Key Recovery Service Provider

Key Recovery Service Provider Interface (KRSPI) Specification

Table of Contents

List of Figures

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.

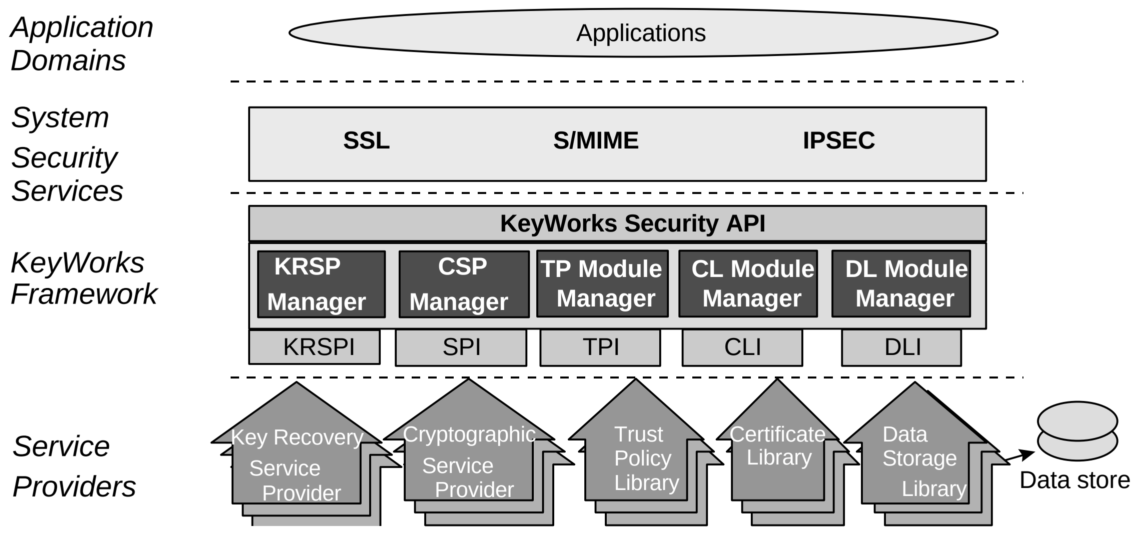

Figure 1. IBM KeyWorks Toolkit Architecture

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 key recovery 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. Key Recovery Overview

Key recovery mechanisms serve many useful purposes; they may be used by individuals to recover lost or corrupted keys, or they may be used by enterprises to deter corporate insiders from using encryption to bypass the corporate security policy regarding the flow of proprietary information. Corporations may also use key recovery mechanisms to recover employee keys in certain situations, e.g., in the employee's absence. The use of key recovery mechanisms in web-based transactional scenarios can serve as an additional technique of non-repudiation and audit that may be admissible in a court of law. Finally, key recovery mechanisms may be used by jurisdictional law enforcement bodies to access the contents of confidentiality-protected communications and stored data. Thus, there appears to be multiple incentives for the incorporation as well as adoption of key recovery mechanisms in local and distributed encryption-based systems.

Denning and Brandstad [Key Escrow 1996] present a taxonomy of key escrow systems. A different scheme of nomenclature was adopted in order to exhibit some of the finer nuances of key recovery schemes. The term *key recovery* encompasses mechanisms that allow authorized parties to retrieve the cryptographic keys used for data confidentiality, with the ultimate goal of recovery of encrypted data. The remainder of this section will discuss the various types of key recovery mechanisms, the phases of key recovery, and the policies with respect to key recovery.

2.1 Key Recovery Types

There are two classes of key recovery mechanisms based on the way keys are held to enable key recovery:

- Key Escrow **-** Techniques based on the paradigm that the government or a trusted party, called an *escrow agent*, holds the actual user keys or portions thereof.
- Key Encapsulation **-** Techniques based on the paradigm that a cryptographically encapsulated form of the key is made available to parties that require key recovery. The technique ensures that only certain trusted third parties called, *Key Recovery Agents (KRAs)*, can perform the unwrap operation to retrieve the key material buried inside.

There may also be hybrid schemes that use some escrow mechanisms in addition to encapsulation mechanisms.

An orthogonal way to classify key recovery mechanisms is based on the nature of the key:

- Long-term private keys
- Ephemeral keys

Both types of keys can be escrowed or encapsulated. Since escrow schemes involve the actual archival of keys, they typically deal with long-term keys in order to avoid the proliferation problem that arises when trying to archive the myriad ephemeral keys. Key encapsulation techniques, on the other hand, usually operate on the ephemeral keys.

For a large class of key recovery (escrow as well as encapsulation) schemes, there is a set of *Key Recovery Fields (KRFs)* that accompanies an enciphered message or file. These KRFs may be used by the appropriate authorized parties to recover the decryption key and or the plaintext. Typically, the KRFs comprise information regarding the key escrow or KRAs that can perform the recovery operation; they also contain other pieces of information to enable recovery.

In a key escrow scheme for long-term private keys, the *escrowed* keys are used to recover the ephemeral data confidentiality keys. In such a scheme, the KRFs may comprise the identity of the escrow agents, identifying information for the escrowed key, and the bulk encryption key which is wrapped in the recipient's public key (part of an escrowed key pair); thus the KRFs include the key exchange block in this case. In a key escrow scheme where bulk encryption keys are archived, the KRFs may comprise information to identify the escrow agents and the escrowed key for that enciphered message.

In a typical key encapsulation scheme for ephemeral bulk encryption keys, the KRFs are distinct from the key exchange block (if any). The KRFs identify the KRAs and contain the bulk encryption key encapsulated using the public keys of the KRAs.

To ensure the integrity of the KRFs, and its association with the encrypted data, it may be required for processing by the party performing the data decryption. The processing mechanism ensures that successful data decryption cannot occur unless the integrity of the KRFs is maintained at the receiving end. In schemes where the KRFs contain the key exchange block, decryption cannot occur at the receiving end unless the KRFs are processed to obtain the decryption key; thus the integrity of the KRFs are automatically verified. In schemes where the KRFs are separate from the key exchange block, additional processing must be done to ensure that decryption of the ciphertext occurs only after the integrity of the KRFs are verified.

2.2 Lifetime of Key Recovery Fields

Cryptographic products fall into one of two fundamental classes: *archived-ciphertext products* and *transient-ciphertext products*. When the product allows either the generator or the receiver of ciphertext to archive the ciphertext, the product is classified as an archived-ciphertext product. On the other hand, when the product does not allow the generator or receiver of ciphertext to archive the ciphertext, it is classified as a transient-ciphertext product.

It is important to note that the lifetime of KRFs should never be greater than the lifetime of the associated ciphertext. This is somewhat obvious, since recovery of the key is only meaningful if the key can be used to recover the plaintext from the ciphertext. Hence, when archived-ciphertext products are key recovery enabled, the KRFs are typically archived as long as the ciphertext. Similarly, when transient-ciphertext products are key recovery enabled, the KRFs are associated with the ciphertext for the duration of its lifetime. It is not meaningful to archive KRFs without archiving the associated ciphertext.

2.3 Key Recovery Policy

Key recovery policies are mandatory policies that may be derived from enterprise-based or jurisdiction-based rules on the use of cryptographic products for data confidentiality. 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 domains, and may mandate key recovery policies on the cryptographic products within their own domain.

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 (i.e., 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.

2.4 Operational Scenarios for Key Recovery

There are three basic operational scenarios for key recovery, including:

- Enterprise key recovery
- Law Enforcement key recovery
- Individual key recovery

Enterprise key recovery allows enterprises to enforce stricter monitoring of the use of cryptography, and the recovery of enciphered data when the need arises. Enterprise key recovery is based on a mandatory key recovery policy; however, this policy is set (typically through administrative means) by the organization or enterprise at the time of installation of a recovery enabled cryptographic product. The enterprise key recovery policy should not be modifiable or bypassable by the individual using the cryptographic product. Enterprise key recovery mechanisms may use special, enterprise-authorized key escrow or KRAs.

In the law enforcement scenario, key recovery is mandated by the jurisdictional law enforcement authorities in the interest of national security and law enforcement. For a specific cryptographic product, the key recovery policies for multiple jurisdictions may apply simultaneously. The policies (if any) of the jurisdictions of manufacture of the product, as well as the jurisdiction of installation and use, need to be applied to the product such that the most restrictive combination of the multiple policies is used. Thus, law enforcement key recovery is based on mandatory key recovery policies; these policies are logically bound to the cryptographic product at the time the product is shipped. There may be some mechanism for vendor-controlled updates of such law enforcement key recovery policies in existing products; however, organizations and end users of the product are unable to modify this policy at their discretion. The key escrow or KRAs used for this scenario of key recovery need to be strictly controlled, in most cases, to ensure that these agents meet the eligibility criteria for the relevant political jurisdiction where the product is being used.

Individual key recovery is user-discretionary in nature and is performed for the purpose of recovery of enciphered data by the owner of the data, if the cryptographic keys are lost or corrupted. Since this is a nonmandatory key recovery scenario, it is not based on any policy that is enforced by the cryptographic product; rather, the product may allow the user to specify when individual key recovery enablement will be performed. There are few restrictions on the use of specific key escrow or KRAs.

Key recovery enabled cryptographic products must be designed so that the key recovery enablement operation is mandatory and noncircumventable in the law enforcement and enterprise scenarios, and discretionary for the individual scenario. The key escrow and KRAs that are used for law enforcement and enterprise scenarios must be tightly controlled so that the agents are validated to belong to a set of authorized or approved agents. In the law enforcement and enterprise scenarios, the key recovery process typically needs to be performed without the knowledge and cooperation of the parties involved in the cryptographic association.

The components of the KRFs also vary somewhat between the three scenarios. In the law enforcement scenario, the KRFs must contain identification information for the key escrow or KRAs, whereas for the enterprise and individual scenarios, the agent identification information is not so critical, since this information may be available from the context of the recovery enablement operation. For the individual scenario, there needs to be a strong user authentication component in the KRFs to allow the owner of the KRFs to authenticate themselves to the agents; however, for the enterprise and law enforcement scenarios, the authorization credentials checked by the agents may be in the form of legal documents or enterprise authorization documents for key recovery, which may not be tied to any authentication component in the KRFs. For the law enforcement and enterprise scenarios, the KRFs may contain recovery information for both the generator and receiver of the enciphered data; in the individual scenario, only the information of the generator of the enciphered data is typically included (at the discretion of the generating party).

2.5 Key Recovery Phases

The process of cryptographic key recovery involves three major phases. The first phase is an optional *key recovery registration* phase in which the parties that desire key recovery performs some initialization operations with the key escrow or KRAs. These operations include obtaining a user public key certificate (for an escrowed key pair) from an escrow agent, or obtaining a public key certificate from a KRA. In phase two, parties that are involved in cryptographic associations have to perform operations to enable key recovery (such as the generation of KRFs, etc.); this is typically called the *key recovery enablement* phase. In the third and final phase, authorized parties that desire to recover the data keys do so with the help of a Key Recovery Server (KRS) and one or more escrow agents or KRAs; this is the *key recovery request* phase.

Figure 2 illustrates the three phases of key recovery. In Figure 2(a), a key recovery client registers with a KRA prior to engaging in cryptographic communication. In Figure 2(b), two key recovery enabled cryptographic applications are communicating using a key encapsulation mechanism. The KRFs are passed along with the ciphertext and key exchange block to enable subsequent key recovery. The key recovery request phase is illustrated in Figure 2(c), where the KRFs are provided as input to the KRS along with the authorization credentials of the client requesting service. The KRS interacts with one or more local or remote KRAs to reconstruct the secret key that can be used to decrypt the ciphertext.

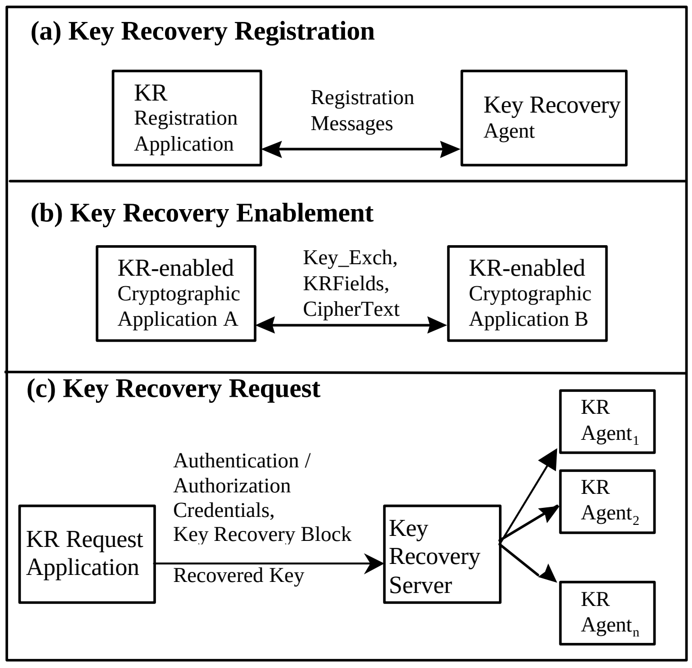
(a) Key Recovery Registration
KR Registration Application
Registration Messages
Key Recovery Agent
(b) Key Recovery Enablement
KR-enabled Cryptographic Application A
Key_Exch, KRFields, CipherText
KR-enabled Cryptographic Application B
(c) Key Recovery Request
KR Request Application
Authentication / Authorization Credentials, Key Recovery Block
Recovered Key
Key Recovery Server
KR Agent$_1$
KR Agent$_2$
KR Agent$_n$

Figure 2. Key Recovery Phases

It is likely that governments or organizations will operate their own KRS hosts independently, and that KRSs may support a single or multiple key recovery mechanisms. There are a number of important issues specific to the implementation and operation of the KRSs, such as vulnerability and liability. The focus of this document is a framework-based approach to implementing the key recovery operations pertinent to end parties that use encryption for data confidentiality. The issues with respect to the KRS and agents will not be discussed further in this section.

2.6 Key Recovery Module Manager

The Key Recovery Module Manager is responsible for handling the key recovery application programming interface (API) functions and invocation of the appropriate Key Recovery Service Provider Interface (KRSPI) functions. The Key Recovery Module Manager 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.

2.6.1 Operational Scenarios

The IBM KeyWorks architecture supports three distinct operational scenarios for key recovery, namely, key recovery for law enforcement purposes, enterprise purposes, and individual purposes. The law enforcement and enterprise scenarios for key recovery are mandatory in nature, thus the KeyWorks layer code enforces the key recovery policy with respect to these scenarios through the appropriate sequencing of key recovery API and cryptographic API calls. On the other hand, the individual scenario for key recovery is completely discretionary, and is not enforced by the KeyWorks layer code. The application/user requests key recovery operations using the key recovery APIs at their discretion.

The three operational scenarios for key recovery enablement drive certain design decisions with respect to the KeyWorks. The details of the specific features of the operational scenarios are described in the following subsections.

2.6.2 Key Recovery Profiles

The KRSPs require certain pieces of information related to the parties involved in a cryptographic association in order to generate and process KRFs. These pieces of information (such as the public key certificates of the KRAs) are contained in *key recovery profiles*. A key recovery profile contains all of the
per-user parameters for KRF generation and processing for a specific KRSP. In other words, each user has a distinct profile for each KRSP.

The information contained in the profile comprises the following:

- User identity
- Public key certificate chain for the user
- Set of KRA certificate chains for enterprise key recovery
- Set of KRA certificate chains for law enforcement key recovery
- Set of KRA certificate chains for individual key recovery
- Authentication Information (AI) field for individual key recovery
- Set of key recovery flags that fine tune the behavior of a KRSP
- Extension field

The key recovery profiles support a list of KRA certificate chains for each of the law enforcement, enterprise, and individual key recovery scenarios, respectively. While the profile allows full certificate chains to be specified for the KRAs, it also supports the specification of leaf certificates; in such instances, the KRSP and the appropriate Trust Policy (TP) modules are expected to dynamically discover the intermediate Certificate Authority (CA) certificates up to the root certificate of trust. One or more of these certificate chains may be set to NULL, if they are not needed or supported by the KRSP involved.

The user public key certificate chain is also part of a profile. This is a necessary parameter for certain key escrow and encapsulation schemes. Similarly, certain schemes support the notion of an authentication field for individual key recovery. This field is used by the KRS and/or KRAs to verify the authorization of the individual requesting the key. One or more of fields may be set to NULL if their use is not required or supported by the KRSP involved.

The key recovery flags are defined values that are pertinent for a large class of escrow and recovery schemes. The extension field is for use by the KRSPs to define additional semantics for the key recovery profile. These extensions may be flag parameters or value parameters. The semantics of these extensions are defined by a KRSP; the application that uses profile extensions has to be cognizant of the specific extensions for a particular KRSP. However, it is envisioned that these extensions will be for optional use only. KRSPs are expected to have reasonable defaults for all such extensions; this is to ensure that applications do not need to be aware of specific KRSP profile extensions in order to get basic key recovery enablement services from a KRSP. Whenever the extension field is set to NULL, the defaults should be used by a KRSP.

2.6.3 Key Recovery Context

All operations performed by the KRSPs are performed within a *key recovery context*. A key recovery context is programmatically equivalent to a cryptographic context; however, the attributes of a key recovery context are different from those of other cryptographic contexts. There are three kinds of key recovery contexts: registration contexts, enablement contexts, and recovery request contexts. A key recovery context contains state information that is necessary to perform key recovery operations. When the key recovery API functions are invoked by application layer code, the Key Recovery Module Manager passes the appropriate key recovery context to the KRSP using the key recovery SPI function parameters.

A key recovery registration context contains no special attributes. A key recovery enablement context maintains information about the profiles of the local and remote parties for a cryptographic association. When the key recovery API function to create a key recovery enablement context is invoked, the key recovery profiles for the specified communicating peers are specified by the application layer code using the API parameters. A key recovery request context maintains a set of KRFs which are being used to perform a recovery request operation, and a set of flags that denotes the operational scenario of the recovery request operation. Since the establishment of a context implies the maintaining of state information within the KeyWorks, contexts acquired should be released as soon as their need is over.

2.6.4 Key Recovery Policy

The KeyWorks enforces the applicable key recovery policy on all cryptographic operations. There are two key recovery policies enforced by the KeyWorks: a law enforcement key recovery policy and the enterprise key recovery policy. Since the requirements for these two mandatory key recovery scenarios are somewhat different, they are implemented by different mechanisms within the KeyWorks.

The law enforcement key recovery policy is predefined (based on the political jurisdictions of manufacture and use of the cryptographic product) for a given product. The parameters on which the policy decision is made are predefined as well. Thus, the law enforcement key recovery policy is implemented using two Key Recovery Policy Tables (KRPTs); one table corresponding to the policy of the jurisdiction of manufacture, and the second corresponding to the jurisdiction of use of the product. These two law enforcement policy tables are consulted by the key recovery policy enforcement function in the KeyWorks. The law enforcement policy tables are implemented as two separate physical files for ease of implementation and upgrade (as law enforcement policies evolve over time); however, these files are protected using the same integrity mechanisms as the framework module, and thus has the same assurance properties.

The enterprise key recovery policy could vary anywhere between being set to NULL and being very complex (e.g., based on parameters such as time of day). Enterprises are allowed total flexibility with respect to the enterprise key recovery policy. The enterprise policy is implemented within the KeyWorks by invoking a key recovery policy function that is defined by the enterprise administrator. The key recovery API provides a function that allows an administrator to specify the name of a file that contains the enterprise key recovery policy function. The first time this function is used, the administrator can establish a passphrase for subsequent calls on this function. This mechanism assures a level of access control on the enterprise policy, once a policy function has been established. It goes without saying that the file containing the policy function should be protected using the maximal possible protection afforded by the operating system platform. The actual structure of the policy function file is operating system platform-specific.

Every time a cryptographic context handle is returned to application layer code, the KeyWorks enforces the law enforcement and enterprise key recovery policies. For the law enforcement policy, the KeyWorks policy enforcement function and the law enforcement policy table are used. For the enterprise policy, the enterprise policy function file is invoked in an operating system platform-specific way. If the policy check determines that key recovery enablement is required for either law enforcement or enterprise scenarios, then the context is flagged as unusable, otherwise the context is flagged as usable. An unusable context handle becomes flagged as usable only after the appropriate key recovery enablement operation is completed using that context handle. A usable context handle can then be used to perform cryptographic operations.

2.6.5 Key Recovery Enablement Operations

The KeyWorks key recovery enablement operations comprise the generation and processing of KRFs. Within a cryptographic association, KRF generation is performed by the sending side. KRF processing is performed on the receiving side to ensure that the integrity of the KRFs have been maintained in transmission between the sending and receiving sides. These two vital operations are performed via the KR_GenerateRecoveryFields and the KR_ProcessRecoveryFields functions, respectively. These functions are discussed in Chapter 3.

The KRFs generated by the KeyWorks potentially comprises three subfields: law enforcement, enterprise, and individual key recovery scenarios, respectively. The law enforcement and enterprise key recovery subfields are generated when the law enforcement and enterprise usability flags are appropriately set in the cryptographic context used to generate the KRFs. The individual key recovery subfields are generated when a certain flag value is set while invocation of the API function to generate the KRFs. The processing of the KRFs only applies to the law enforcement and enterprise key recovery subfields; the individual key recovery subfields are ignored by the KRFs processing function.

2.6.6 Key Recovery Registration and Request Operations

The KeyWorks also supports the operations of registration and recovery requests. The KRSP exchanges messages with the appropriate KRA and KRS to obtain the results required. If additional inputs are required for the completion of the operation, the supplied callback may be used by the KRS. The recovery request operation can be used to request of batch of recoverable keys. The result of the registration operation is a key recovery profile data structure, while the results of a recovery request operation are a set of recovered keys.

2.7 Extensions to the Cryptographic Module Manager

The Cryptographic Module Manager of the IBM KeyWorks is responsible for handling the cryptographic functions of the KeyWorks. In order to introduce the necessary dependencies between the cryptographic operations and the key recovery enablement operations, the Cryptographic Module Manager is extended with conditional behavior.

The cryptographic context data structure, which holds the many parameters that must be specified as input to a cryptographic function, has been augmented to include the following key recovery extension fields:

- Usability field for key recovery
- Workfactor field for law enforcement key recovery

The usability field denotes whether a cryptographic context needs to have key recovery enablement operations (either for law enforcement or enterprise needs) performed before it can be used for cryptographic operations such as encryption or decryption. The workfactor field holds the allowable workfactor value for law enforcement key recovery. These two additional fields of the cryptographic context are not available to the API for modification. They are set by the Key Recovery Module Manager when the latter makes the key recovery policy enforcement decision for law enforcement and enterprise policies.

The behavior of some of the cryptographic functions of the KeyWorks API are designed to accommodate the above mentioned extensions to the cryptographic context, as follows:

- Invoke key recovery policy enforcement functions for cryptographic context creation and update operations.
- Set the usability field in the cryptographic context to render the context unusable if key recovery enablement operations are mandated.
- Check the cryptographic context usability field before allowing encryption/decryption operations to occur.

Whenever a cryptographic context is created or updated using the KeyWorks API functions, the Cryptographic Module Manager invokes a Key Recovery Module Manager policy enforcement function module; the latter checks the law enforcement and enterprise policies to determine whether the cryptographic context defines an operation where key recovery is mandated. If so, the usability field value is set in the cryptographic context data structure to signify that the context is unusable until key recovery enablement operations are performed on this context. The usability field is essentially a bitmap that signifies whether key recovery is required by the law enforcement or enterprise key recovery policies. When the appropriate key recovery enablement operations are performed on this context, the bits in the usability field are appropriately toggled so that the cryptographic context becomes usable for the intended operations.

When the encryption/decryption operations are invoked through the KeyWorks API, and the Key Recovery Module Manager is present in KeyWorks, the Cryptographic Module Manager checks the key recovery usability field in the cryptographic context to determine whether the context is usable for encryption/decryption operations. If the context is flagged as unusable, the Cryptographic Module Manager does not dispatch the call to the Cryptographic Service Provider (CSP) and returns an error to the caller. When the appropriate key recovery enablement operations are performed on that context, the Key Recovery Module Manager resets the context flags making that context usable for encryption/decryption.

Chapter 3. Key Recovery Service Provider Interface

The generic KeyWorks module management functions are used to install and attach a Key Recovery Service (KRS) module. These functions are specified in detail in the *IBM KeyWorks Application Programming Interface Specification*. The applicable generic management functions include the following:

- CSSM_ModuleInstall
- CSSM_ModuleUninstall
- CSSM_ListModules
- CSSM_ModuleAttach
- CSSM_ModuleDetach
- CSSM_GetModuleInfo
- CSSM_FreeModuleInfo

The new management function, CSSM_KR_SetEnterpriseRecoveryPolicy**,** is directly supported by the Key Recovery Module Manager in the IBM KeyWorks Framework.

CSSM_KR_SetEnterpriseRecoveryPolicy
This call establishes the filename that contains the enterprise-based key recovery policy function for use by the Key Recovery Module Manager in KeyWorks.

3.1 Key Recovery Registration Functions

KRSP_RegistrationRequest
Performs a recovery registration request operation. A callback may be supplied to allow the registration operation to query for additional input information, if necessary. The result of the registration request operation is a reference handle that may be used to invoke the KRSP_RegistrationRetrieve function.

KRSP_RegistrationRetrieve
Completes a recovery registration operation. The result of the registration operation is returned in the form of a key recovery profile.

3.2 Key Recovery Enablement Functions

KRSP_GenerateKRFields
Accepts as input the key recovery context handle, the session-based recovery parameters, the cryptographic context handle, and several other parameters of relevance to the Key Recovery Service Provider (KRSP). Outputs a buffer of the appropriate mechanism-specific Key Recovery Fields (KRFs) in a format defined and interpreted by the specific KRSP involved. On successful completion, the input cryptographic context handle may now be used for the encryption APIs in the cryptographic framework.

KRSP_ProcessKRFields
Accepts as input the key recovery context handle, cryptographic context handle, several other parameters of relevance to a KRSP, and the unparsed buffer of KRFs. On successful return, the input cryptographic context handle can be used for the decryption APIs in the cryptographic framework.

3.3 Key Recovery Request Functions

KRSP_RecoveryRequest

Performs a recovery request operation for one or more recoverable keys. A callback may be supplied to allow the recovery request operation to query for additional input information, if necessary. The result of the recovery request operation is a results handle that may be used to obtain each recovered key and its associated meta information using the KRSP_GetRecoveredObject function.

KRSP_RecoveryRetrieve

Completes a recovery request operation for one or more recoverable keys. The result of the recovery operation is that the results handle may be used to obtain each recovered key and its meta information using the KRSP_GetRecoveredObject function.

KRSP_GetRecoveredObject

Retrieves a single recovered key and its associated meta information.

KRSP_RecoveryRequestAbort

Terminates a recovery request operation and releases any state information associated with it.

3.4 Privilege Functions

KR_PassPrivFunc

This function supports a privileged mode of the KRSP with respect to the policies enforced by the framework.

3.5 Data Structures

This section describes the data structures that may be passed to or returned from a KRSP function. Applications use these data structures to prepare and then pass input parameters into KeyWorks API function calls, which are passed without modification to the appropriate key recovery. The key recovery is responsible for interpreting them and returning the appropriate data structure to the calling application via the KeyWorks Framework. These data structures are defined in the header file, cssmtype.h, which is distributed with the IBM KeyWorks Toolkit.

3.5.1 CSSM_BOOL

```
typedef uint32 CSSM_BOOL;

#define CSSM_TRUE   1
#define CSSM_FALSE  0
```

3.5.2 CSSM_CERTGROUP

This data structure contains a set of certificates that are based on cosignaturing. This certificate group is a syntactic representation of a trust model.

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

Definitions:

NumCerts **-** Count of the number of certificates in the list.

CertList **-** Pointer to a list of certificate items.

Reserved **-** Reserved for future use.

3.5.3 CSSM_CONTEXT_ATTRIBUTE

The key recovery context creation operations return key recovery context handles that are represented as cryptographic context handles. In order to use the CSSM_CONTEXT data structure to implement key recovery contexts, the CSSM_CONTEXT will be used to hold new types of attributes, as shown below:

```
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;
```

All but the last member of the union above are part of the core *IBM KeyWorks Application Programming Interface Specification*. The descriptions of these basic fields and members are in the KeyWorks API document. The *KRProfile* member of the union has been added specifically to support key recovery contexts, which is described below.

Definitions:

AttributeType **-** One of the defined CSSM_ATTIBUTE_TYPE values.

AttributeLength **-** The length of the attribute.

KRProfile **-** A pointer to the key recovery profile structure that defines the user parameters with respect to the key recovery process.

Several new attribute types were defined to support the key recovery context attributes. The following definitions are added to the enumerated type CSSM_ATTRIBUTE_TYPE:

```
    CSSM_ATTRIBUTE_KRPROFILE_LOCAL  = (CSSM_ATTRIBUTE_LAST + 1),
    CSSM_ATTRIBUTE_KRPROFILE_REMOTE = (CSSM_ATTRIBUTE_LAST + 2)
```

3.5.4 CSSM_KRC_HANDLE

This data structure represents the key recovery context handle.

```
typedef uint32 CSSM_KRC_HANDle
```

3.5.5 CSSM_KR_NAME

This data structure contains a typed name. The namespace type specifies what kind of name is contained in the third parameter.

```
typedef struct cssm_kr_name {
      uint8 Type; /* namespace type */
      uint8 Length; /* name string length */
      char *Name; /* name string */
} CSSM_KR_NAME *CSSM_KR_NAME_PTR;
```

Definitions:

Type **-** The type of the key recovery name space.

Length **-** The length of the name (in bytes).

Name **-** The name represented in a string.

3.5.6 CSSM_KR_PROFILE

This data structure encapsulates the key recovery profile for a given user and a given key recovery mechanism.

```
typedef struct cssm_kr_profile {
    CSSM_KR_NAME            UserName;
    CSSM_DATA_PTR           UserCertificate;
    uint8                   LE_KRANum;
    CSSM_CERTGROUP_PTR      LE_KRACertChainList;
    uint8                   enterprise_KRANum;
    CSSM_CERTGROUP_PTR      ENT_KRACertChainList;
    uint8                   INDIV_KRANum;
    CSSM_CERTGROUP_PTR      INDIV_KRACertChainList;
    CSSM_DATA_PTR           INDIV_AuthenticationInfo;
    uint32                  KRSPFlags;
    CSSM_DATA_PTR           KRSPExtensions;
} CSSM_KR_PROFILE, *CSSM_KR_PROFILE_PTR;
```

Definitions:

UserName **-** Name of the entity using this profile.

*UserCertificat*e **-** The X.509 Version 3 ASN.1 DER-encoded public key certificate of the user. It is used for identity and authentication when performing policy evaluation.

LE_KRANum - The number of law enforcement KRAs in LE_KRACertChainList.

LE_KRACertChainList **-** A list of certificates chains, one per KRA, authorized for law enforcement key recovery.

ENT_KRANum - The number of enterprise KRAs in ENT_KRACertChainList.

ENT_KRACertChainList - A list of certificates chains, one per KRA, authorized for enterprise key recovery.

INDIV_KRANum - The number of individual KRAs in LE_KRACertChainList.

INDIV_KRACertChainList - A list of certificates chains, one per KRA, authorized for individual key recovery.

INDIVAuthenticationInfo - Authentication Information (AI) to be used for individual key recovery.

KRSPFlags - A bit-mask specifying the user's selected service options specific to the selected KRS module.

KRSPExtensions - Reserved for future use.

3.5.7 CSSM_KRSP_HANDLE

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

```
typedef uint32 CSSM_KRSP_HANDLE
```

3.5.8 CSSMKRSPI

```
typedef CSSMAPI CSSMKRSPI
```

3.5.9 CSSM_PRIV_FUNC_PTR

This defines the prototype for the privileged entry point to the IBM KeyWorks Framework that allows the toggling of the privileged mode of a specified cryptographic context.

```
typedef CSSM_RETURN  (* CSSM_PRIV_FUNC_PTR) (
    CSSM_CC_HANDLE hContext,
    CSSM_BOOL    Priv)
```

Definitions:

hContext - The context whose privilege mode will be set/reset.

Priv - The Boolean that denotes whether the context will be made privileged or regular. If Priv is TRUE, the hContext will be made privileged.

3.5.10 CSSM_KRSPINFO

Two structures are used to contain all of the static information that describes a key recovery module: the *krspinfo* structure and *the krspsubservice* structure. This descriptive information is securely stored in the KeyWorks registry when the key recovery module is installed with the KeyWorks Framework. A key recovery module may implement multiple types of services and organize them as subservices. For example, a key recovery module supporting two mechanisms: an encapsulation mechanism and an escrow mechanism, may organize its implementation as two subservices.

The descriptive information stored in these structures can be queried using the function CSSM_GetModuleInfo and specifying the key recovery module Globally Unique ID (GUID).

```
typedef struct cssm_krspinfo {
    CSSM_VERSION Version;
    char    *Vendor;
    char    *Description;
    char    *Jurisdiction;
    CSSM_BOOL ThreadSafe;
    uint32 NumberOfSubServices;
    CSSM_KRSPSUBSERVICE_PTR SubServices;
} CSSM_KRSPINFO, *CSSM_KRSPINFO_PTR;
```

Definitions:

Version **-** The major and minor version number of the service provider module.

Vendor **-** A character string containing the name of the vendor who implemented and manufactured the key recovery module.

Description **-** A character string containing a general description of the key recovery module.

Jurisdiction **-** A character string describing the geographical region where the key recovery module is installed.

NumberOfSubservices **-** The number of subservices implemented by the key recovery module. Every key recovery module implements at least one subservice.

Threadsafe **-** A Boolean that indicates whether the KRSP module is enabled for multitasking. This field should be set to either CSSM_TRUE or CSSM_FALSE.

Subservices **-** A pointer to an array of subservice structures. Each structure contains detailed information about that subservice.

3.5.11 CSSM_KRSPSUBSERVICE

Two structures are used to contain all of the static information that describes a key recovery module: the *krspinfo* structure and the *krspsubservice* structure. This descriptive information is securely stored in the KeyWorks registry when the key recovery module is installed with the KeyWorks Framework. A key recovery module may implement multiple types of services and organize them as subservices. For example, a key recovery module supporting two mechanisms: an encapsulation mechanism and an escrow mechanism, may organize its implementation as two subservices.

The descriptive information stored in these structures can be queried using the function CSSM_GetModuleInfo and specifying the key recovery module GUID.

```
typedef struct cssm_krspsubservice {
    uint32 SubServiceId;
    CSSM_STRING Description;
    CSSM_STRING Jurisdiction;
} CSSM_KRSPSUBSERVICE, *CSSM_KRSPSUBSERVICE_PTR;
```

Definitions:

SubServiceId **-** A unique, identifying number for the subservice described in this structure.

Description **-** A string containing a descriptive name or title for this subservice.

Jurisdiction - A character string describing the geographical region where the key recovery module is installed.

3.5.12 CSSM_KR_WRAPPEDPRODUCTINFO

```
typedef struct cssm_kr_wrappedproductinfo {
    CSSM_VERSION StandardVersion;/
    CSSM_STRING  StandardDescription;
    CSSM_VERSION ProductVersion;
    CSSM_STRING ProductDescription;
    CSSM_STRING ProductVendor;
    uint32 ProductFlags;
} CSSM_KR_WRAPPEDPRODUCT_INFO, *CSSM_KR_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 KRSP.

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

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

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

3.5.13 CSSM_RETURN

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

3.6 Key Recovery Registration Functions

3.6.1 KRSP_RegistrationRequest

CSSM_RETURN CSSMKRSPI KRSP_RegistrationRequest

```
(CSSM_KRSP_HANDLE  KRSPHandle,
CSSM_CC_HANDLE  KRRegistrationContextHandle,
const CSSM_CONTEXT_PTR  KRRegistrationContext,
CSSM_DATA_PTR  KRInData,
CSSM_CRYPTO_DATA_PTR  UserCallback,
unit32  *EstimatedTime
CSSM_HANDLE_PTR  ReferenceHandle)
```

This function performs a key recovery registration operation. The *KRInData* contains known input parameters for the recovery registration operation. A *UserCallback* function may be supplied to allow the registration operation to interact with the user interface, if necessary. When this operation is successful, a *ReferenceHandle* and an *EstimatedTime* parameter are returned. The *ReferenceHandle* will be used to invoke the CSSM_KR_RegistrationRetrieve function after the *EstimatedTime* in seconds.

Parameters

KRSPHandle (input)
The handle that describes the KRSP module used to perform upcalls to KeyWorks for the memory functions managed by KeyWorks.

KRRegistrationContextHandle (input)
The handle that describes the context of this key recovery operation used to link to the KRSP-managed information.

KRRegistrationContext (input)
Pointer to CSSM_CONTEXT structure that describes the attributes with this key recovery context.

KRInData(input)
Input data for key recovery registration.

UserCallback (input)
A callback function that may be used to collect further information from the user interface.

EstimatedTime (output)
The estimated time after which the KR_RegistrationRetrieve call should be invoked to obtain registration results.

ReferenceHandle (output)
The handle to use to invoke the KR_RegistrationRetrieve function.

Return Values

A CSSM return value. This function returns CSSM_OK if successful, and returns CSSM_FAIL if an error has occurred. Use CSSM_GetError to determine the exact error.

3.6.2 KRSP_RegistrationRetrieve

CSSM_RETURN CSSMKRSPI KRSP_RegistrationRetrieve

```
(CSSM_KRSP_HANDLE KRSPHandle,
CSSM_HANDLE ReferenceHandle,
unit32 *EstimatedTime,
CSSM_KR_PROFILE_PTR KRProfile)
```

This function completes a key recovery registration operation. The results of a successful registration operation are returned through the *KRProfile* parameter, which may be used with the profile management API functions.

If the results are not available when this function is invoked, the *KRProfile* parameter is set to NULL, and the *EstimatedTime* parameter indicates when this function should be repeated with the same *ReferenceHandle*.

Parameters

KRSPHandle (input)
The handle that describes the KRSP module used to perform *upcalls* to KeyWorks for the memory functions managed by KeyWorks.

ReferenceHandle (input)
The handle to the key recovery registration request that will be completed.

EstimatedTime (output)
The estimated time after which this call should be repeated to obtain registration results. This is set to a non-zero value only when the *KRProfile* parameter is NULL.

KRProfile (output)
Key recovery profile that is filled in by the registration operation.

Return Values

A CSSM return value. This function returns CSSM_OK if successful, and returns CSSM_FAIL if an error has occurred. Use CSSM_GetError to determine the exact error.

3.7 Key Recovery Enablement Functions

3.7.1 KRSP_GenerateKRFields

```
CSSM_RETURN CSSMKRSPI KRSP_GenerateKRFields
                    (CSSM_KRSP_HANDLE  KRSPHandle,
                    CSSM_KRC_HANDLE KRContextHandle,
                    const CSSM_CONTEXT_PTR KRContext,
                    CSSM_CC_HANDLE CryptoContextHandle,
                    const CSSM_CONTEXT_PTR CryptoContext,
                    CSSM_DATA_PTR KRAlgAttributes,
                    uint32 KRFlags,
                    CSSM_DATA_PTR KRFields)
```

This function generates the KRFs for a cryptographic association given the key recovery context, and the cryptographic context containing the key that will be made recoverable. The session attribute and the flags are interpreted by the KRSP. A set of KRFs (KRFields) is returned if the function is successful. The KRFlags parameter may be used to fine tune the contents of the KRFields produced by this operation. On successful return, the cryptographic context handle may be used with the encryption operations of the KeyWorks.

Parameters

KRSPHandle (input)
The handle that describes the KRSP module used to perform *upcalls* to KeyWorks for the memory functions managed by KeyWorks.

KRContextHandle (input)
The handle that describes the context of this key recovery operation used to link to the KRSP-managed information.

KRContext (input)
Pointer to CSSM_CONTEXT structure that describes the attributes with this key recovery context.

CryptoContextHandle (input)
The handle that describes the cryptographic context used to link to the Cryptographic Service Provider (CSP)-managed information.

CryptoContext (input)
Pointer to CSSM_CONTEXT structure that describes the attributes of the cryptographic context.

KRAlgAttributes (input)
The KRSP specific options. These options are not interpreted by the KeyWorks Framework, but are passed on to the KRSP.

KRFlags (input)
Flag values for KRFs generation. Defined values include the following:

KR_IND - Signifies that only the individual KRFs should be generated.

KR_ENT - Signifies that only the enterprise KRFs should be generated.

KR_LE - Signifies that only the law enforcement KRFs should be generated.

KR_ALL **-** Signifies that law enforcement, enterprise, and individual KRFs should be generated.

KR_OPTIMIZE **-** Signifies that performance optimization options will be adopted by a KRSP while implementing this operation

KR_DROP_WORKFACTOR **-** Signifies that the KRFs should be generated without using the key size work factor.

KRFields (output)
The KRFs in the form of a data blob.

Return Values

A CSSM return value. This function returns CSSM_OK if successful, and returns CSSM_FAIL if an error has occurred. Use CSSM_GetError to determine the exact error.

3.7.2 KRSP_ProcessKRFields

```
CSSM_RETURN CSSMKRSPI KRSP_ProcessKRFields
                (CSSM_KRSP_HANDLE KRSPHandle,
                CSSM_KRC_HANDLE KRContextHandle,
                const CSSM_CONTEXT_PTR KRContext,
                CSSM_CC_HANDLE CryptoContextHandle,
                const CSSM_CONTEXT_PTR CryptoContext,
                CSSM_DATA_PTR KRAlgAttributes,
                uint32 KRFlags,
                CSSM_DATA_PTR KRFields)
```

This call processes a set of KRFs given the key recovery context, and the cryptographic context for the encryption operation. On successful return, the cryptographic context handle may be used for the decryption API calls of the KeyWorks.

Parameters

KRSPHandle (input)
The handle that describes the KRSP module used to perform *upcalls* to KeyWorks for the memory functions managed by KeyWorks.

KRContextHandle (input)
The handle that describes the context of this key recovery operation used to link to the KRSP-managed information.

KRContext (input)
Pointer to CSSM_CONTEXT structure that describes the attributes with this key recovery context.

CryptoContextHandle (input)
The handle that describes the cryptographic context used to link to the CSP-managed information.

CryptoContext (input)
Pointer to CSSM_CONTEXT structure that describes the attributes of the cryptographic context.

KRAlgAttributes (input)
The KRSP specific options. These options are uninterpreted by KeyWorks, but passed on to the KRSP.

KRFlags (input)
Flag values for KRFs generation. Defined values include the following:

KR_ENT - Signifies that only the enterprise KRFs should be processed.

KR_LE - Signifies that only the law enforcement KRFs should be processed.

KR_ALL - Signifies that law enforcement and enterprise KRFs should be processed.

KR_OPTIMIZE - Signifies that available optimization options will be adopted.

KRFields (input)
The KRFs to be processed in the form of a data blob.

Return Values

A CSSM return value. This function returns CSSM_OK if successful, and returns CSSM_FAIL if an error has occurred. Use CSSM_GetError to determine the exact error.

3.8 Key Recovery Request Functions

3.8.1 KRSP_GetRecoveredObject

```
CSSM_RETURN CSSMKRSPI KRSP_GetRecoveredObject
                                   (CSSM_KRSP_HANDLE KRSPHandle,
                                   CSSM_HANDLE_PTR ResultsHandle,
                                   unit32 IndexInResults,
                                   CSSM_CRYPTO_DATA_PTR PassPhrase,
                                   CSSM_KEY_PTR RecoveredKey,
                                   uint32 Flags,
                                   CSSM_DATA_PTR OtherInfo)
```

This function is used to step through the results of a recovery request operation in order to retrieve a single recovered key at a time, along with its associated meta information. The reference handle from a successful CSSM_KR_RecoveryRetrieve operation is used as the results handle in this operation. When multiple keys are recovered by a single recovery request operation, the index parameter indicates which item to retrieve through this function.

The RecoveredKey parameter serves as an input template for the key to be returned. If a private key will be returned by this operation, the PassPhrase parameter is used to inject the private key into the CSP indicated by the RecoveredKey template. The corresponding public key is returned in the RecoveredKey parameter. Subsequently, the PassPhrase and the public key may be used to reference the private key when operations using the private key are required. The OtherInfo parameter may be used to return other meta data associated with the recovered key.

Parameters

KRSPHandle (input)
The handle that describes the KRSP module used to perform *upcalls* to KeyWorks for the memory functions managed by KeyWorks.

ResultsHandle (input)
The handle used in a successful KR_RecoveryRequest function.

IndexInResults (input)
The index into the results that are referenced by the ResultsHandle parameter.

PassPhrase (input)
This parameter is only relevant if the recovered key is a private key. It is used to protect the private key when it is inserted into the CSP specified by the RecoveredKey template.

RecoveredKey (output)
This parameter serves as an input template and contains the recovered key.

Flags (input)
Flag values relevant for recovery of a key. Possible values include the following:

CERT_RETRIEVE - If the recovered key is a private key, return the corresponding public key certificate in the OtherInfo parameter.

OtherInfo (output)
This parameter is used if there is additional information associated with the recovered key (such as the public key certificate when recovering a private key) that will be returned.

Return Values

A CSSM return value. This function returns CSSM_OK if successful, and returns CSSM_FAIL if an error has occurred. Use CSSM_GetError to determine the exact error.

3.8.2 KRSP_RecoveryRequest

```
CSSM_RETURN CSSMKRSPI KRSP_RecoveryRequest
                    (CSSM_KRSP_HANDLE KRSPHandle,
                    CSSM_CC_HANDLE KRRequestContextHandle,
                    const CSSM_CONTEXT_PTR KRRequestContext,
                    CSSM_DATA_PTR KRInData,
                    CSSM_CRYPTO_DATA_PTR UserCallback,
                    unit32 *EstimatedTime,
                    CSSM_HANDLE_PTR ReferenceHandle)
```

This function performs a key recovery request operation. The KRInData contains known input parameters for the recovery request operation. A UserCallback function may be supplied to allow the recovery operation to interact with the user interface, if necessary. If the recovery request operation is successful, a ReferenceHandle and an EstimatedTime parameter are returned. The ReferenceHandle will be used to invoke the KR_RecoveryRetrieve function after the EstimatedTime in seconds.

Parameters

KRSPHandle (input)
The handle that describes the KRSP module used to perform *upcalls* to KeyWorks for the memory functions managed by KeyWorks.

KRRequestContextHandle (input)
The handle that describes the context of this key recovery operation used to link to the KRSP-managed information.

KRRequestContext (input)
Pointer to CSSM_CONTEXT structure that describes the attributes with this key recovery context.

KRInData(input)
Input data for key recovery requests. For encapsulation schemes, the KRFs are included in this parameter.

UserCallback (input)
A callback function that may be used to collect further information from the user interface.

EstimatedTime (output)
The estimated time after which this call should be repeated to obtain recovery results. This is set to a non-zero value only when the ResultsHandle parameter is NULL.

ReferenceHandle (output)
Handle returned when recovery request is successful. This handle may be used to invoke the KR_RecoveryRetrieve function.

Return Values

A CSSM return value. This function returns CSSM_OK if successful, and returns CSSM_FAIL if an error has occurred. Use CSSM_GetError to determine the exact error.

3.8.3 KRSP_RecoveryRequestAbort

```
CSSM_RETURN CSSMKRSPI KRSP_RecoveryRequestAbort
                                        (CSSM_KRSP_HANDLE KRSPHandle,
                                        CSSM_HANDLE ResultsHandle)
```

This function terminates a recovery request operation and releases any state information related to the recovery request.

Parameters

KRSPHandle (input)
The handle that describes the KRSP module used to perform *upcalls* to KeyWorks for the memory functions managed by KeyWorks.

ResultsHandle (input)
The handle used in a successful KR_RecoveryRequest function.

Return Values

A CSSM return value. This function returns CSSM_OK if successful, and returns CSSM_FAIL if an error has occurred. Use CSSM_GetError to determine the exact error.

3.8.4 KRSP_RecoveryRetrieve

CSSM_RETURN CSSMKRSPI KRSP_RecoveryRetrieve
(CSSM_KRSP_HANDLE KRSPHandle,
CSSM_HANDLE_PTR ReferenceHandle,
unit32 *EstimatedTime, uhnit32 *NumberOfResults)

This function completes a key recovery request operation. The ReferenceHandle parameter indicates which outstanding recovery request will be completed. On successful return of this function, the results of the recovery operation are now referenced by the ReferenceHandle parameter, which may be used with the CSSM_KR_GetRecoveredObject function to retrieve the recovered keys one at a time.

If the results are not available at the time this function is invoked, the EstimatedTime parameter indicates when this operation should be repeated with the same ReferenceHandle.

Parameters

KRSPHandle (input)
The handle that describes the KRSP module used to perform *upcalls* to KeyWorks for the memory functions managed by KeyWorks.

ReferenceHandle (input)
Handle that indicates which key recovery request operation will be completed.

EstimatedTime (output)
The estimated time after which this call should be repeated to obtain recovery results. This is set to a non-zero value only when the the results are not yet available, and the ReferenceHandle parameter needs to be used to repeat this call.

NumberOfResults (output)
The number of recovered key objects that may be obtained using the ReferenceHandle.

Return Values

A CSSM return value. This function returns CSSM_OK if successful, and returns CSSM_FAIL if an error has occurred. Use CSSM_GetError to determine the exact error.

3.9 Key Recovery Privilege Functions

3.9.1 KRSP_PassPrivFunc

CSSM_RETURN CSSMKRSPI KRSP_PassPrivFunc
(CSSM_PRIV_FUNC_PTR SetContextPriv)

This function supports a privileged mode of the KRSP with respect to the policies enforced by the framework. The framework uses this SPI to pass down a privileged framework entry point in the form of the SetContextPriv function. When the KRSP needs to bypass the policy enforcement performed by the framework, the KRSP uses this privileged entry point (SetContextPriv( )) to make an encryption context privileged. This privileged context can be used by the KRSP and the framework will not enforce its policies on this context. As soon as the KRSP has finished using the privileged context, it is expected that the KRSP will use the SetContextPriv( ) entry point to reset the privileged mode of the context.

Parameters

SetContextPriv (input)
The privileged entry point to the framework that can be used to set a context to a privileged mode.

Return Values

A CSSM return value. This function returns CSSM_OK if successful, and returns CSSM_FAIL if an error has occurred. Use CSSM_GetError to determine the exact error.

Appendix A. List of Acronyms

AI	Authentication Information
API	Application Programming Interface
CA	Certificate Authority
CL	Certificate Library
CRL	Certificate Revocation List
CSP	Cryptographic Service Provider
DL	Data Storage Library
DLL	Dynamically Linked Library
GUID	Globally Unique ID
ISV	Independent Software Vendor
KRA	Key Recovery Agent
KRF	Key Recovery Field
KRPT	Key Recovery Policy Table
KRS	Key Recovery Server
KRSP	Key Recovery Service Provider
SPI	Service Provider Interface
TP	Trust Policy

Appendix B. 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. Rivest-Shamir-Adelman (RSA) 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 Providers	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 the 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: public key and a private key. The two keys are mathematically related, but is virtually impossible to derive the private key from the public key. A message this 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 a 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 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 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) 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 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.
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 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 purchased 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-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 DES (Data Encryption Standard) 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.