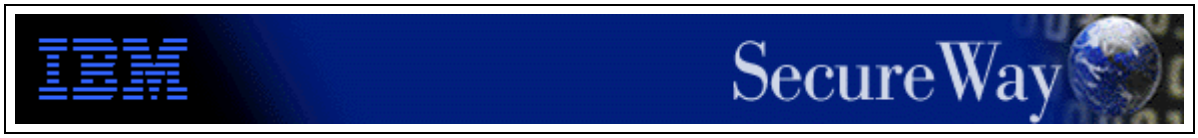

Towards a Framework-based solution to Cryptographic Key Recovery

Currently available architectures for key-recovery-enabled cryptographic systems are either very restrictive or inflexible. We propose a new architecture for such systems that builds in the potential to support a wide variety of key recovery mechanisms and cryptographic mechanisms under a common uniform framework. This paper will highlight the motivations for, and architecture of, such a framework-based, key-recovery solution, and discuss some of its salient features.

Towards a Framework-based solution to Cryptographic Key Recovery

1. Introduction

In recent times, cryptography has come into widespread use in meeting multiple security needs, such as confidentiality, integrity, authentication and non-repudiation [1]. The use of cryptographic products for confidentiality creates the need for supporting conflicting requirements between users and their respective governments or enterprises. While users have a legitimate need to establish and maintain confidentiality of their data and communications, governments and enterprises have, at times, a legitimate need to intercept and recover such confidential data and communications under proper legal conditions. This conflict becomes especially apparent when users' applications begin to use strong encryption techniques, which can either be too expensive or impossible to break within reasonable time.

Some governments such as the US, Canada, and France, impose controls on the export and foreign dissemination of cryptographic products on the premise that they are critical to national security and foreign policy interests. This is a major hindrance for manufacturers and vendors of cryptographic products since the market for their encryption products is severely restricted by such jurisdiction based controls. To mitigate this, key recovery techniques have been proposed as a means to relax export controls on cryptographic products. Certain governments, such as the US, now have a stated policy that strong encryption based products can be licensed for general purpose export if they can be shown to incorporate an acceptable mechanism for key recovery. Adoption of such a policy enables cryptographic product vendors to develop a single international version of their product that contains strong encryption along with some technique for key recovery.

Key recovery mechanisms serve other useful purposes as well. They may be used by individuals to recover lost or corrupted keys; they may be used to deter corporate insiders from using encryption to bypass the corporate security policy regarding the flow of proprietary information across jurisdictions. Corporations may also use key recovery mechanisms to recover employee keys in certain situations, e.g. in the employee's absence. Finally, 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. Thus, there appear to be added incentives - beyond those of satisfying the government's needs - for the incorporation as well as adoption of key-recovery mechanisms in local and distributed encryption based systems.

Several vendors, such as Hewlett Packard, Trusted Information Systems and others have or are developing exportable cryptographic systems based on key recovery techniques. The major deficiency in all currently available or proposed architectures for key-recovery-enabled cryptographic systems is that they are very restrictive and inflexible. The design of some of these products is based on restrictive assumptions such as all users need to be certified under a common public key infrastructure (PKI). Others products are very inflexible since they bundle a specific key recovery mechanism with a specific key transport mechanism or a specific cryptographic engine. Others support a single proprietary key recovery mechanism, which may not be acceptable

for export to certain jurisdictions that choose not to adopt or legislatively support that key recovery mechanism. To avoid these and possibly other limitations, we propose an architecture for key-recovery-enabled cryptographic systems that builds in the potential to support a wide variety of key recovery mechanisms and cryptographic mechanisms under a common uniform framework. Additional benefits of such a framework-based solution is that it is not tied to any particular communications protocol or key transport mechanism, and can be adapted to conform to any jurisdiction-based, key-recovery policy.

This paper will highlight the motivations for, and architecture of, such a framework-based, key-recovery solution, and discuss some of its salient features. Section 2 discusses the nomenclature relevant to key recovery. Section 3 presents the motivation for defining a framework-based solution to key recovery. Section 4 delves into the architecture of IBM's Key Management Framework. Sections 5 and 6 discuss the meaning of trust policy and noncircumventability, respectively, in the context of the framework-based architecture. Section 7 highlights some of the inter-operability scenarios for key recovery products. Finally, Section 8 presents our conclusions.

2. Key Recovery Nomenclature

Denning and Brandstad [2], present a taxonomy of key escrow systems. In this paper, 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 third 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* and *key encapsulation*. Key escrow techniques are based on the paradigm that the government or a trusted third party called an *escrow agent*, holds the actual user keys or portions thereof. Key encapsulation techniques, on the other hand, are based on the paradigm that a cryptographically encapsulated form of the key is made available to third parties that require key recovery; the encapsulation technique ensures that only certain trusted third parties called *recovery agents* 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 that is either escrowed or encapsulated. Some schemes rely on the escrow or encapsulation of long-term keys, such as private keys, while other schemes are based on the escrow or encapsulation of ephemeral keys such as session keys.

Mechanism	Advantages	Disadvantages
Key Escrow for long-term keys	* No change to existing communication protocols * no latency in using ephemeral key	* lack of privacy for individuals * coarse granularity for recoverable keys * latency involved in obtaining and using long-term key * need for individual to belong to government approved PKI
Key Encapsulation for ephemeral keys	* more privacy for individuals * fine granularity for recoverable keys * no latency to obtain and use public keys * no need for individuals to belong to government approved PKI	* requires modifications to existing communications protocols * some latency involved in key encapsulation for every ephemeral key

Table 1. Comparison of typical escrow and encapsulation schemes

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 myriads of ephemeral keys. These long-term "escrowed" keys are then used to retrieve the ephemeral keys used for data confidentiality.

Key encapsulation techniques can also choose to archive the encapsulated keys, but typically they do not. Instead, these techniques usually operate on the ephemeral keys, and associate the encapsulated key with the actual enciphered message and thereby dispense with the archival process. The encapsulated key is put into a key recovery block that is generated by the party performing the data encryption, and associated with the encrypted data. To ensure the transmission and the integrity of the key recovery block, 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 key recovery block is processed successfully. Since the key recovery block has to be associated with the cryptographic session in some way, key encapsulation schemes may require the perturbation of the communication protocol used.

Table 1 illustrates the advantages and disadvantages of key escrow and key encapsulation mechanisms. With escrow schemes for long terms keys, a major advantage is that the cryptographic communication protocol does not need adaptation; however, the serious disadvantages are the lack of privacy for individuals (since their private keys are held by a separate entity), and the lack of granularity with respect to the recoverable keys. Additionally, there is the burden that every individual has to obtain and use public keys through an approved public key infrastructure in order for the key recovery scheme to work.

The major advantage to key encapsulation schemes based on ephemeral keys are that there is much greater privacy for the individuals; they can generate and keep their own private keys. Each ephemeral key can be recovered independently so there is maximum possible granularity with respect to the recoverable keys. A disadvantage is that the communication protocol between

sending and receiving parties needs to be adapted to allow the flow of the encapsulated key. Another disadvantage is that there is some performance penalty in key encapsulation for each ephemeral key; however, this may be minimized by caching techniques.

2.2. Key Recovery Phases

The process of cryptographic key recovery involves two major phases. First, parties that are involved in cryptographic associations have to perform operations to enable key recovery (such as the escrow of user keys, or the generation of key recovery blocks, etc.) - this is typically called the *key recovery enablement* phase. Next, authorized third parties that desire to recover the data keys, do so with the help of a recovery server and one or more escrow agents or recovery agents - this is the actual *key recovery service* phase. Figure 1 illustrates the two phases of key recovery.

It is envisioned that governments or organizations will operate their own recovery server hosts independently, and that key recovery servers may support a single or multiple key recovery mechanisms. There are a number of important issues specific to the implementation and operation of the key recovery servers, such as vulnerability and liability. The focus of this paper is a framework based approach to key recovery enablement, thus the issues with respect to the key recovery server will not be discussed further here.

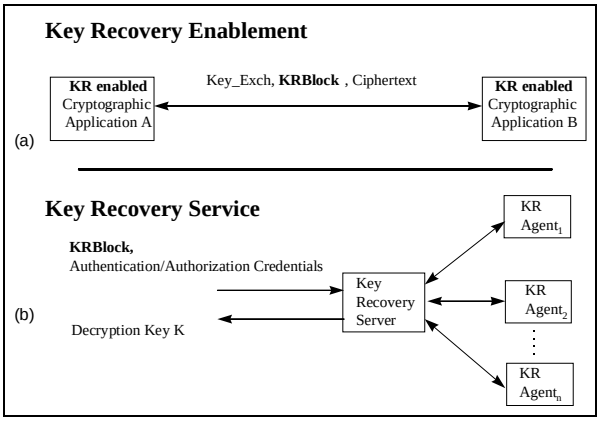

Figure 1. Key Recovery Phases

As illustrated in Figure 1(a), two key-recovery-enabled cryptographic applications are communicating using a key encapsulation mechanism; the key recovery block is passed along with the ciphertext, to enable subsequent key recovery. The key recovery service is illustrated in Figure 1(b), where the key recovery block is provided as input to the key recovery server along with the authorization credentials of the client requesting service. The key recovery server interacts with one or more local or remote key recovery agents to reconstruct the secret key that can be used to decrypt the ciphertext.

2.3. Key Recovery Policy

Key recovery policies are mandatory policies that are derived from jurisdiction-based export and import regulations on cryptographic products. Key recovery policies apply to cryptographic

products that cross jurisdiction boundaries. The jurisdictions for the key recovery policy typically (but not always) coincide with the political boundaries of countries. Jurisdictions may choose to define their key recovery policies for cryptographic products based on export controls, import controls, or both.

Key recovery policies come in two flavors: *key recovery enablement policies* and *key recovery inter-operability policies*. Key recovery enablement policies specify the exact cryptographic protocol suites (algorithms, modes, key lengths etc.) where key recovery enablement is mandated. Key recovery inter-operability policies specify to what degree a key-recovery-enabled cryptographic product is allowed to inter-operate with other cryptographic products.

For a specific cryptographic product, the key recovery policies for multiple jurisdictions may apply simultaneously. The policies (if any) of the jurisdiction(s) 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.

3. Motivation for Framework Approach

In the context of this paper, a framework is *a layer of abstraction* that isolates application code (that requires a set of services) from the modules that actually implement the services; the application deals with an abstract application programming interface (API), while the service providers conform to abstract service provider interfaces (SPIs). There are several well-established advantages of a framework-based system. By providing clean standardized interfaces for various parts of the system, a framework promotes layering, a high degree of modularity, decoupling of components, ease of module-based conformance testing, and the ability to mix and match modules that are produced by independent vendors that conform to standardized interfaces.

We have already established that there are several strong incentives for developing key-recovery-enabled cryptographic systems, and that existing implementations are seriously lacking in terms of flexibility and extensibility. Therefore, the primary motivation for a framework based solution to key recovery is to develop a very general purpose and open architecture that will allow the support of varied key recovery and cryptographic mechanisms. Moreover, such a framework based approach offers added benefits in terms of the communications protocols supported and the flexibility to use multiple PKIs.

There are multiple vendors that have or are in the process of developing cryptographic service provider modules that are based on hardware or software cryptographic engines. These vendors would like to internationalize their products. Since, different countries around the world support different key recovery systems, embedding a single key recovery mechanism *into* the cryptographic engine is counterproductive since the composite product may not be suitable for use in certain jurisdictions. Additionally, there is a burden on the vendors of cryptographic service providers (CSPs) to incorporate key recovery mechanisms that are by and large unrelated to the central purpose of the product. Thus, it is highly desirable to decouple the key recovery enablement service from the underlying cryptographic service.

The concept of a cryptographic framework (such as Microsoft's Cryptographic API, or Intel's Common Data Security Architecture) grew out of the need to provide an abstract set of cryptographic services to application level code; the cryptographic framework isolates the applications from the idiosyncrasies of specific cryptographic service providers. Similarly, the concept of a framework-based key recovery solution grew out of the need to provide an *abstract set of key-recovery-enabled cryptographic services to application level code.* While the application uses a uniform set of APIs, the framework allows the plug-in of multiple and varied cryptographic service provider modules and key recovery service provider modules. The framework architecture decouples the cryptographic mechanisms from the key recovery mechanisms, while *allowing the seamless integration of independent CSPs and key recovery service providers (KRSPs)* to provide a uniform set of services to application level code. This is a very important point since it *allows the CSPs to remain unchanged and oblivious of key recovery mechanisms*. The abstraction provided by a framework-based architecture allows the APIs to be useful in a broad range of communication protocol and encryption environments.

4. SecureWay(TM)[1] Key Management Framework

The SecureWay(TM) Key Management Framework (SKMF) defines an infrastructure for a complete set of cryptographic services augmented with key recovery enablement. The layered architecture of a SKMF based system is depicted in Figure 2. There are essentially three major layers - the application layer invokes the SKMF layer, while the SKMF layer invokes the service provider (SP) layer. The application layer code invokes the cryptographic API and key-recovery API supported by the SKMF. Multiple key recovery mechanisms and cryptographic mechanisms can be implemented as service providers that plug-in underneath the framework using the well-defined service provider interfaces provided by the framework. The SKMF implements the supported API calls by making appropriate invocations of the service provider modules using the SPIs. The primary functionality of the SKMF layer is to maintain state regarding the connections between the application layer code and the service providers underneath. Additionally, the SKMF mediates all interactions between applications and CSPs, and implements and enforces the applicable key recovery export policy. Finally, the SKMF allows the seamless integration of the key recovery functions and the cryptographic functions provided by independent service provider modules.

The design of the SKMF starts with an existing cryptographic framework that meets certain abstract properties such as extensibility and modularity. Next, the cryptographic framework is extended laterally to include a key recovery module manager that supports a set of key recovery APIs (KR-APIs) for use by application layer code and a set of key recovery SPIs (KR-SPIs) to support plug-in key recovery service provider modules. The KR-APIs are used to set state and attribute information for use by the key recovery service providers. The KR module manager also contains the static key recovery enablement policy table and an enforcement module that ensures that all cryptographic associations abide by the key recovery enablement policy.

---

1 SecureWay is a registered trademark of IBM Corporation.

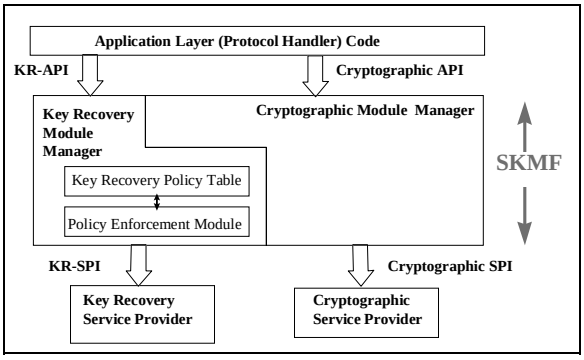

Figure 2. SKMF System Architecture

The base cryptographic module manager is modified to accommodate the key recovery module manager. While the base cryptographic API is kept intact (in order to perturb existing applications as little as possible), the cryptographic module manager is modified to invoke the key recovery policy enforcement module. If key recovery enablement is required for a certain cryptographic operation, the KRSP is invoked appropriately before the invocation of the CSP to perform the cryptographic operation. As a result, the key recovery enablement occurs transparently as part of the cryptographic operations. Since key recovery enablement typically requires additional parameters such as public keys of key recovery agents, authorization information for users, etc., these additional parameters need to be provided to the SKMF through some mechanism. The KR-APIs can be used by the application layer code to set up the necessary parameters for key recovery enablement. These parameters are held as state information within the KR management module. When the cryptographic APIs are used by the application, the KR state information is used to invoke the key recovery service provider module as necessary.

The KRSPs perform two fundamental operations with respect to key recovery enablement: key recovery block generation, and key recovery block processing. On the sending side of a cryptographic association, key recovery enablement implies that a key recovery block be generated and sent out to the receiving side. On the receiving side, key recovery enablement requires that the key recovery block be processed before decryption can occur; processing ensures that the key recovery block was transmitted intact from the sender to the receiver.

4.2. Properties of the SKMF

The SKMF layer possesses several abstract properties that allow it to provide key recovery enablement to a set of cryptographic services. These properties are:

- **Replacement Independence and Modularity of Services**: This property ensures that the service providers supported underneath the SKMF can be seamlessly replaced without affecting the operation of the application running on top of the framework. In other words, the SKMF APIs and SPIs are abstract enough to allow the plug-in of various KRSPs and CSPs.

- **Separation of Policy from Mechanism**: The SKMF layer is responsible for enforcing the key recovery policy. The actual mechanism for key recovery enablement is embedded in the KRSP that is used and is clearly separated from the SKMF.
- **Noncircumventability**: Noncircumventability implies that the application layer code using the SKMF to obtain cryptographic services cannot bypass the framework to directly access the service providers underneath. Available operating system protection mechanisms are to be utilized to the fullest extent to ensure that the framework is not bypassable. A further aspect of the noncircumventability property dictates that there must exist an invocation dependency between the cryptographic operations and the key recovery enablement operations, such that the appropriate key recovery operations are always invoked before the application can obtain cryptographic services from the framework.
- **Isolation and tamperproofness** : The SKMF needs to be isolated from the application layer code in order to ensure that the key recovery policy enforcement module cannot be tampered with. To achieve isolation and tamperproofness, the SKMF must ensure that (1) the parameters passed in through the APIs are verified at each invocation, (2) there are controlled entry points through which the APIs may be invoked, and (3) the address spaces of the KR mechanisms and policies are separated (to the extent possible within the application domain) from those of the application layer code invoking the APIs.

4.3. Packaging of the SKMF System Components

There are various options in packaging the individual components of the SKMF based system. One option is to statically bundle the application layer code along with the SKMF and a set of service providers into a single exportable product package. The entire package has to undergo rigorous export compliance tests in this case.

Other options include packaging the application layer code without the SKMF or the service providers, but allowing the latter to be dynamically linked in at runtime. In this case, the application layer code undergoes minimal (if any) export compliance testing. The SKMF and a set of service providers may be bundled into a dynamic link library (DLL) with well-defined exported interfaces. This DLL has to undergo rigorous export compliance tests to ensure that whether the application layer code is well-behaved or ill-behaved, the key recovery enablement operations will occur before cryptographic operations are obtained by the application code.

In order to allow dynamic additions to the set of service providers plugged-in under the framework, there needs to be a way to package the service providers as separate DLLs that can be added to an existing SKMF implementations. Since these service provider DLLs may contain export regulated cryptography, they have to undergo strict export compliance testing to ensure that they can be usable only when plugged in under a export-approved SKMF implementation. The next two sections of this paper discuss some implementation schemes to facilitate the packaging options described in this section.

5. Trust Policy

A system that provides key-recovery-enabled cryptographic services needs to possess a certain degree of trust. This is because the system handles critical data such as cryptographic keys, makes policy and access control decisions, establishes the validity of public key certificates, and generates and processes the key recovery block. Since the key recovery policy that applies to a system is typically a mandatory policy within a given environment, the code that implements and enforces these operations have to be placed within a perimeter of trust. The notion of trust in this context refers to a degree of assurance that the modules that pertain to key recovery are ***correct*** in implementation, and that they are ***tamperproof*** and ***noncircumventable***. For well-known reasons, it is desirable to minimize the perimeter of trust as much as possible.

In a non-modular and monolithic key-recovery-enabled cryptographic system, the perimeter of trust extends over the entire product. In the SKMF architecture, the perimeter of trust encompasses the framework and the service providers underneath. The framework is trusted to implement and enforce the appropriate key recovery policy (based on the jurisdictions of manufacture and installation of the product). The CSPs are trusted to perform the cryptographic operations requested of them, while the Key Recovery Service Providers are trusted to generate and process the key recovery block relevant to a cryptographic association. If the framework and the service providers are packaged as dynamically linked modules that are used by an application, then there needs to be a chain of trust that establishes the required trust perimeter.

In the SKMF architecture, the chain of trust is established as follows:

- the framework performs self-integrity checks using digital signature verification techniques
- the framework verifies the integrity of each service provider before attaching to it
- the service provider authenticates the framework before allowing itself to be attached dynamically

6. Noncircumventability

In the SKMF architecture, the SKMF always intercepts all connections between the application layer code and the service providers underneath. This is a very fundamental feature of the SKMF, which allows the SKMF to mediate all cryptographic operations and perform key recovery enablement operations when required. Basically, this implies that the SKMF be noncircumventable or non-bypassable. The architecture has to ensure that the application layer code cannot directly invoke the service provider interfaces without going through the framework.

Depending on the operating system environments where the SKMF based system is used, various available mechanisms are used to ensure the property of noncircumventability. If the SKMF and its underlying service providers are implemented within the operating system, then the address space separation provided by the system can be utilized to ensure that only the SKMF APIs are available as entry points for the application layer code. On the other hand, if the SKMF and its underlying service providers are packaged as separate dynamic link libraries used by the application, then the noncircumventability property is enforced by the following technique. The

service provider DLLs do not possess any exported interfaces. When the service provider (such as a CSP) is attached by a process, a phase of mutual verification continues between the SKMF and the service provider until both entities are certain about the other entity; after this is completed, the service provider dynamically sends the SKMF the set of entry points for its services. This tight coupling mechanism ensures two basic purposes: only the SKMF can attach to the service provider DLLs, and the DLLs dynamically transmit their entry points to the attaching application only after ensuring that it is the SKMF.

7. Inter-operability Scenarios for Key-recovery-enabled Products

The key recovery inter-operability policy specifies how a key recovery product can inter-operate with other products. According to the US export policy, key recovery products are allowed to inter-operate with other key recovery products using strong encryption as long as the key recovery block is made available to law enforcement. For products based on private key escrow mechanisms tied to the use of a particular public key infrastructure, inter-operability between systems translates to the ability of the sender and receiver to establish trust in the certificates of the “escrowed” public key infrastructure and use these public keys for ephemeral key transport.

For products using key encapsulation techniques, the key recovery policy stipulates that when using strong encryption, the sending side cannot perform data encryption without producing the necessary key recovery block, and that the receiving side cannot perform data decryption without processing the relevant key recovery block. The key recovery block has to be transmitted from the sender to the receiver - implying that the communication protocol between the sender and the receiver has to be augmented to allow the passage of the additional fields. Fundamentally there are three scenarios of inter-operability as shown in Figure 3.

Let us examine each of the three scenarios of inter-operability with respect to a SKMF based system where the SKMF and its plug-in service providers are high assurance modules, while the application layer code is low assurance. In other words, we have trust in the SKMF and SP components, while we do not have trust in the application layer code. There is a good reason for assuming that the SKMF and SPs are trusted, since these components undergo strict screening for export controls while the application code undergoes little or no screening for exportability. Since the SKMF can at best generate and process the key recovery block, it is the application layer code that actually has the burden of sending and receiving it. Let us define the application layer code as being “well-behaved” if it sends out the key recovery block (generated by the SKMF) along with the session data and receives the key recovery block from the session data and passes it down to the SKMF for processing.

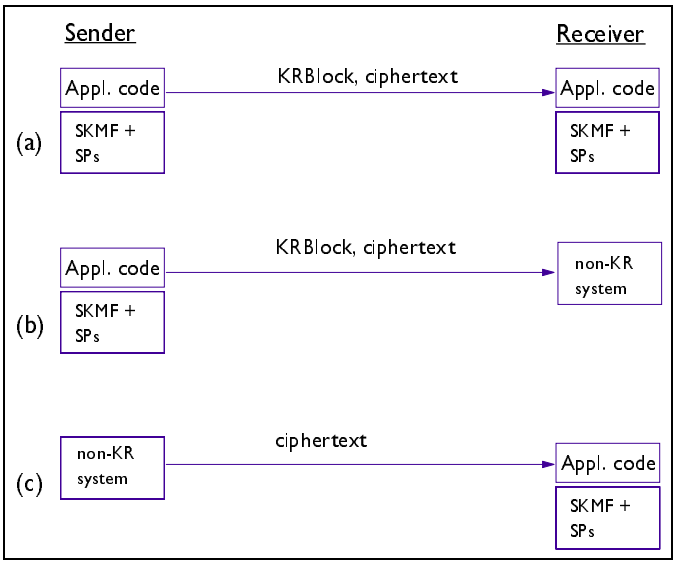

Figure 3. Inter-operability Scenarios for Key Recovery Enabled Products

In scenario (a), the sender generates the key recovery block and sends it over to the receiver, and the receiver verifies them before allowing data decryption. This inter-operability scenario is easy to implement, since both the sender and receiver are aware of the additional key recovery block and knows how to handle it. As long as the application layer code on at least one side is well-behaved, the key recovery enablement operation remains intact, since the key recovery block needs to flow along with the session data in order for a successful cryptographic association to occur. However, if the application layer code on both the sending and receiving sides are ill-behaved and are acting in collusion, then there are various cut-and-paste attacks that can be mounted such that the key recovery block either does not flow between sender and receiver, or flows out-of-band with the session; thus the key recovery enablement operation is inhibited.

In scenario (b), the sender is a KR system and the receiver is not. Full inter-operability in this scenario is a little more tricky since the sending side sends over the key recovery block along with the session data, but the receiving side has to be able to ignore the additional fields and proceed with the decryption step. If the application layer code on the sending side is well-behaved, the key recovery operation is successful; however, if the application code on the sending side is ill-behaved, and does not send the key recovery block as part of the session, the key recovery enablement operation is inhibited.

In scenario (c), the receiver is a key-recovery-enabled system while the sender is not. The receiving system should not be able to decrypt data until the key recovery block is received from the sender. Since the sender has no notion of a key recovery block, inter-operability in this case is nebulous at best. In key encapsulation schemes, a pure sender receiver association where only the receiver is key-recovery-enabled may be infeasible.

In discussing the above schemes, it becomes apparent that the set of requirements for a key-recovery-enabled system are somewhat in opposition to the inter-operability requirements for such systems. As pointed out before, the actual communication protocol between the sender and receiver has to be modified somewhat to allow the key recovery block to flow. To allow inter-operability with systems that are not key-recovery-enabled, the key recovery block is typically sent in a way that does not disturb the basic key transport mechanism or the ciphertext for the session. In this case, it may not be feasible to ensure the flow of the key recovery block within the session from the sender to the receiver since the key recovery block is not really needed for data decryption. Consider a variation of the SKMF scheme, where the key recovery block is an integral part of the key transport mechanism, such that the actual key used to decrypt the data is a function of the key recovery block. In this case, there is a very tight binding between the key recovery block and the session data, but the inter-operability scenarios (b) and (c) become difficult, since the non-key recovery systems will not know how to reconstruct the decryption key using the key recovery block.

8. Conclusions

Cryptographic key recovery enablement is fast becoming a required feature for internationalization of strong cryptographic products. The most flexible, modular and uniform approach to integrating key recovery enablement operations with cryptographic operations is to adopt a framework based approach, such as the SKMF. The elegance of the SKMF architecture arises from its innate flexibility in terms of support for various kinds of cryptographic service providers and key recovery service providers, while allowing a wide range of communication protocols to be implemented in the application layer code and supporting any available public key infrastructure.

References

[1] Schneier, Bruce**,** “Applied Cryptography”, 2nd Edition, John Wiley and Sons, Inc, 1996.

[2] Denning, Dorothy E. and Branstad, Dennis, “A Taxonomy for Key Escrow Encryption Systems”, Communications of the ACM, Vol 39, No. 3, March 1996.