default data store, if required.

CSPHandle (input)
The handle referencing a CSP to be used to perform this operation.

PolicyIdentifiers (input/optional)
The policy identifier is an OID/value pair. The CSSM_OID structure contains the name of the policy and the value is an optional caller-specified input value for the TP module to use when applying the policy.

NumberofPolicyIdentifiers (input)
The number of policy identifiers provided in the *PolicyIdentifiers* parameter.

VerificationAbortOn (input/optional)
When a TP module verifies multiple conditions or multiple policies, the TP module can allow the caller to specify when to abort the verification process. If supported by the TP module, this selection can effect the evidence returned by the TP module to the caller. The default stopping condition is to stop evaluation according to the policy defined in the TP Module. The specifiable stopping conditions and their meaning are defined as follows in Table 1.

Table 1. CSSM_TP_STOP_ON Values

Value	Definition
CSSM_STOP_ON_POLICY	Stop verification whenever the policy dictates it.
CSSM_STOP_ON_NONE	Stop verification only after all conditions have been tested (ignoring the pass-fail status of each condition).
CSSM_STOP_ON_FIRST_PASS	Stop verification on the first condition that passes.
CSSM_STOP_ON_FIRST_FAIL	Stop verification on the first condition that fails.

The TP module may ignore the caller's specified stopping condition and revert to the default of stopping according to the policy embedded in the module.

CertToBeVerified (input)
A pointer to the CSSM_CERTGROUP structure containing a certificate containing at least one signature for verification. An unsigned certificate template cannot be verified.

AnchorCerts (input/optional)
A pointer to the CSSM_DATA structure containing one or more certificates to be used in order to validate the subject certificate. These certificates can be root certificates, cross-certified certificates, and certificates belonging to locally designated sources of trust.

NumberofAnchorCerts (input)
The number of anchor certificates provided in the *AnchorCerts* parameter.

VerifyScope (input/optional)
A pointer to the CSSM_FIELD array containing the OID indicators specifying the certificate fields to be used in the verification process. If VerifyScope is not specified, the TP module must assume a default scope (portions of each certificate) when performing the verification process.

ScopeSize (input)
The number of entries in the verify scope list. If the verification scope is not specified, the input scope size must be zero.

Action (input/optional)
An application-specific and application-defined action to be performed under the authority of the input certificate. If no action is specified, the TP module defines a default action and performs verification assuming that action is being requested. Note that it is possible that a TP module verifies certificates for only one action.

Data (input/optional)
A pointer to the CSSM_DATA structure containing the application-specific data or a reference to the application-specific data upon which the requested action should be performed. If no data is specified, the TP module defines one or more default data objects upon which the action or default action would be performed.

Evidence (output/optional)
A pointer to a list of CSSM_DATA objects containing an audit trail of evidence constructed by the TP module during the verification process. Typically, this is a list of certificates and CRLs that were used to establish the validity of the *CertToBeVerified*, but other objects may be appropriate for other types of trust policies.

EvidenceSize (output)
The number of entries in the *Evidence* list. The returned value is zero if no evidence is produced. *Evidence* may be produced even when verification fails. This evidence can describe why and how the operation failed to verify the subject certificate.

Return Value

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

See Also

CSSM_TP_CertGroupConstruct, CSSM_TP_CertGroupPrune

2.6 Extensibility Functions

2.6.1 TP_PassThrough

CSSM_DATA_PTR CSSMTPI TP_PassThrough (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_CC_HANDLE CCHandle,
uint32 PassThroughId,
const void * InputParams)

The TP module allows clients to call TP module-specific operations that have been exported. Such operations may include queries or services specific to the domain represented by the TP module.

Parameters

TPHandle (input)
The handle that describes the TP module used to perform this function.

CLHandle (input)
The handle that describes the CL module used to perform this function.

DLHandle (input)
The handle that describes the DL module used to perform this function.

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

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

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

InputParams (input)
A pointer to the CSSM_DATA structure containing parameters to be interpreted in a function-specific manner by the TP module.

Return Value

A pointer to the CSSM_DATA 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.

Chapter 3. Attach/Detach Example

The Trust Policy (TP) module performs certain operations when KeyWorks attaches to or detaches from it These operations should be performed in a function called AddInAuthenticate, which must be exported by the module. The AddInAuthenticate function will be called by the framework when the module is loaded. The steps in Section 3.1 must be performed in order for the attach process to work properly.

In the code in Section 3.1, 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.

3.1 AddInAuthenticate

```
#include “cssm.h”

/* global variables used for registration */
CSSM_REGISTRATION_INFOreg_info;
CSSM_SPI_TP_FUNCStp_jmp_tbl;
CSSM_SPI_MEMORY_FUNCSupcall_tbl = {NULL, NULL, NULL, NULL};
CSSM_MODULE_FUNCSmodule_funcs;
CSSM_GUID tp_guid =
{ 0x83badc39, 0xfac1, 0x11cf, { 0x81, 0x72, 0x0, 0xaa, 0x0, 0xb1, 0x99, 0xdd }
};

CSSM_RETURN CSSMAPI AddInAuthenticate(char* cssmCredentialPath, char*
cssmSection) {

       CSSM_RETURN     ret_code;

       /* first set up the TP jump table */
       TRACE(ibmtp_trace_info, FNCODE_AddInAuthenticate);

       memset(&tp_jmp_tbl, 0, sizeof(tp_jmp_tbl) );

       /* This is the only API function supported in this module */
       tp_jmp_tbl.CertGroupVerify = CertGroupVerify;

       /* set up the module specific info that CSSM needs */
       memset(&module_funcs, 0, sizeof(module_funcs));
       module_funcs.ServiceType = CSSM_SERVICE_TP;
       module_funcs.TpFuncs = &tp_jmp_tbl;

       /* ok, now set up the registration structure that CSSM understands */
       memset(&reg_info, 0, sizeof(reg_info) );

       reg_info.Initialize= Initialize;
       reg_info.Terminate=  Uninitialize;
       reg_info.ThreadSafe= CSSM_FALSE;
       reg_info.ServiceSummary = CSSM_SERVICE_TP;
       reg_info.NumberOfServiceTables = 1;
       reg_info.Services= &module_funcs;

       /* Register services with CSSM */
       ret_code = CSSM_RegisterServices(&tp_guid, &reg_info, &upcall_tbl,
       NULL);

       return ret_code;

}
```

Appendix A. IBM KeyWorks Errors

The error codes given in this section constitute the generic error codes that are defined by KeyWorks for use by all Certificate Libraries (CLs) in describing common error conditions. A CL may also define and return vendor-specific error codes. The error codes defined by KeyWorks are considered to be comprehensive and few if any vendor-specific codes should be required. Applications must consult vendor-supplied documentation for the specification and description of any error codes defined outside of this specification.

All Trust Policy service provider interface (TP SPI) functions return one of the following:

- 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 2 lists the range of error numbers defined by KeyWorks for TP modules and those available for use individual Cryptographic Service Provider (CSP) modules.

Table 2. Trust Policy Module Error Numbers

Error Number Range	Description
7000 – 7999	TP errors defined by KeyWorks
8000 – 8999	TP errors reserved for individual TP modules

The calling application must determine how to handle the error returned by CSSM_GetError. Detailed descriptions of the error values will be available in the corresponding specification, the cssmerr.h header file, and the documentation for specific modules. If a routine does not know how to handle the error, it may choose to pass the error to its caller.

A.1.Trust Policy Module Errors

Table 3. Trust Policy Errors

Error Code	Error Name
7001	CSSM_TP_NOT_LOADED
7002	CSSM_TP_INVALID_TP_HANDLE
7003	CSSM_TP_INVALID_CL_HANDLE
7004	CSSM_TP_INVALID_DL_HANDLE
7005	CSSM_TP_INVALID_DB_HANDLE
7006	CSSM_TP_INVALID_CC_HANDLE
7007	CSSM_TP_INVALID_CERTIFICATE
7008	CSSM_TP_NOT_SIGNER
7009	CSSM_TP_NOT_TRUSTED
7010	CSSM_TP_CERT_VERIFY_FAIL
7011	CSSM_TP_CERTIFICATE_CANT_OPERATE
7012	CSSM_TP_MEMORY_ERROR
7013	CSSM_TP_CERT_SIGN_FAIL
7014	CSSM_TP_INVALID_CRL
7015	CSSM_TP_CERT_REVOKE_FAIL
7016	CSSM_TP_CRL_VERIFY_FAIL
7017	CSSM_TP_CRL_SIGN_FAIL
7018	CSSM_TP_APPLY_CRL_TO_DB_FAIL
7019	CSSM_TP_INVALID_GUID
7020	CSSM_TP_UNISTALL_FAIL
7021	CSSM_TP_INCOMPATIBLE_VERSION
7022	CSSM_TP_INVALID_ACTION
7023	CSSM_TP_VERIFY_ACTION_FAIL
7024	CSSM_TP_INVALID_DATA_POINTER
7025	CSSM_TP_INVALID_ID
7026	CSSM_TP_PASS_THROUGH_FAIL
7027	CSSM_TP_INVALID_CSP_HANDLE
7028	CSSM_TP_ANCHOR_NOT_SELF_SIGNED
7029	CSSM_TP_ANCHOR_NOT_FOUND

Appendix B. List of Acronyms

API	Application Programming Interface
CA	Certificate Authority
CL	Certificate Library
CRL	Certificate Revocation List
CSP	Cryptographic Service Provider
DB	Database
DL	Data Storage Library
DLL	Dynamically Linked Library
GUID	Globally Unique ID
ISV	Independent Software Vendor
KRF	Key Recovery Field
KRSP	Key Recovery Service Provider
OID	Object Identifier
SPI	Service Provider Interface
TP	Trust Policy

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 text 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 a 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, 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 the 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 the 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 either by a KRA to recover the original symmetric key, 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 IBM KeyWorks. It mediates all cryptographic services provided by the IBM 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 IBM 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 recovery 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 Key Recovery Block (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 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 the 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.