Secure Key Recovery

Rosario Gennaro
Paul Karger
Stephen Matyas
Mohammad Peyravian
Allen Roginsky
David Safford
Michael Willett
Nev Zunic

IBM Cryptography Center of Competence

IBM Thomas J. Watson Research Center

ABSTRACT

A two-phase cryptographic secure key recovery (SKR) system is presented. In the first phase, the sender establishes a secret value with the receiver. For each key recovery agent, the sender generates a key-generating value as a one-way function of the secret value and encrypts the key-generating value with a public key of the key recovery agent. In the second phase, performed per cryptographic session, the sender generates for each key recovery agent a key-encrypting key as a one-way function of the corresponding key-generating value and multiply encrypts the session key with that key-encrypting key. The encrypted key-generating values and the multiply encrypted session key are transmitted with the encrypted session. To recover the secret value, the encrypted key-generating values and public recovery information are presented to the key recovery agents, who decrypt the key-generating values, regenerate the key-encrypting keys, and provide the regenerated key-encrypting keys. The recovering party uses the key-encrypting keys to recover the session key. Since the key-generating values cannot be derived from the key-encrypting keys, they may be used over multiple cryptographic sessions.

Additionally, a parameter validation scheme based on the Diffie-Hellman key exchange is presented which improves the verification of key recovery information. Each communicating party has its own Diffie-Hellman key pair, as does each recovery agent. The sender non-interactively generates a first shared Diffie-Hellman key pair comprising a first shared secret value, shared with the receiver but not with any recovery agent, and a corresponding public value. For each recovery agent, the sender then non-interactively generates an additional shared secret value, shared with the receiver and the recovery agent, from the first shared secret value and the public value of the recovery agent. The sender uses the additional shared secret value as a symmetric key to encrypt recovery information for each recovery agent, which is transmitted with the session data. Each recovery agent can decrypt its recovery information. The receiver can verify the correctness of the recovery information for each recovery agent by decrypting the information using the additional shared secret value for that recovery agent, without having to recreate the recovery information or perform computationally expensive public key operations. Use of Elliptic Curve techniques further improve performance by reducing the public-key key length needed for a given level of security.

KEYWORDS

- Confidentiality
- Encryption
- Key escrow
- Key recovery
- Security
- Public key
- Cryptography

CONTENTS

1. Introduction

Various cryptographic key recovery systems have been proposed as a compromise between the demands of communicating parties for privacy in electronic communications and the demands of law enforcement agencies for access to such communications when necessary to uncover crimes or threats to national security. Generally, in such key recovery systems, all or part of the key used by the communicating parties is made available to one or more key recovery agents, either by actually giving key portions to the key recovery agents (in which case the key portions are said to be "escrowed") or by providing sufficient information in the communication itself (as by encrypting the key portions) to allow the key recovery agents to regenerate the key portions. Key recovery agents would reveal the escrowed or regenerated key portions to a requesting law enforcement agent only upon presentation of proper evidence of authority, such as a court order authorizing the interception. The use of multiple key recovery agents, all of which must cooperate to recover the key, minimizes the possibility that a law enforcement agent can improperly recover a key by using a corrupt key recovery agent.

Key recovery systems serve the communicants' interest in privacy, since their encryption system retains its full strength against third parties and does not have to be weakened to comply with domestic restrictions on encryption or to meet export requirements. At the same, key recovery systems serve the legitimate needs of law enforcement by permitting the interception of encrypted communications in circumstances where unencrypted communications have previously been intercepted (as where a court order has been obtained).

In addition to serving the needs of law enforcement, key recovery systems find application in purely private contexts. Thus, organizations may be concerned about employees using strong encryption of crucial data where keys are not recoverable. Loss of keys may result in loss of important data.

An important aspect of key recovery systems is the method used to ensure that correct recovery information is provided. If the recovery information provided is not correct, either through unintentional error, or deliberate attempt to conceal, the functionality of the key recovery system can be thwarted. Validation can be provided in several ways, including direct checking by the participants, checking by the trustees, and checking by the recovery entity. Correctness can also be ensured by redundant calculation and disclosure of the recovery information by more than one of the communicating parties.

Verification methods should have several desirable properties. The verification method should not require communication with parties external to the given communication, either at installation, or during message creation and transmission. The method of verification should ensure correctness even if some (but not all) of the communicating parties deliberately attempt to provide incorrect recovery information. The verification method should minimize the required computations, and transmission overhead associated with the verification.

Key recovery systems of various types are described in a paper by Denning and Branstad [1]. This is a recent paper that contains a systematic analysis of various known key escrow methods.

In 1993 the US government proposed a technology that would allow it to reconstruct keys it would need to get access, as necessary, to a variety of communications. This technology is based on the so-called Clipper chip. Several papers (e.g., [4,5]) provide the technical details and the applications of the Clipper technology and its effect on the telecommunications industry. The US government regulations on the commercial key recovery schemes are addressed in [8]. An international perspective may be found in [11].

Among other key recovery schemes is Micali's “fair public key cryptosystems,” see [9] for its description and [3] for an analysis of this algorithm. Other methods, also different from the Clipper chip can be found in [6,7].

These government and commercial key recovery schemes were studied in recent years and some attacks were already found against them (see, for example, [3,10]).

The joint US government and industry initiatives to develop key escrow standards and products are described in [2].

In this paper we describe an alternative method of key recovery which will be referred to as Secure Key Recovery (SKR). The proposed key recovery system permits a portion of the key recovery information to be generated once and then used for multiple encrypted data communications sessions and encrypted file applications. In particular, the portion of the key recovery information that is generated just once is the only portion that requires public key encryption operations. Hence, SKR provides a way of reducing the costly public key encryption operations required of other key recovery methods.

We also describe a verification mode in which the communicating parties each produce SKR recovery information independently, without checking the other's so produced information. In this mode, if at least one side is correctly configured, all required recovery information is correctly produced. In addition, the communicating parties are free to include any optional recovery fields without causing a false invalidation of what the other parties sent.

Further, we present a method of verification of key recovery information within a key recovery system, based on a variation of the three-party Diffie-Hellman key agreement procedure. Without communication with a trustee, the sender is able to encrypt recovery information in such a way that both the receiver and the respective trustee can decrypt it. This reduces the number of encryptions, and inherently validates the recovery information when the receiver decrypts it. The method is compatible with interactive and non-interactive communication, and can preserve perfect forward secrecy. The method allows full caching of all public key operations, thus further reducing computational overhead.

2. Key Recovery Environment

The key recovery system described in this paper will be referred to as Secure Key Recovery (SKR). A key for which SKR key recovery information has been created will be called an SKR- protected key. An SKR-protected key can be any type of secret key, although the discussion will be limited primarily to data keys. Data keys are keys used to encrypt messages transmitted within sessions or within e-mail applications or keys used to encrypt data files. A policy option provides that a user may reveal only a part of a given SKR-protected key.

Communications can be among two or more parties, although the discussion will be limited primarily to two- party communications. The SKR protocol can be easily extended to three or more parties. The SKR protocol supports session communications, wherein a sender and receiver interact in real time, and store-and-forward communications such as electronic mail and file transfer applications, wherein the sender and receiver do not interact in real time. Also, the SKR protocol supports file applications involving a single user or application. In this latter case, the application or user who creates a data file is the same application or user who retrieves the data file.

Figure 1 shows a communication system in which SKR can be used. A sender ("Alice") in country X communicates with a receiver ("Bob") in country Y by transmitting one or more encrypted messages (making up a communications session or electronic mail application) or transmitting one or more

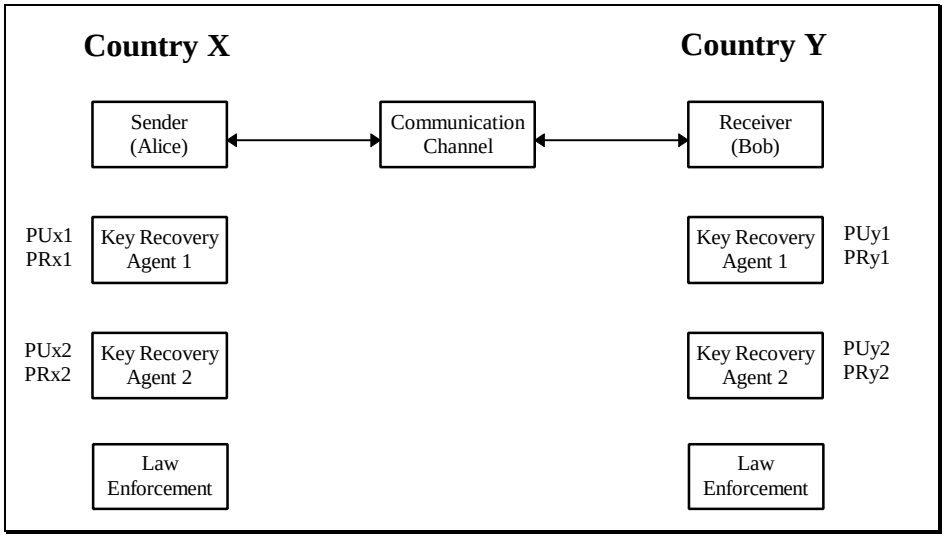


Figure 1: SKR communication system model

encrypted files (making up a file transfer application) over a communication channel. The sender and receiver may each comprise computer workstations, suitably programmed to provide the encryption and key recovery functions described below. Although the example of Figure 1 assumes that the sender and receiver are located in different countries (X and Y), the scheme may also be used entirely within a single country.

The example of Figure 1 specifically addresses the case where there are two different communicating parties (Alice and Bob). However, this two-party model also describes (to a degree) the one-party model in which Alice encrypts files. In that case, the file encryption application may be thought of as a store-and-forward application, where Bob = Alice and a long time exists between the file sending (Alice acting as the sender) and file receiving (Alice acting as the receiver) operations. However, obvious differences affecting the protocols in each of these cases will be pointed out.

The transmitted messages are encrypted by the sender using a data encryption key and decrypted by the receiver using a corresponding data decryption key. If a symmetric encryption scheme (e.g., DES) is used, then the data encryption key (which is also the data decryption key) is the "secret" data key for the purposes of key recovery (i.e., the SKR-protected data key). However, the SKR system is not limited to the protection of data keys. The SKR system can also be used to provide key recovery services for any secret key. For example, if an asymmetric encryption scheme (e.g., RSA) is used, then the private part of the key is the "secret" SKR-protected key for the purposes of key recovery.

A pair (in this particular example) of key recovery agents are selected in country X, while a pair of key recovery agents are selected in country Y. The establishment of key recovery agents could take place as part of the establishment of a general public-key infrastructure. Each key recovery agent has at least one public and private pair of keys (denoted PU, PR). The public and private parts of the key for Alice's two key recovery agents in country X are (PUx1, PRx1) and (PUx2, PRx2) and for Bob's two key recovery agents in country Y are (PUy1, PRy1) and (PUy2, PRy2). This enables Alice and Bob to use public key cryptography (e.g., RSA, Diffie-Hellman, or Elliptic Curve algorithms) to encrypt and protect certain SKR information.

Communications are assumed to be subject to interception by third parties, including law enforcement agents in countries X and/or Y. Non-governmental third parties intercepting the encrypted communications will be unable to decipher the communications unless they successfully use one or more cryptanalytic techniques. On the other hand, a law enforcement agent having proper authority will be

able to recover an SKR-protected key, using the key recovery agents for his country as described below, and the sender (Alice) or receiver (Bob) will also be able to recover an SKR-protected key using a similar procedure.

3. Two-Phase Protocol

The SKR system calls for a two-phase process of embedding key recovery information within encrypted communications or encrypted files, consistent with the laws and regulations of the country or countries where the encrypted communications take place or encrypted files are produced.

Figure 2 shows the two phases of the SKR Communication Process. In Phase 1, the communicating parties (Alice and Bob) establish a common random seed (S), from which random key-generating keys (KG) specific to each recovery agent (not shown in Figure 2) are derived. The KGs are encrypted using the public keys of the respective agents. The encrypted KGs are included in an SKR Block 1 (B1) prepared by Alice, which she sends to Bob.

The manner in which the random seed (S) is established depends on the communications environment. In an interactive session, Alice and Bob could establish S interactively (e.g., using a Diffie-Hellman key exchange protocol), or Alice could generate S and send it to Bob (e.g., using an RSA key transport protocol). In store-and-forward applications, Alice would send S to Bob (as mentioned above), although it would be possible for Alice and Bob to set up a special session, allowing them to establish S interactively.

In Phase 2, the communicating parties (Alice and Bob) establish a secret data key K. Any key transport or key agreement mechanism can be used to establish K, as the means by which K is established is independent of the SKR system. The data key (K) is encrypted in a nested fashion using a symmetric algorithm and the key-encrypting keys (KK) derived from the stored KGs (via a hierarchical key derivation procedure). If allowed, only part of the data key K is made available in the recovery information, by holding back a portion, R, of the encrypted K. The encrypted K, or portion thereof (if a portion R is retained), is included in an SKR Block 2 (B2) prepared by Alice, which she sends to Bob. For file encryption applications, where Bob = Alice, Alice would merely generate and store K, and the encrypted K would be made available by including it in an SKR Block 2 prepared and stored by Alice, until a later time when it may be needed by the recovery process (i.e., it is sent from Alice to Alice with a long time delay). Finally, the data encrypted with K is communicated or stored in a file.

The division of the key recovery preparation process into two phases allows the separation of the "Establish S" operation and the associated public key encryptions required in the preparation of "SKR Block 1" (Phase 1) from the per session exchange of K and the associated calculation and verification of

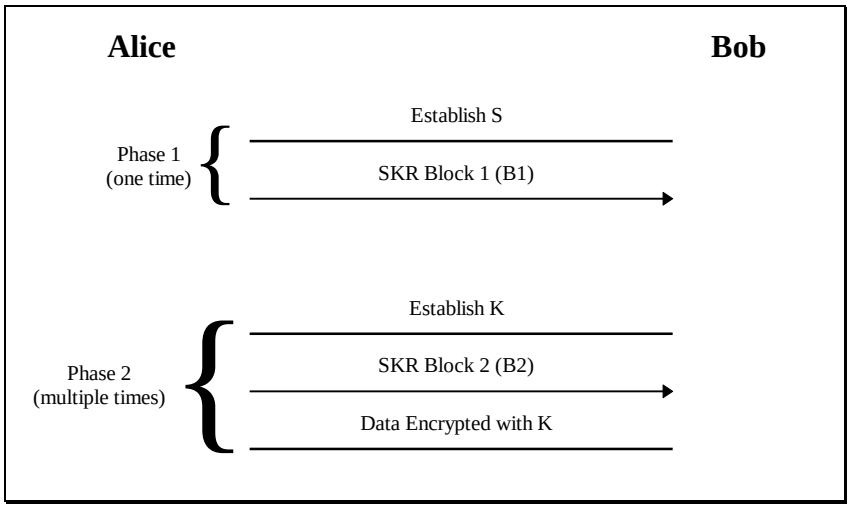


Figure 2: SKR communication process

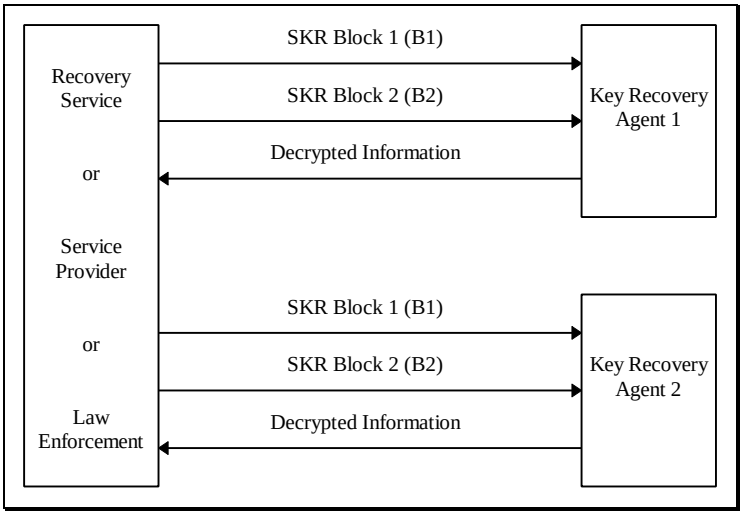


Figure 3: SKR key recovery process

the session key recovery information (Phase 2). The information provided in Phase 1 (i.e., S and SKR Block 1) is not in any way based on the per communication or per file information, and can therefore be used across multiple communications or created files. If the information (S and SKR Block 1) is cached, the expensive public-key encryptions done in Phase 1 can be avoided in all but the first communication between parties or first created file, thus greatly reducing the computational effort needed.

Figure 3 shows the SKR key recovery process that allows SKR-protected keys to be recovered. A Recovery Service interacts with Key Recovery Agents 1 and 2 to recover a key K. The Recovery Service may be operated by a independent Service Provider party on behalf of users or companies, who want to recover damaged or lost keys, or it may be operated by law enforcement on behalf of law enforcement, who want to recover keys in order to read intercepted communications or files of individuals known or suspected of engaging in criminal activities.

A party who wants to recover a key first provides the needed SKR Blocks (1 and 2) to the Recovery Service. Recovery Service then provides SKR Block 1 and part of SKR Block 2 to Key Recovery Agent 1. In turn, Key Recovery Agent 1 uses the provided information together with the private part of his key to decrypt the encrypted KG value, and then to calculate a key-encrypting key (KK) from KG, which is returned to the Recovery Service in the "Decrypted Information". Likewise, the Recovery Service provides SKR Blocks 1 and part of 2 to Key Recovery Agent 2. In turn, Key Recovery Agent 2 uses the provided information together with the private part of his key to decrypt the encrypted KG value, and then to calculate a key-encrypting key (KK) from KG, which is returned to the Recovery Service in the Decrypted Information. The nested encrypted value of K in Block 2 is withheld from the Key Recovery Agents. In that way, the Recovery Agents themselves cannot recover K. The two key-encrypting keys are then used by the Recovery Service to decrypt the encrypted K in SKR Block 2, and the clear value of K is returned to the requesting party.

Law Enforcement can obtain a key K only if it has a valid warrant or court order that can be provided by the Recovery Service to Key Recovery Agents 1 and 2. Likewise, a user or company can only recover K if it provides the correct authentication information to the Recovery Service, who in turn provides the authentication information to each key recovery agent. Each key recovery agent uses this authentication information to validate the user's or company's right to recover K.

3.1 SKR Top-Level Design for Interactive Communication

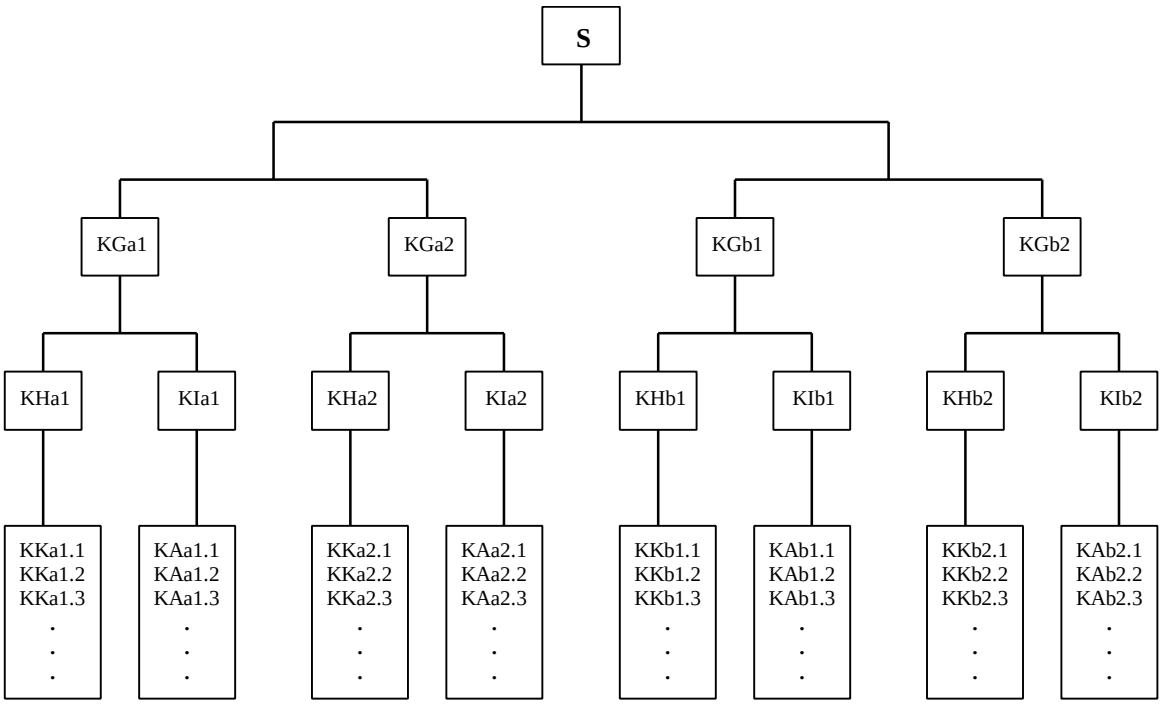
Nominal data flows are shown for a case assuming one sender (Alice), one receiver (Bob), and two recovery agents for each. In addition, the description assumes the case of an interactive session with an existing mechanism for the exchange of S and K in a secure fashion. SKR works as well for cases of any number of communicating parties, any number of agents per party, and non-interactive as well as interactive communication. SKR can also provide a default key exchange mechanism as needed (i.e., where S and K can be established using mechanisms provided by SKR). For simplicity, only the nominal case is described here, and other cases will be covered in the detailed design discussion.

3.1.1 Key Generation Hierarchy

SKR is largely an exercise in symmetric key management. Symmetric keys are used to protect recovery information for a given SKR-protected data key (session key), and optionally, to protect recovery information used to authenticate recovery requests related to a given SKR-protected data key (session key).

Figure 4 shows the SKR key generation hierarchy. The SKR key hierarchy is an organization of keys into levels according to their uses and derivational relationships. It is the basis on which the functions of the SKR system are organized.

The keys in the SKR key hierarchy are generated in a hierarchical fashion from an initial random secret seed, S. From S, pseudo-random key-generating keys KG are generated, one per key recovery agent,



Legend:

S	Secret Seed
KGij	Key-Generating Key for Alice or Bob's (i = Alice, Bob) jth key recovery agent (j = 1, 2) used to generate KH and KI keys
KHij	Key-Generating Key for Alice or Bob's (i = Alice, Bob) jth key recovery agent (j = 1, 2) used to generate key-encrypting keys (KK)
KIij	Key-Generating Key for Alice or Bob's (i = Alice, Bob) jth key recovery agent (j = 1, 2) used to generate key-encrypting keys (KA)

Figure 4: SKR key generation hierarchy

which are associated with a given agent by encryption under the public key of the agent. KGa1 and KGa2 are the KGs associated with Alice's 1st and 2nd key recovery agents, respectively. KGb1 and KGb2 are the KGs associated with Bob's 1st and 2nd key recovery agents, respectively.

The KG key-generating keys are in turn used to derive subordinate pseudo-random key-generating keys, KH and KI. For example, KGa1 is used to generate KHa1 and KIa1, KGa2 is used to generate KHa2 and KIa2, and so forth. Finally, the KH key-generating keys are used to derive subordinate sets of pseudo-random key-encrypting keys KK. In like manner, the KI key-generating keys are used to derive subordinate sets of pseudo-random key-encrypting keys KA. The key-encrypting keys KK are used to encrypt/decrypt the data key (i.e., the session key protected by SKR), where a unique KK is used for each session. The key-encrypting keys KA are used for the protection of optional authorization information (AI) fields. However, the SKR system does not limit the use of the KA keys to only key-encrypting keys, and other ways of using the KA keys are described below.

One method of deriving the keys in the SKR Key Hierarchy is to make use of the Cryptographic Key Derivation procedure described in Working Draft American National Standard X9.42.

The example of Figure 4 illustrates the case of two sets of agents, one set for Alice (a) and one set for Bob (b), where each set contains two agents (1 and 2). In the general case, there may be any number of sets of agents (a, b, c, etc.) and any number of agents in a given set (1, 2, 3, etc.).

Although S is specified here as a random number, the SKR system does not limit S to a random number. Additionally, the SKR system does not limit nor dictate the minimum or maximum number of sessions that may be protected by a single S. Moreover, it does not prescribe how S can be established or calculated or the particular size of S or the information that may or may not be used in determining the value of S.

3.1.2 Phase 1

Phase 1 must occur at least once before the start of one or more encrypted communication sessions. In the typical case, it could be done once before the start of the first session between parties, and the information reused for subsequent sessions between the parties, until some time at which phase 1 is repeated. If desired, phase 1 can be performed for every session, although this eliminates the performance benefit of reusing the public key operations.

In phase 1, Alice and Bob exchange a random value S, using an external key exchange mechanism. If no exchange mechanism is available, Alice can simply pick a random S, and have SKR transport it to Bob as part of the phase 1 SKR data block. Alice uses this S to generate pseudo- random KG values, one for each of the recovery agents needed for both Alice and Bob, as depicted in Figure 4. These KGa1, KGa2, KGb1, and KGb2 values are encrypted under the public keys of the respective key recovery agents, and placed in the SKR phase 1 data block (B1) that is sent to Bob. Both Alice and Bob may cache at least S along with a hash of B1 for future use.

A simplified data flow of Phase 1 consists of:

- Alice and Bob exchange a random S.
- Alice derives from S a value KG for each agent, by hashing S and the respective agent's ID.
- Alice encrypts the KG values under the respective agents' public keys.
- Alice sends to Bob the SKR phase 1 data block B1 composed of: T1, ePUa1(KGa1), ePUa2(KGa2), ePUb1(KGb1), and ePUb2(KGb2).

where the above quantities are defined as follows:

T1	a public header containing IDs of Alice, Bob, and all key recovery agents.
e	public key encryption
PUai	public key for Alice's ith agent
PUbi	public key for Bob's ith agent
KGai	key-generating key for Alice's ith agent
KGbi	key-generating key for Bob's ith agent

3.1.3 Phase 2

Phase 2 occurs prior to the start of encryption for each encrypted session. In phase 2, Alice and Bob exchange a data key K (session key) using some external key exchange mechanism. If needed, Alice can pick a random K and the SKR phase 2 data block can transport it to Bob. For each set of agents, Alice nested encrypts K under the respective set of key encrypting keys (KK), and sends these values to Bob in the phase 2 data block, along with a hash of the corresponding phase 1 block (B1).

A simplified data flow of Phase 2 consists of:

- Alice and Bob exchange a random data key K (session key).
- Alice derives a key generating KH value for each agent as the hash of the respective KG value, and a public key-generating header.
- Alice derives a session-specific KK for each agent as the hash of the respective KH value and the session's public header T2.
- Alice nested encrypts K under the sets of session-specific KK key-encrypting keys.
- Alice sends to Bob the SKR phase 2 data block (B2) composed of: T2, Hash(B1), fKKa1.1(fKKa2.1(K)), and fKKb1.1(fKKb2.1(K)).

where the above quantities are defined as follows:

T2	a public header containing T1, session ID, timestamp, etc.
f	symmetric key encryption
Hash(B1)	hash of block B1
KKai.1	first key-encrypting key for Alice's ith agent
KKbi.1	first key-encrypting key for Bob's ith agent

3.1.4 Key Recovery Phase

In a recovery phase, a set of agents are given all of block 1, and T2 from block 2. They are not given the doubly encrypted fKK1(fKK2(K)), so they are not able to recover the key, even if they pooled their respective KG information. Given the appropriate SKR information, they decrypt their respective blocks, obtaining their respective KG values. Given their KG values, and T1 and T2 headers, they can generate the KK and KA values for the session covered by the T2 header. If an authorization information field (AI) exists, the secret KA key may be used to validate the recovery request. If a recovery request is valid, each agent returns only the calculated KK value to the Recovery Service, which is then able to recover the data key K (session key) from the doubly encrypted recovery field fKK1(fKK2(K)).

Importantly, an agent does not return the generating KG value, but only specific values derived from KG. For example, an agent would only return specific key-encrypting keys (KK) to Law Enforcement. Thus, the longer-term secret KG information is not compromised, and can remain in effect, as other data keys (session keys) are not recoverable, given one KK value.

A user and agent may decide that the user wants the agent to return a KH if the user provides sufficient AI validation, so that the user may more conveniently recover a set of sessions. This decision is left up to the application, user and agent. However, the SKR system does not contemplate a set of circumstances that would demand an agent to divulge KG to anyone, as the separation of knowledge between the recovery service and the agents is a fundamental feature of SKR, and it is important to the secure reuse of the public key encryptions of the KG secrets.

4. SKR Design Details

SKR is designed such a way to allow sufficient flexibility and options for cases involving interactive or non-interactive communications. The following subsections discuss SKR design details.

4.1 Selection of Encryption Algorithms Within SKR

SKR is independent of the session-level encryption used between Alice and Bob. SKR itself, however, uses both symmetric and asymmetric encryption to calculate the values within the SKR blocks. The encryption algorithms used within SKR may also be freely chosen, and the choices are indicated as parameters within the T1 and T2 headers. By default, RSA is used for public key encryption, and a keyed shuffler (described below) is used for symmetric encryption. The keyed shuffler is used by default, as it has the useful property of accepting arbitrary size encryption keys, and arbitrary size data blocks. Another possible choice for the symmetric encryption used within SKR would be to use the same encryption algorithm as used by the application for the session level encryption. This would offer the advantage of availability, as the application has already implemented it, and therefore the symmetric key encryption within SKR would have the same strength as the encryption used to protect the session data.

4.2 Recovery Validation Hash and R

In the top-level design, the simplified data flow showed the transmission of the entire value of the double encryption of the data key K. If a part R is to be held back from the recovery field, this is done by removing R from the resultant encrypted K, as in: fKKa1.1(fKKa2.1(K)) - R. For example, if the session key K is 168 bits (e.g., triple DES), and the part R to be held back is 40 bits, then the least significant 40 bits of the doubly encrypted K are removed, and only the most significant 128 bits are included in the SKR block.

To be able to validate a trial R value during the recovery process, additional recovery validation fields Ha and Hb (one per set of agents) are added to the phase 2 SKR block. Ha and Hb are defined to be:

- Ha = Hash(validation header, counter, fKKa1.1(fKKa2.1(K)), T2, KKa1.1, KKa2.1)
- Hb = Hash(validation header, counter, fKKb1.1(fKKb2.1(K)), T2, KKb1.1, KKb2.1)

where Hash() is the hash of the concatenation of the listed data items.

The hash is performed as many times as necessary, with an incrementing counter, so the validation hash is long enough to ensure that the probability of collision (i.e., a false validation of R) is sufficiently small. If r represents the number of bits in R and the length of Ha is k, then for the collision probability to be less than 2^{-t}, then k has to be greater than or equal to $2r + t - 1$.

With these added fields, trial values of R can be used to construct trial values of fKKa1.1(fKKa2.1(K)), then the hash taken, and compared to see if the trial R was correct. Note that this search cannot proceed unless both agents in a set have decrypted their respective encrypted KG fields from the phase 1 SKR block, calculated the respective KK key-encrypting keys (e.g., KKa1.1 and KKa2.1), and returned them to the recovery process.

4.3 Non-Interactive Communications

The design presented for SKR describes an interactive session between Alice and Bob. In the more general situation, a communication may be non-interactive, in which case the processes on both ends may occur at different times, so they cannot exchange information during the execution of the protocol. Examples of non-interactive communications include (1) e-mail, where the composition of the mail by Alice, and the reading of the mail by Bob occur at different times, and (2) data archiving, in which the archiving of data by Alice is done at one time, and the retrieval or the archived data (conceivably also by Alice) is done at a later time.

The SKR design handles non-interactive communications without modification. As non-interactive communications cannot exchange information during the execution of the SKR protocol, this may restrict the methods used for the exchange of the S and K keys, but this nominally occurs outside the SKR process. If SKR is used to exchange S and K, it uses a method suitable for either interactive or non-interactive situations (i.e., encryption under the public key of the recipient).

4.4 Validation of SKR blocks 1 and 2

The design presented so far does no validation of SKR blocks 1 and 2. While this may be acceptable for some applications, in the general case, some applications may wish to validate the recovery information in some manner, so that both sides of a conversation can ensure that proper recovery information has been provided.

One method of validation is for Alice to generate the SKR blocks as described, and for Bob to validate them. Validation by Bob is straightforward, as Bob knows the secret values S and K, and can therefore duplicate the creation of the blocks, and verify his results against the values sent from Alice.

One problem with this method is that Alice must have complete knowledge of Bob's recovery requirements, including any optional recovery fields he may want to include. In addition, Bob would either need to know Alice's information in advance, or wait to see the T1 and T2 header information before he could start validation, thus doubling the SKR startup delay.

A different, and possibly preferred method for ensuring that all recovery information is correctly provided is for Alice and Bob to each produce SKR recovery blocks independently and for Alice and Bob not to check each other's blocks. In this mode, if at least one side is correctly configured, all required recovery information is correctly produced. In addition, both Alice and Bob are free to include any optional recovery fields without causing a false invalidation of what the other sent. One limitation of this method is that it cannot be used in non-interactive communications.

SKR does not dictate which method of generation-validation is used. In all cases, Alice creates SKR blocks. A given application may have Bob check Alice's block, produce his own blocks, or do no checking, depending on the application's needs, and possibly local laws or regulations.

In addition to varying the method of validation, the level of checking may also vary. In those cases in which Bob is validating Alice's SKR blocks, he may do complete validation of all entries in the blocks (including validating the parts encrypted for the agents), or he may check only parts of the messages. For example, he may not want to validate the public key-encrypted parts, due to the computational effort involved in public key encryption, but he could still validate all other parts. While this would not guarantee that Alice had performed the encryptions correctly, it would validate that her selection of agents and their respective R values was correct. This type of partial checking could be important for applications sensitive to communication latency -- if Alice is able to precompute the SKR block, and Bob does only partial checking, the startup latency would be significantly faster, as no public key operations would be involved, and Bob would still be validating some important parts of Alice's SKR process. This is particularly useful when both Bob and Alice are generating independent SKR information, as partial validation would provide greater probability of proper recovery (e.g., that the other's system is configured properly and operating properly), without the additional time needed for full validation of the included public key encryptions.

4.5 Additional Caching

In the design presented so far, Bob and Alice are able to cache the public key encrypted KG values derived from their shared secret seed S. In the general case, these encrypted KG values can only be reused in subsequent conversations between Alice and Bob, but not with other parties, as S is known only to Alice and Bob, and S must be known for another receiver to fully validate the recovery information.

The encrypted KG values may be reused in conversations with other parties under two different conditions. First, they can be reused if the value of S is explicitly transmitted to the new party. Second, they can be reused if either partial validation of B1, or no validation of B1 is to be done by the receiver. For example, if Alice creates a set of encrypted KG for her agents at the start of a session with Bob, she could reuse these encrypted KG for a conversation with Ted, if either she sends the S to Ted, or Ted does not use S to validate the encrypted KG values in the transmitted B1. The only complication of the nominal data flow in phase 1 to support such reuse is to recognize that a given encryption of KG also encrypts the hash of a given T1, so if the encryption is reused in a new phase 1 block, (which has a new T1), the old T1 would also need to be sent with the encrypted KG field, so that the agent could later match the encrypted hash of T1 to the old T1.

In the preferred case, Alice and Bob have an interactive session, each generate the SKR blocks independently, and do only partial checking of each other's SKR blocks. In this situation, Alice and Bob do not have to exchange S at all, and are free to reuse an S that was previously calculated, possibly even from sessions with other parties. If the agents involved are the same, the reuse of S would avoid additional public key encryptions. As Alice may communicate with many parties with the same set of recovery agents, the ability to reuse encrypted KG for a given set of agents offers a significant further reduction in the computation required for the public key operations used to encrypt each agent's KG value.

In the general case, public key encryption can be expressed in the form

$c = K(D, X)$

where the cipher text c is derived as an encryption under key K of the data D and random value X. Any one of several public key encryption algorithms can be used, including RSA PKCS-1 encryption, RSA

encryption with enhanced optimal asymmetric encryption (EOAE), and Diffie-Hellman encryption. The first and last of these algorithms are well known.

RSA encryption with enhanced optimal asymmetric encryption (EOAE) is an encryption procedure in which a formatted block (comprising data plus appended values) is subjected to a plurality of masking rounds (similar to those of the keyed shuffler described below except that the hashes are not keyed) before encrypting the result of the masking rounds using RSA encryption. The procedure is described in the ANSI X9.44 RSA Key Transport draft standard and in [12].

For RSA PKCS-1 encryption, the X is used to seed a random number generator used for random padding of the data up to the block size. For RSA encryption with enhanced optimal asymmetric encryption (EOAE) (described below), the X corresponds to the salt appended explicitly to the data, as no other random padding is done. In the case of Diffie-Hellman encryption, X can represent the random value used to generate Alice's ephemeral key pair, which along with the agent's public Diffie-Hellman key, is used to generate the common key, which is then typically used as a mask for the data.

For example, if Alice wishes to send KGa to agent “a” using Diffie-Hellman, she would use X to generate an ephemeral Diffie-Hellman key pair, {KPx, KSx}, where KPx is the public part, and KSx is the secret part. Given the public key part of agent “a” (KPa), Alice would calculate the common key KC = KSx to the power KPa. The cipher text c would now be the compound set {KPx, KC(M)} which is her ephemeral public key, and the original data D encrypted by the common key KC. (This encryption may be simply using KC to derive a mask which is XORed with D, or a more general symmetric encryption method using KC as the key.) Given {KPx, KC(D)}, the agent would calculate KC = KSa to the power KPx, and would use this to decrypt D.

In summary, Diffie-Hellman can easily be used in SKR, so long as the random X is used to create the ephemeral key pair, and the ephemeral public key KPx is included as part of the cipher text.

In all of these cases, Bob may need to know the value X to be able to reproduce, and thus validate, the public key-encrypted fields within the SKR phase 1 block. To enable Bob and Alice to agree on the random salts used, the salts are pseudo-randomly derived from the exchanged S, which is also used as the generation source of the KG values. Thus S is the source of all necessary randomness used in the SKR process. Specifically, the KG and salt values are generated as:

- KGx = Hash(KG generating header, counter, T1, agent x's ID, S)
- saltx = Hash(salt generating header, counter, T1, agent x's ID, S)

where

- Hash(x) denotes a hash of x.
- KG generating header and salt generating header are defined constants.
- counter is a simple integer counter.
- T1 is the public header containing IDs of Alice, Bob, and all key recovery agents.
- S is the exchanged random S.
- Hash(T1) Binding in e: The detailed design of the encrypted KG values is: ePUa1(Hash(T1), KGa1, salt1)

The data block encrypted with PUa1 (above) shows the inclusion of both a hash of the public header T1 and a salt. The inclusion of the hash of T1 binds the KG value to a specific header, so that the agents can associate the decrypted KG with the indicated set of parties and agents as specified in the public header T1. The inclusion of a salt provides further protection of the encrypted KG value.

4.7 Authorization Information

Alice and Bob may optionally include key recovery Authorization Information (AI) in the SKR information (e.g., the phase 2 SKR block). For practical purposes, the AI information in the SKR system includes the defined uses of the self-identification data, as well as the ways for using AI described in this paper.

The AI information (identification information, authentication information, or authorization information) is optionally used to validate possible subsequent key recovery requests. The SKR framework allows the exact content of such recovery authorization records to vary depending on the needs of the application and user. Each SKR phase 2 block defines values (KAa1.1, KAa1.2, etc.), derived from an agent's KI. These KA values are used in some way to protect the AI data. In the nominal case, the AI information is simply hashed with the respective KA key, as in a keyed hash of the form Hash(AI, KA, T2). In this case, Alice can validate a recovery request by providing AI (typically a passphrase) to the agent, who can then validate the hash, and thereby validly ask for the return of the corresponding KK. In this example, KA is used as a key in a seeded hash operation.

Other methods may be used to protect the AI data. For example, Alice could use a passphrase to encrypt the KA, as in AI(KA, T2). At recovery time, Alice could use the passphrase to decrypt the field to recover KA, and submit for recovery authorization. In this case, KA is used as a codeword or authorization token. As only Alice, the agent, and in some cases Bob (if Alice and Bob share S) know KA, it protects Alice's passphrase from forward attack. In this mode, Alice has the added benefit of not having to reveal her passphrase to the agent.

By default, the AI information is included in block 2 in plaintext in the form:

- Hash(authorization header, counter, AIa1, KAa1.i, T2 for i-th session),
- Hash(authorization header, counter, AIa2, KAa2.i, T2 for i-th session),
- Hash(authorization header, counter, AIb1, KAb1.i, T2 for i-th session),
- Hash(authorization header, counter, AIb2, KAb2.i, T2 for i-th session)

The AI may be the same for all agents and sessions, across agents and session, or combinations thereof as desired. The values of AI may simply be some secret password or passphrase, although users and agents may agree on any standard contents.

In one embodiment of the SKR system, Bob validates the encrypted KG values from Alice, in which case the encrypted values cannot contain AI. Therefore, the SKR system requires a method of binding AI and KG that will operate in all embodiments, including the case where Bob validates the encrypted KG from Alice. To allow this, we first derive KI and thus KA from KG, so that the secret used in the coupling operation is not KG, thereby not exposing KG repeatedly in the authentication code generation process. Then AI and KA are combined, possibly with other information, and the combination is hashed to produce an authentication code. Alice is able to create the authentication code, since she knows S and can therefore generate any of the keys in the key hierarchy, and she knows her own AI, which may contain secrets.

To allow the key recovery agent to validate the Hash(AI, KA, T1), Alice must provide AI to the key recovery agent. There are different possible options for doing this, which includes appearing in person and presenting credentials, or using an intermediary who validates a user's credentials, prepares an AI record and sends it securely to the key recovery agent. Alice could prepare her own AI record, encrypt it under a key known to the key recovery agent, and send the encrypted AI to the agent electronically. For example, AI could be encrypted under the public key of the agent and sent to the agent.

By decoupling AI and KG, in the manner described, the AI provided or made available to the key recovery agent can be used repeatedly for a long period of time. Typically, AI would remain constant over many different KG. Also, the AI may contain a large amount of data, and hence may require multiple public key encryptions, in which case it would be disadvantageous to repeatedly send AI together with each KG.

The SKR Authentication Code Generation process is as follows. A user prepares Authentication Information AI, which is protected (e.g., via encryption under a key known to the key recovery agent). The resultant Protected AI is then made available to the key recovery agent. These steps could be accomplished by encrypting the AI with the public key of the key recovery agent and then sending the encrypted AI to the key recovery agent. This step would be performed prior to using the SKR system. Later, a KG is encrypted with the public key of a key recovery agent, producing an encrypted KG, which is made available to a key recovery agent. In this case, the encrypted KG is "made available" by including it in SKR Block 1. For each session, a key derivation step is performed to derive a KA from KG. The so produced KA and the AI and possibly other information (e.g., control information specifying which KK is to be returned in response to a valid key recovery request) are combined (e.g., by concatenating the values) and the combination is hashed to produce an Authentication Code. The Authentication Code is then "made available" to the key recovery agent by including it in SKR Block 2.

As a consequence of the described method of Authentication Code Generation, a user's request for KK is validated as follows: The user must first prove his identity or right to access the requested key by supplying information that the key recovery agent can verify against his copy of the stored AI. Part of the validation step requires the computation of KA. For example, if the user's request is for the i-th KK, then the key recovery agent first calculates the corresponding i-th KA. The key recovery agent next calculates a hash on the input value consisting of (AI, KA, T2), where T2 is obtained from the session recovery information provided by the user. If the calculated hash value matches the value of the corresponding hash-of-reference included in the supplied SKR Block 1, then the request is considered valid, and the requested KK is calculated and returned by the key recovery agent.

4.8 IDs and Namespaces

IDs for users and agents can be of many types, such as X.509 Distinguished Names, PGP ascii names, PGP keyids, IP addresses, DNS host names, e-mail addresses, etc. In SKR, all IDs may be expressed as a tuple consisting of a namespace identifier and an ascii name within the indicated namespace, in accordance with draft RFC draft-ietf-ipsec-cdp-00. This draft reserves namespace indexes from 250 to 255 for application private use. SKR may treat namespace 250 as private to the application, and therefore expect the application to handle all queries with respect to all IDs of this type, such as requests for public keys, requests for country information, etc. As there is no defined namespace for country IDs, SKR may use namespace 251 for SKR-specific country names.

4.9 User Registered Public Keys

In the nominal design, the KG values are encrypted under the public key of the respective agents. This description implies that each agent has only one public key. The SKR data format specifies a keyid used for a given agent, so the agent may have any number of keys. This is an important feature, as the use of a single key would raise concerns about the long-term security of the corresponding private key. By allowing for multiple keys per agent, an agent may simply have a set of keys for all users, or they may have keys specific to a set of users (as those of a particular organization, which may be regarded as the "user"), or they could even have a separate key for each individual user. Users could even generate the key pair, and register them with an agent, ensuring that the keys were well-made, and changing them at a desired frequency.

Any one of several different methods may be used to register user-generated recovery keys with key recovery agents. In one method, each user registers his or her own public and private key pair for use in the key recovery procedure with one or more key recovery agents. Each user then uses the public part of his or her registered key for encrypting key recovery values. With this granularity, users have the flexibility to periodically change the recovery key pair to limit exposure, in case the private recovery key is somehow compromised.

This first method assumes the presence of a public key infrastructure in which key recovery agents have public/ private signature (certification) keys for signing and public/private keys used for encryption and distribution of keys. It also assumes that means exist for the generation and distribution of these keys and that means exist for users to validate the public portion of these keys.

In this first method, users register the private portion of the recovery key pair by encrypting it with the public encryption key of the key recovery agent. This allows the on-line registration of recovery keys with the key recovery agent. As part of the registration process, the key recovery agent validates that the public and private keys are a valid key pair. This way, the key recovery agent knows that it has the correct private key for decrypting information encrypted with the public key.

In this method, the key recovery agent optionally makes use of a signature (certification) key which is made available to users. After validating a user's registered key pair, the key recovery agent signs (certifies) the registered public key with its signature (certification) key. The signed (certified) public key is returned to the user, who validates the signature (certificate) with the public portion of the key recovery agent's signature (certification) key. This way, the user knows that the key recovery agent has the means to decrypt key recovery values, and therefore that the key recovery procedure can be carried out.

Where there are a large number of individual users, it could easily become impractical for every user to register his or her recovery keys with key recovery agents. Therefore, a variant of the first method provides for granularity of recovery keys at the organizational level rather than that of the individual user. Groups of users who belong to a common organization (e.g., the employees of the same company) would have the same recovery key.

In this variant, each organization registers its own public and private key pair for use in the recovery procedure with one or more key recovery agents. For example, a company would generate its own recovery key pair and register these keys with one or more key recovery agents. In this case, the company would generate a first recovery key pair which it would register to a first key recovery agent and it would generate a second recovery key pair which it would register with a second key recovery agent. Company employees would then be expected to use the public portions of the company-generated recovery keys to encrypt key recovery values under the recovery procedure.

Except as noted, this variant is the same as the individual user variant. However, with organization-wide recovery keys, the number of registered (or escrowed) recovery keys is significantly reduced, since the number of potential organizations who might register keys under this variant would be far less than the number of potential users who might register keys under the individual user variant of this first method.

However, a disadvantage of either variant of this first method is that the private portion of the recovery key is made available to the key recovery agent on-line by encrypting it with a key belonging to the key recovery agent. Thus, the advantage of having multiple recovery keys, and spreading the exposure across these multiple keys, is rendered ineffective by encrypting the private portions of these keys under a single key belonging to the key recovery agent.

Therefore, in an alternative method of recovery key registration, the private portion of the recovery key is not registered on-line by encrypting it with a key belonging to the key recovery agent. Instead, the key

registration is implemented in a way that is both provably secure and auditable, so that the users and companies have full confidence in the key recovery agent.

In this alternative method, a user could register a tamper-proof hardware public key token (e.g., a smart card or PCMCIA card) with the agent. This token would be designed to generate its public-key pair based on internal hardware random numbers, and would reveal only the public portion of the pair. It would never reveal the private part of the key, but would instead only use the private part for decryption of any submitted recovery requests. (Alternatively, a private key could be externally generated and stored in the token, or the public portion of the recovery key could be provided in some other manner, if so desired.) Thus the user could be extremely confident that their recovery private key is never compromised, and that any use of the key would be auditable.

The card would be plugged into a reader at the key recovery agent, so that the public key could be retrieved, and so that the card could be challenged to verify that it contains the secret key matching the newly submitted public key. The process of validating that the token contains the correct private part of the recovery key is well-known. It could consist of sending a series of random values to the token and requesting the token to encrypt the values (one at a time) under the secret key. The public key would then be used to decrypt and compare the outputs with the known inputs. If the values compare equally, then the private key on the token is valid. Otherwise, the token is rejected. Once verified, the token would be unplugged and stored in a vault or safe. The token would be retrieved from the vault or safe and used only under a warrant or court order, and even then it would not reveal the private key. It would only decrypt the supplied encrypted key recovery values.

With this alternative method, one has the advantage that the private portion of the recovery key is never exposed by encrypting it with a key belonging to the key recovery agent. In this case, the key is stored on a token and even the key recovery agent cannot determine the value of the private portion of the recovery key.

With this alternative method, one also has the advantage that the tokens can be audited. In this case, the private portion of the recovery key is stored within a physical device that can be inspected for evidence of tampering. It can be marked or tagged and can be periodically audited to ensure that it has not been misused. The token can also record all instances of usage, and this information can be "dumped" during an audit, to ensure that the card information agrees with the audit log of the token.

With this alternative method, an organization can be certain that the key and token will only be called into use by a key recovery agent if one of its employees is the subject of investigation. The recovery of other keys by law enforcement, which belong to non-organization employees, will not require the use of the organization-supplied tokens containing organization-supplied recovery keys. Therefore, one has greater assurance that the recovery keys are protected.

In each of these methods, it is assumed that the key recovery system software or hardware contains a list of public certification keys. These public certification keys could include public certification keys belonging to key recovery agents as well as certification authorities (CAs). In this case, we have a two-level hierarchy in which the public portion of the recovery keys are signed (certified) with a certification key and where the certificates containing the public portion of the recovery keys is validated dynamically using one of the public certification keys "hard coded" into the recovery system software or hardware.

Given this type of facility, users would be able to choose their level of recovery security versus cost. Most users would probably simply use the certified public encryption key of the agents of their choice. For additional cost they could use the on-line method to register their own recovery key periodically. For a higher cost, they could register their key physically on a smart card. The higher costs associated with audited physical storage of a smart card would probably limit its use to large corporations, or businesses with exceptionally valuable communications, but the users would be able to choose both the level and frequency of key update.

Recovery agencies would probably welcome registration by smart cards, as the auditable physical security provided would make it difficult for a user to claim that their key was compromised in any way by the recovery agency.

With these alternative methods, a user could choose from a security spectrum which trades the inconvenience of some form of prior registration with the agent for greater security.

4.10 Hash Properties and Key Generation

SKR uses a hash for key and salt derivation, for generating a recovery validation hash, for generating recovery authorization information, and as part of the keyed shuffler method of symmetric encryption. Nominally SKR uses SHA-1 for the hash, although any cryptographic hash may be used. The hash mainly needs to be non-invertible, so that knowledge of the hash of a value does not provide a tractable method of determining the original value. In addition, the hash needs to depend on all input bits, so that its use in the recovery validation and recovery authorization is reliable. The hash does not need the stronger property of collision resistance, although collision-resistant hash functions may be used as long as they satisfy the prior two requirements.

In this paper, the hash is used in the form H(a, b, c, ...). With a "good" hash, the order of the input values does not matter, and may be specified in any order. As a practical matter, the order may be significant, particularly with the treatment of a key in the keyed hash. The orders shown in this paper are merely possible orders, and the SKR system is not limited to the specific orders of the inputs in any of the hashing operations.

4.11 Degenerate Case Where S = K

If needed, SKR can transmit either or both S and K. Provided that Bob has a public key pair to perform key distribution, this can be done by encrypting the S or K value under Bob's public key, and adding the encrypted value to the SKR information sent to Bob by including it in a SKR data block.

For some applications it may be difficult or undesirable to exchange S, or to have SKR exchange it in phase 1. For example, if the application cannot support an additional exchange of S, and Bob's public key is unknown or Bob does not have a public key, there would be no way to communicate a random S to Bob. In this case there are two possibilities: either Alice and Bob can create their own independent S values, and not do full checking of each other's packets (they could do partial checking, but could not validate the encrypted KG values), or they could let S = K.

If S = K, the SKR blocks are defined the same as before, but the method has several disadvantages. First, since K must be known, the phase 1 SKR block cannot be precalculated. (Once a phase 1 SKR block is calculated from a K, it can still be cached.) Second, if a given S (set equal to some K) is cached, the separation of knowledge between the agents and law enforcement is violated, as a lawful recovery of the K would reveal S to law enforcement, so recovery of all keys recoverable under that S would also be recoverable by law enforcement. Third, semantic security of the key K is lost. That is, given a guess of K, anyone can test the guess by forward calculating the SKR blocks and comparing them for equality.

As setting S = K reduces security, and imposes some performance disadvantage, it should be done only if there is no alternative, which means only if: 1) the application cannot exchange S; 2) Bob's public key is not known or is not available; and 3) the communication is non-interactive. For file applications, where Bob = Alice, there should be no need to set S = K.

5. Detailed SKR Protocol

The following is a detailed nominal SKR protocol specification, which attempts to be as general as possible, allowing for cases involving interactive or non-interactive communications, one or more communicating parties, with one or more agents per party, and one or more communications per SKR phase 1 block. This protocol specification is only one example specification, as the SKR system may be tailored for a given application. For example, if an application deals only with domestic encrypted data archives, the application could use a substantially smaller and simpler protocol.

5.1 Phase 1

1. If possible, Alice and Bob exchange a random SKR seed S.
2. Alice sends to Bob the SKR phase 1 data block (B1), composed of:
   - M: an identifier to indicate an SKR block 1
   - T1: the SKR phase 1 public header
   - PRi: Party Record for communicating party i
   - RARj: Recovery Agent Record for agent j

where T1 is composed of:

- SKR version number
- SKR public key encryption method (nominally RSA)
- SKR secret key encryption method (nominally keyed shuffler)
- SKR hash method (nominally SHA-1)
- flag indicating if SKR is to transmit S to parties
- number of parties
- number of agents
- size in bits of supplied KG
- crypto period of KG information
- SKR block 1 timestamp
- SKR block 1 ID

Each PRi block is composed of:

- party id — namespace number and communicating user's name
- country id — namespace number and country name
- keyid — id of party's public key
- ePUi(S, Ri) — if SKR is doing the exchange of S

Each RARi block is composed of:

- set number — which set of agents this agent is in
- agent number — which set of agent this is agent is in
- agent ID — namespace, name
- agent's keyid — which of the agent's public keys was used
- agent's key size in bits
- ePUx(T1 || KGx, saltx), where
  KGx = Hash(key generating header, counter, T1, agent x's id, S)
  saltx = Hash(salt generating header, counter, T1, agent x's id, S)

5.2 Phase 2

1. If possible, Alice and Bob exchange a random session key K.
2. Alice sends to Bob the SKR phase 2 data block (B2), composed of:
   - M2 an identifier to indicate an SKR block2
   - T2
   - Hash(B1)
   - PRi optional Party Record for party i
   - RARj optional Recovery Authentication Record for agent j
   - KRRm Key Recovery Record for set of agents m

where T2 is composed of:

- T1
- flag to indicate if PR records are used to exchange K
- flag to indicate if optional RAR records are included
- session id
- session timestamp
- session key type (i.e., encryption algorithm -- DES, IDEA,...)
- session key size in bits
- session key crypto period

Optional PRi is composed of:

- party id namespace number and name of communicating user
- keyid id of party's public key
- ePUi(K, Ri) if SKR is doing the exchange of K

Optional RARi is composed of:

- set number s which set of agents
- agent number a which agent within the set
- Hash(authorization header, counter, AIsa, KAsa.i, T2 for i-th K)

KRRi is composed of:

- set number I which set of agents
- size of R in bits
- fKKi1.1(fKKi2.1(K)) - R
- Hi = Hash(validation header, counter, fKKi1.1(fKKi2.1(K)), T2, KKi1.1, KKi2.1)
- KGx = Hash(G generating header, counter, T1, agent x's ID, S)
- KHx = Hash(H generating header, counter, T1, agent x's ID, KGx)
- KIx = Hash(I generating header, counter, T1, agent x's ID, KGx)
- KKx = Hash(KK generating header, counter, T2, agent x's ID, KHx)
- KAx = Hash(KA generating header, counter, T2, agent x's ID, KIx)

Country	Encryption Algorithm	Maximum Allowed Key Length (bits)			Public Keys			
		Without SKR Intra-Country	With SKR Inter-Country		Key Recovery Agent 1	Key Recovery Agent 2	...	Key Recovery Agent N
			R	Entire Key				
X	DES	inf.	40	64	1FCD38...	74901A...	...	30FA67...
	RC5	inf.	40	64	1FCD38...	74901A...	...	30FA67...
Y	DES	128	40	128	E52AC3...	F32AB7...	...	5EF200...
	RC5	128	40	128	E52AC3...	F32AB7...	...	5EF200...
Z	DES	64	64	128	6494FF...	66673E...	...	342781...
	RC5	64	64	128	6494EF...	66673E...	...	342781...
W	DES	128.	40	128	E52AC3...	F32AB7...	...	5EF200...
	RC5	128.	40	128	E52AC3...	F32AB7...	...	5EF200...
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.

Table 1: Global jurisdiction/policy table

5.3 Global Jurisdiction/Policy Table

Referring to Table 1, the information required by the key recovery system of the SKR system is stored in a table called the global jurisdiction/policy table. Table 1 is for illustrative purposes only. In an actual implementation the data would be stored appropriately, perhaps in separate tables, one specifying the key recovery agents' public keys and one specifying the rules. The table contains information allowing the system to calculate the sizes of the keys and key recovery values for specific algorithms and users located in different countries. It may also contain the public keys of key recovery agents authorized for each country. The numbers in the table are examples only to demonstrate the kind of flexibility the SKR system permits. The variations are virtually unlimited. In particular, each country may have many key recovery agents.

For inter-country communications, the key recovery system can determine the receiver's country ID from his public key certificate or comparable system configuration information. Using the sender's origin country ID and the receiver's destination country ID, and the algorithm ID of the intended encryption algorithm to be used by the sender and the receiver, the key recovery system will determine the following: (1) the maximum key length that the sender and the receiver can use, (2) the allowed R values, which can be different for the sender and the receiver, and (3) the required key recovery agent IDs needed by the key inversion function. The key recovery system then passes this information to the key inversion function. The key length is the smaller of the two key length values. For example, for countries X and Y the key values for DES are 64 and 128 bits, in which case 64 is the value used.

5.4 Keyed Shuffler

The keyed shuffler function (referred to as shuffler function or just shuffler) is an invertible function that transforms an n-bit input X into a "shuffled" n-bit output Y. The shuffler function performs $i > 3$ iterations of shuffling (where i is a variable), which guarantees complete mixing. That is, each bit in the output depends on each bit in the input. In our example, we choose $i = 4$. The specification of the shuffler does not prescribe a particular value for i.

Figure 5 provides a high-level block diagram of the shuffler function. As a simplification, we assume that the input consists of two 160-bit parts (XL and XR). (The general specification of the shuffler function calls for the function to handle an input of arbitrary length.) At hash iteration 1, XR is hashed to produce a 160-bit hash value H(XR), which is Exclusive-ORed with XL to produce the masked output mXL. At

hash iteration 2, mXL is hashed to produce a 160-bit hash value H(mXL), which is Exclusive-ORed with XR to produce the masked output mXR. At hash iteration 3, mXR is hashed to produce a 160-bit hash value H(mXR), which is Exclusive-ORed with mXL to produce the masked output mmXL. At hash iteration 4, mmXL is hashed to produce a 160-bit hash value H(mmXL), which is Exclusive-ORed with mXR to produce the masked output mmXR.

The shuffler can be run in reverse order to "unschuffle" mmXL||mmXR to recover the original input XL||XR. The hash iterations are performed in the reverse order (4, 3, 2, and 1). At hash iteration 4, mmXL is hashed to produce a 160-bit hash value H(mmXL), which is Exclusive-ORed with mmXR to recover mXR. At hash iteration 3, mXR is hashed to produce a 160-bit hash value H(mXR), which is Exclusive-ORed with mmXL to recover mXL. At hash iteration 2, mXL is hashed to produce a 160-bit hash value H(mXL), which is Exclusive-ORed with mXR to recover the original input XR. At hash iteration 1, XR is hashed to produce a 160-bit hash value H(XR), which is Exclusive-ORed with mXL to recover the original input XL.

5.4.1 General Definition of the Shuffler Function

The n-bit input X is processed by the shuffler function as follows:

1. X is divided into a left part (XL) and right part (XR), where the length of XL and XR are defined as follows: a) if n is even, then XL and XR each contain n/2 bits, whereas b) if n is odd, then XL contains (n - 1)/2 bits and XR contains (n + 1)/2 bits.
2. XR is used to mask XL. The masked XL is then used to mask XR. The masked XR is used to mask the masked XL. And finally, the masked XL is used to mask the masked XR. The masking

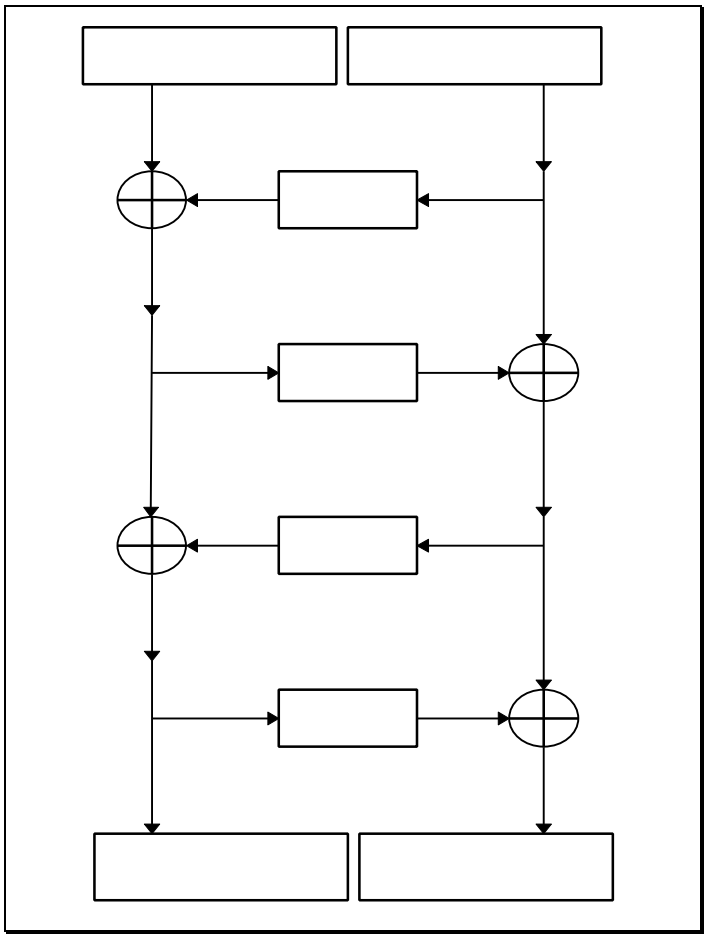


Figure 5: Keyed shuffler process

operations are defined separately below.

5.4.1.1 Masking Operation

XL is first divided into blocks of k bits. If the length of XL is a multiple of k, then each block contains k bits. If the length of XL is not a multiple of k, then XL will consist of one short block (< k bits) and optionally 1 or more blocks of k bits. If a short block exists, it will be constructed from the most significant bits of XL.

k is defined as the block size of the hash algorithm. For example, if SHA-1 is the hash algorithm, then k = 160. If MD5 is the hash algorithm, then k = 128. Unless otherwise stated, it's assumed that SHA-1 is the hash algorithm employed by the shuffler function.

The blocks in XL are divided as follows:

≤ k bits	. . .	k bits	k bits
Block i	. . .	Block 2	Block 1

The input to be hashed is defined as:
Input = Public Header || Key || XR
where the Public Header is a 10-byte encoded structure as follows:

- 1 byte Identifier: "shuffler" id
- 4 bits Iteration: 1 = 1st iteration
- 4 bits Algorithm: ID of hashing algorithm
- 4 bytes Counter: 1, 2, etc. the counter matches the block number in XL to be masked.
- 4 bytes Length: length of data to be shuffled in bits.

The masking operation consists of the following steps:

1. Set Counter = 1. Hash the Input (Public Header || Key || XR) with the hash algorithm to produce a k-bit hash H1, where "Key" is the secret key specified to the keyed-shuffler function. Exclusive OR H1 with block 1 from XL to produce the masked block 1 (denoted mblock 1).
2. Set Counter = 2. Hash the Input (Public Header||Key||XR) with the hash algorithm to produce a k-bit hash H2, where "Key" is the secret key specified to the keyed-shuffler function. Exclusive OR H2 with block 2 from XL to produce the masked block 2 (denoted mblock 2).
3. The operation continues until the last block (block i) has been masked. If the last block is a short block containing j bits (j < k), then block i is masked by Exclusive ORing the least significant j bits of Hi with block i to produce the masked block i (denoted mblock i).

The masked XL is denoted mXL.

XR is masked with mXL in the same way that XL was masked with XR, except that the Iteration Number in the Public Header is set to 2 (denoting "2nd iteration"). The input to be hashed is defined as:
Input = Public Header || Key || mXL The masked XR is denoted mXR.

mXL is masked with mXR in the same way that XL was masked with XR, except that the Iteration Number in the Public Header is set to 3 (denoting "3rd iteration"). The input to be hashed is defined as:
Input = Public Header || Key || mXR The masked mXL is denoted mmXL.

mXR is masked with mmXL in the same way that XL was masked with XR, except that the Iteration Number in the Public Header is set to 4 (denoting "4th iteration"). The input to be hashed is defined as:
Input = Public Header || Key || mmXL The masked mXL is denoted mmXR.

5.4.2 Key Specified to the Shuffler

The SKR system contemplates that there may be a variety of ways to "key" the shuffler. One method would be to combine the key and the remainder of the input to be hashed, i.e., input = (public header || key || data) where data may be XR, mXL, mXR, mmXL, etc. For example, the key could be prepended (front of input) or appended (back of input). However, just prepending or appending the key is not necessarily enough as it is possible to construct distinguishers (i.e., a statistical test that detects that the function is not behaving as a random one). However, appending and prepending parts of the key to the data will avoid this problem [13].

A paper by Bellare et. al. [14] proves that a structure like the shuffler is a pseudo-random permutation if the function used inside is a pseudo-random function. This paper proves that appending and prepending parts of the key to the data are pseudo-random functions if the collision function of the hash algorithm H is pseudo-random.

Therefore, the SKR system contemplates that it is preferable to divide the key and prepend one part and append the other part. However, the SKR system is not limited to this method of combining the key with the data to be hashed within the shuffler. One may also divide the key into smaller parts and interleave these parts, or combine the key with the data using some other combining function, which may be a simple function or a complex function, before the hashing operation is performed.

6. Improved Parameter Validation

In the SKR system as described above, Bob cannot validate the recovery information of block B1 without repeating the computationally expensive public-key encryption steps performed by Alice. Also, in order for Bob to validate the recovery information KGa1-KGb2, he must be able to recreate it; this in turn requires either (1) that Alice and Bob agree to the random seed S in advance or (2) that Alice transmit the seed S to Bob in an additional encrypted field in B1, increasing the size of the required communications. The enhanced SKR system described below, denoted SKR/DH (for its use of Diffie-Hellman shared secret techniques) eliminates the need to send additional secret information (S) in encrypted form, since it enables Bob to recover the KG values directly by decryption rather than having to reconstruct them from such secret information.

In general, SKR/DH changes phase 1 in two ways. First, the secret value S is not exchanged; Alice simply picks random KG values for each key recovery agent. Second, these values of KG are encrypted using a shared DH key as described below. Given these changes, Bob learns the values of KG not by deriving them from S, but by decrypting them from the data block B1. This decryption can also serve to verify for Bob that the encrypted KG values can correctly be decrypted by the respective agents.

In more detail, the modified SKR phase 1 is (assuming two agents for Alice and two agents for Bob):

Let Alice have the DH keypair $\{x, g^x \bmod p\}$

Bob have the DH keypair $\{y, g^y \bmod p\}$

Alice's Agent1 have the DH keypair $\{a, g^a \bmod p\}$

Alice's Agent2 have the DH keypair $\{b, g^b \bmod p\}$

Bob's Agent1 have the DH keypair $\{c, g^c \bmod p\}$

Bob's Agent2 have the DH keypair $\{d, g^d \bmod p\}$

where the components x, y, a-d are secret values and the components (g^x mod p, etc.) are public values known to all parties.

Figure 6 shows the initial key generation procedure performed by Alice and Bob, either as a preliminary to the phase 1 procedure or as a step of the phase 1 procedure if it has not been performed previously. Both Alice and Bob calculate the following first pair of shared secret and public values shared by Alice and Bob:

- u = g^{xy} mod p (their common key)
- {u, g^u mod p} (a new DH keypair shared between Alice and Bob)

as well as the following additional shared secret values shared by Alice, Bob and a particular agent:

- e = g^{au} mod p (a new common key between Alice-Bob and her Agent1)
- f = g^{bu} mod p (a new common key between Alice-Bob and her Agent2)
- G = g^{cu} mod p (a new common key between Alice-Bob and his Agent1)
- h = g^{du} mod p (a new common key between Alice-Bob and his Agent2)

(Note the use of the uppercase G to avoid conflict with the symbol g for the generator.)

Figure 7 shows the phase 1 procedure followed by Alice for a given phase 1 block B1. Alice calculates a public header:

- T1 (a public header including IDs of Alice, Bob, all key recovery agents, and a timestamp)

as well as the following keys for encrypting the key-generating keys KGa1-KGb2:

- I = Hash(T1 || e)
- J = Hash(T1 || f)
- k = Hash(T1 || G)
- L = Hash(T1 || h)

where || indicates concatenation. (Note the use of lowercase k to avoid conflict with the symbol K for the session key.) This additional hashing operation enhances the security of the system, since even if an attacker ascertains, say, I for a particular T1, he will be unable to reverse the hash transformation so as to obtain either e or I for another T1. Any suitable hash function may be used for these and other hashing operations described herein, so long as it is infeasible either to invert (i.e., find an input producing a given output) or to find another input producing the same output as a given input. One such suitable hash function is the Secure Hash Algorithm (SHA-1), described, for example, in [15].

Alice then picks random KGa1, KGa2, KGb1, KGb2, encrypts them with keys I, J, k, L, using any suitable symmetric encryption procedure such as the DES. If desired, Alice may concatenate each of the KG values with a random salt before encrypting it, for example:

I(KGa1 || Salt)

This enhances the resistance of the system to certain types of attacks. Unlike the SKR system described above (where Bob must be able to replicate the encryption procedure, which uses the public key of an agent), Alice may use truly random salts for this purpose without having to communicate any secret values to Bob.

After encrypting the values KGa1-KGb2, Alice sends to Bob a phase 1 block B1 containing:

- T1 (the phase 1 public header)
- g^u (the shared DH public key)

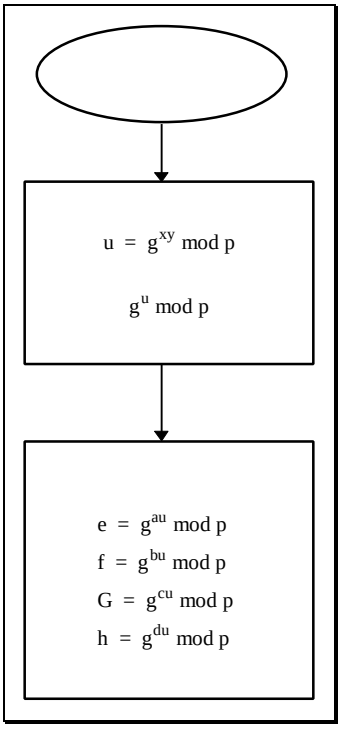


Figure 6: DH key generation

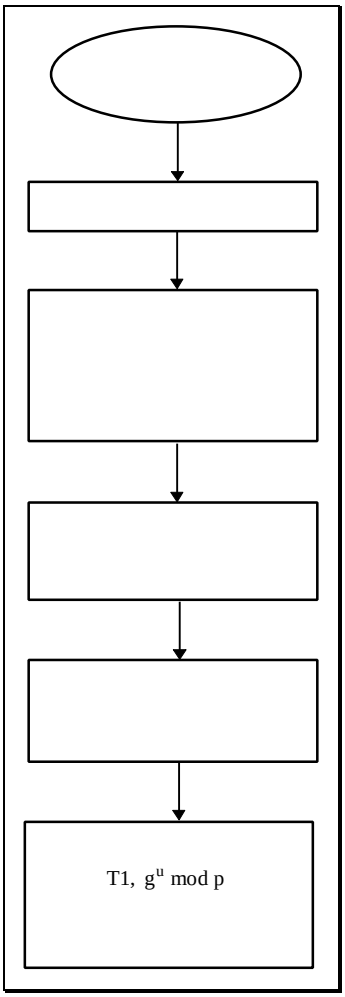


Figure 7: Phase 1 procedure

- I(KGa1) (symmetric encryption of KGa1 with key I)
- J(KGa2) (symmetric encryption of KGa2 with key J)
- k(KGb1) (symmetric encryption of KGb1 with key k)
- L(KGb2) (symmetric encryption of KGb2 with key L)

Upon receiving the phase 1 block B1, Bob calculates the I, J, k, L values from e, f, G, h and B1.

Thereafter, Bob decrypts the values KGa1, KGa2, KGb1, KGb2 using the values I, J, k, L, checking that the KG values decrypt properly. For this purpose it is assumed that the symmetric encryption includes (in addition to any random salt) a message authentication code (MAC), since the values KGa1-KGb2 are arbitrary and would provide no indication of whether they had been recovered correctly. Any suitable means may be used to generate such a MAC, such as concatenating the KG value being encrypted with a hash of itself and encrypting the concatenation result, as for example:

I(KGa1 || Hash(KGa1))

The hash value provides a check on the decryption. If Bob uses the wrong value of I, or if the encrypted value is corrupted in transmission, it is exceedingly unlikely that the hash of the decrypted KG would match the MAC.

Bob then caches the decrypted values KGa1, KGa2, Kgb1, KGb2, indexed by the Hash(B1) of the full block B1.

The Phase 2 procedure remains unchanged.

In summary, upon receipt, Bob performs a validation step, the nature of which depends on the number of copies of the session key K involved. If there is only a single set of key recovery agents and Bob has no independent copy of the session key K other than the one obtained by decrypting a recovery field, then no validation is necessary, since Bob is necessarily using the same values as the law enforcement agent and key recovery agents would use to recover the key. On the other hand, if Bob has an independent copy of the session key K (i.e., obtained outside the described procedure), or if there are multiple sets of key recovery agents as in the present example, then Bob must ensure that all of the various versions of K agree if he is to fully validate the recovery information; if they do not agree, then at least one set of key recovery agents has invalid recovery information.

Validation of a particular recovery field containing a fKK1(fKK2(K)) (where KKi signifies KKai or KKbi) may be done in any one of several ways. Most straightforwardly, one may doubly decrypt the recovery field using KK1 and KK2 and see if the resulting K agrees with the one independently obtained. Alternatively, one may doubly encrypt an independently obtained K using KK1 and KK2 and see if the result agrees with the recovery field. As a further alternative, one may singly decrypt the recovery field using KK1 and singly encrypt the independent K using K2 and see if the two operations produce the same value fKK2(K).

The preliminary procedure is performed once for each set of DH secret values x, y, a-d and corresponding public values; the phase 1 procedures are performed once for each set of values KGa1-KGb1, the phase 2 procedures are performed once for each session key K; and the encryption and decryption procedures are performed once for each encrypted message K(message) sent from Alice to Bob. Data blocks B1 and B2 are sent at least often enough to ensure their interception by law enforcement agents, since these blocks are essential for key recovery. They may be sent more often, if desired, and could even accompany each encrypted message.

6.1 Recovery Procedure

A law enforcement agent first obtains the data blocks B1-B2 and the encrypted message K(message) by any suitable means, such as by monitoring Alice's transmissions over the communications channel. The law enforcement agent then presents the data block B1 and public header T2, along with appropriate evidence of authority such as a judicial search warrant, to the key recovery agents. Preferably, the agents do not get the doubly encrypted recovery field KKa1(KKa2(K)), as that would allow them to reconstruct the session key K, which is generally not desired.

If they are satisfied with the authority of the law enforcement agent, the key recovery agents regenerate respective key-encrypting keys KKa1-KKa2 and return these values to the law enforcement agent. Upon receiving these values from the key recovery agents, the law enforcement agent decrypts the recovery field fKKa1(fKKa2(K)) using KKa1 and KKa2 to recover the session key, and decrypts the encrypted message K(message) using the recovered session key K.

Upon receiving B1, T2 and a warrant from law enforcement agent, the key recovery agent inspects the warrant to determine whether the requested recovery is authorized. If it is, then the key recovery agent generates the shared secret value

$$e = g^{au} \bmod p$$

using the public value g^u from B1 and its own secret value a, and hashes this value to generate the value

$$I = \text{Hash}(T1 \,||\, e)$$

where T1 is obtained from B1. The key recovery agent then decrypts I(KGa1) using I, and generates the key-encrypting key KKa1 from KGa1 using the key derivation procedure used by Alice and by Bob. Finally, the key recovery agent hands over the key-encrypting key KKa1 to the law enforcement agent, who uses it as described above to recover the session key K.

7. Conclusion

In this paper, we presented a new cryptographic key recovery system that enables authorized agencies to recover keys from both interactive and non-interactive (i.e., store-and-forward) communications. The presented system provides interoperability between users in different countries with different regulations without compromising the need of users for maximum possible security.

The key recovery system that we described allows the costly public key encryption operations (i.e., encrypting the key-generating keys) to be performed one time and then used multiple times to generate the needed key-encrypting keys which will enable the recovery of multiple keys over a potentially long period of time. Thus, the secret values provided to the key recovery agents are potentially long-lived key-generating keys and the secret values used to cryptographically seal the keys to be protected by the key recovery mechanism are key-encrypting keys. The key-encrypting keys are derived from these key-generating keys using a key derivation procedure that tightly couples the derived key-encrypting keys to the unique public key recovery information associated with the key being protected by the key recovery mechanism. Moreover, the key recovery method presented in this paper uses only published public keys -- no additional keys need be exchanged between the communication parties or the key recovery agents.

The enhanced parameter validation of the presented system, denoted SKR/DH (for its use of Diffie-Hellman shared secret techniques), eliminates the need to send additional secret information (S) in

encrypted form, since it enables the receiver to recover the key-generating key values directly by decryption rather than having to reconstruct them from such secret information. Since SKR/DH uses Diffie-Hellman for the public-key calculations, elliptic curve public-key techniques can be substituted, further improving performance of SKR/DH by reducing the public-key key lengths needed for a given security level.

8. References

[1] D. E. Denning and D. K. Branstad, "*A Taxonomy for Key Escrow Encryption Systems*," Communications of the ACM, vol.39, no. 3, pp. 34-40, March 1996.

[2] D. E. Denning and W. E. Baugh, "*Key Escrow Encryption Policies and Technologies*," Information System Security, vol. 5, no. 2, pp. 44-51, Summer 1996.

[3] J. Kilian and T. Leighton, "*Fair Cryptosystems, Revisited*," Proceedings of CRYPTO '95, pp.208-221, 1995.

[4] E. H. Freeman, "*When Technology and Privacy Collide. Encoded Encryption and the Clipper Chip*," Information System Management, vol. 12, no. 2, pp. 43-46, Spring 1995.

[5] S. Landau, S. Kent, C. Brooks, S. Charney, D. Denning, W. Diffie, A. Lauck, D. Miller, P. Neumann, D. Sobel, "*Crypto Policy Perspectives*," Communications of the ACM, vol. 37, no. 8, pp. 115-121, August 1994.

[6] J. Nechvatal, "*A Public Key Based Key Escrow System*," Journal of System Software, vol. 35, no. 1, pp. 73-83, October 1996.

[7] J. He and E. Dawson, "*A New Key Escrow Cryptosystem*," Proceedings of Cryptography: Policy and Algorithms, pp. 105-114, 1995.

[8] S. T. Walker, S. B. Lipner, C. M. Ellison, D. M. Balenson, "*Commercial Key Recovery*," Communications of the ACM, vol. 39, no. 3, pp. 41-47, March 1996.

[9] S. Micali, "*Fair Public Key Cryptosystems*," Proceedings of CRYPTO '92, pp.113-138, 1992.

[10] Y. Frankel and M. Yung, "*Escrow Encryption Systems Visited: Attacks, Analysis and Designs*," Proceedings of CRYPTO '95, pp.222-235, 1995.

[11] W. J. Caelli, "*Commercial Key Escrow: An Australian Perspective*," Proceedings of Cryptography: Policy and Algorithms, pp.40-64, 1995.

[12] D. B. Johnson and S. M. Matyas, "*Enhanced Optimal Asymmetric Encryption: Reverse Signatures and ANSI X9.44*," Proceedings of RSA Data Security, 1996.

[13] M. Luby, C. Rackoff, "*How to construct pseudo-random permutations from pseudo-random functions*," SIAM Journal of Computing, vol. 17, no. 2, pp. 373-386, April 1988.

[14] M. Bellare, R. Canetti, and H. Krawczyk, "*Pseudo-random functions revisited, the cascade construction and its concrete security*," Proceedings of the 37th Conference on Foundations of Computer Science, pp. 514-523, 1996.

[15] B. Schneier, "*Applied Cryptography*," 2nd edition, John Wiley & Sons Inc., 1996.