Key recovery in a business environment

Konstantinos Rantos

Technical Report
RHUL–MA–2001–4
1 November 2001

Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX, England
http://www.rhul.ac.uk/mathematics/techreports

Key Recovery in a Business Environment

by

Konstantinos Rantos

Thesis submitted to the University of London
for the degree of Doctor of Philosophy

Department of Mathematics
Royal Holloway, University of London

2001

Declaration

These doctoral studies were conducted under the supervision of Chris Mitchell and Peter Wild.

The work presented in this thesis is the result of original research carried out by myself, in collaboration with others, whilst enrolled in the Department of Mathematics as a candidate for the degree of Doctor of Philosophy. This work has not been submitted for any other degree or award in any other university or educational establishment.

Konstantinos Rantos
March 8, 2001

List of Publications

A number of papers resulting from this work have been presented in or submitted to refereed conferences and journals.

- G. Horn, P. Howard, K.M. Martin, C.J. Mitchell, B. Preneel, and K. Rantos. Trialling secure billing with trusted third party support for UMTS applications. In *Proceedings of 3rd ACTS Mobile Communications Summit*, pages 574–579, Rhodes, Greece, 1998.
- K. Rantos and C.J. Mitchell. Remarks on KRA's key recovery block format. *Electronic Letters*, **35**:632–634, 1999.
- C. Markantonakis and K. Rantos. On the life cycle of the certification authority key pair in EMV'96. In *Proceedings of Euromedia '99*, pages 125–130, Munich, Germany, 1999.
- K. Rantos and C.J. Mitchell. Key recovery in ASPeCT authentication and initialisation of payment protocol. In *Proceedings of 4th ACTS Mobile Communications Summit*, pages 629–634, Sorento, Italy, 1999.
- C.J. Mitchell and K. Rantos. A fair certification protocol. *ACM Computer Communication Review*, **29(3)**:47–49, July 1999.
- K. Rantos and C.J. Mitchell. Key recovery for archived data using smart cards. In *Proceedings of the 5th Nordic Workshop on Secure IT Systems*, Reykjavik, Iceland, 2000.
- K. Rantos and C.J. Mitchell. Matching Key Recovery Mechanisms to Business Requirements. Submitted.
- K. Rantos and C.J. Mitchell. Key recovery scheme interoperability – a protocol for mechanism negotiation. Submitted.

Acknowledgements

This thesis would not have been possible without the guidance, technical and personal support of a number of people. My initial and foremost thanks to my supervisor Chris Mitchell for his continued support, interest, patience and guidance throughout my Doctoral Studies at Royal Holloway. His style of supervision along with his important comments and suggestions for improvements have encouraged me to pursue my ideas. Special thanks also to my advisor Peter Wild. I benefited from the added dimension he brought to this thesis.

Particular thanks to my colleague Keith Howker who made my integration into the ASPeCT project less difficult than it might otherwise have been. My thanks to my friends and colleagues Kostas Markantonakis, Keith Martin, Peter Burge, Barry Rising, Zbigniew Ciechanowicz, Ioannis Michalopoulos, Kostas Zafeiris and all of those who helped to make my stay in Royal Holloway so enjoyable.

I cannot thank Aphrodite enough for her love, kindness, tolerance and support over the last few years. She has been a source of constant inspiration to me.

I would also like to thank my sister Anna and Makis for the good time that I have when I go back home, and to congratulate them on their marriage as it made me feel very happy.

My parents have set the cornerstone for my studies. They have guided, motivated and supported me through my academic career and my life. Mum and dad thanks.

Finally, I wish to dedicate this thesis to my grandmother Athina whose funeral I was unable to attend due to my studies far away from home.

Abstract

This thesis looks at the use of key recovery primarily from the perspective of business needs, as opposed to the needs of governments or regulatory bodies.

The threats that necessitate the use of key recovery as a countermeasure are identified together with the requirements for a key recovery mechanism deployed in a business environment. The applicability of mechanisms (mainly designed for law enforcement access purposes) is also examined. What follows from this analysis is that whether the target data is being communicated or archived can influence the criticality of some of the identified requirements.

As a result, key recovery mechanisms used for archived data need to be distinguished from those used for communicated data, and the different issues surrounding those two categories are further investigated. Two mechanisms specifically designed for use on archived data are proposed.

An investigation is also carried out regarding the interoperability of dissimilar key recovery mechanisms, when these are used for encrypted communicated data. We study a scheme proposed by the Key Recovery Alliance to promote interoperability between dissimilar mechanisms and we show that it fails to achieve one of its objectives. Instead, a negotiation protocol is proposed where the communicating parties can agree on a mutually acceptable or different, yet interoperable, key recovery mechanism(s).

The issue of preventing unfair key recovery by either of two communicating parties, where one of the parties activates a covert channel for key recovery by a third party, is also investigated. A protocol is proposed that can prevent this. This protocol can also be used as a certification protocol for Diffie-Hellman keys in cases where neither the user nor the certification authority are trusted to generate the user's key on their own.

Finally, we study the use of key recovery in one of the authentication protocols proposed in the context of third generation mobile communications. We propose certain modifications that give it a key recovery capability in an attempt to assist its international deployment given potential government demands for access to encrypted communications.

Contents

List of Figures

List of Tables

Abbreviations

AE	Authorised entity
AIP	Authentication and Initialisation of Payment
APDU	Application Protocol Data Unit
API	Application Programming Interface
AR	Access Rule
ARI	Access Rule Index
ASPeCT	Advanced Security for Personal Communications Technology
BRV	Backup Recovery Vector
CAD	Card Acceptance Device
CKR	Commercial Key Recovery
CKRB	Common Key Recovery Block
DES	Data Encryption Standard
DRC	Data Recovery Centre
DRF	Data Recovery Field
EES	Escrowed Encryption Standard
ICC	Integrated Circuit Card
IFD	Interface Device
IP	Internet Protocol
JCRE	Java Card Runtime Environment
JCVM	Java Card Virtual Machine
KDC	Key Distribution Centre
KR	Key Recovery

KRA	Key Recovery Agent
KRC	Key Recovery Centre
KRF	Key Recovery Field
KRM	Key Recovery Mechanism
KTC	Key Translation Centre
LEA	Law Enforcement Agency
LEAF	Law Enforcement Access Field
MAC	Message Authentication Code
OCF	OpenCard Framework
PKI	Public Key Infrastructure
SKR	Secure Key Recovery
TLS	Transport Layer Security
TTP	Trusted Third Party
UMTS	Universal Mobile Telecommunications System
VASP	Value Added Service Provider
VFV	Validation Field Value

Chapter 1

Introduction

Contents

The aim of this chapter is to describe the context of the research and present the structure of the thesis.

1.1 Motivation and challenges

In recent years, key recovery has captured the attention of information security specialists as well as government officials and the public. It has been one of the most controversial issues in the information security area.

With increasing demands for the use of encryption on communications, governments have sought routes to lawful access to encrypted data while allowing the use of strong encryption. This lawful access was intended to serve the purpose of dealing with serious crime and for national security reasons. From the government perspective, key escrow gives a means of access to encrypted data. This solution, however, has caused widespread opposition as it has been seen as a potential infringement of the rights of individuals.

Key recovery, and more specifically key escrow, first gained publicity from the United States government's proposal for compulsory escrow of keys [70]. It was part of this government's plan to relax export restrictions on cryptographic products that employ strong encryption. The result was the proposal of a key escrow scheme, namely the *Escrowed Encryption Standard* (EES) or Clipper, which provides users with strong encryption for their communications, and *Law Enforcement Agencies* (LEAs) with access to users' encrypted communications when authorised. The latter, however, reinforced opposition arising from possible threats to privacy.

Key recovery apart from fulfilling LEAs demands for access to encrypted communications, is likely to be an essential tool for businesses and organisations that want to ensure continuous access to their encrypted data. Businesses cannot tolerate denial of access to their data due to loss of access to decryption keys as the lost data could be of considerable value. Key recovery in a business environment should therefore be considered as part of routine disaster recovery planning.

Although much work has been done regarding key recovery mechanisms, the majority of it has been in the context of providing sound key recovery mechanisms that meet government demands for lawful access. There is little published material on key recovery mechanisms and their benefits when they are applied in a business environment. However, through the process of seeking sound mechanisms for law enforcement demands, schemes that serve business requirements have also started to emerge. As in any immature market, the existence of proprietary mechanisms has caused problems, such as interoperability issues among dissimilar schemes, that have not been thoroughly investigated. Moreover, the design of most of these mechanisms has focused on encrypted communications without taking into account their use for encrypted archived data or, at least, the different requirements that surround key recovery mechanisms when deployed for communicated and for archived data. Therefore, interoperability of dissimilar mechanisms and key recovery schemes for archived data are among the issues that urgently need further investigation.

The Key Recovery Alliance is an international industry group that was founded by eleven major information technology vendors in 1996 (by the end of 1997 more than 60 companies had joined the group). It was set up to develop high-level cryptographic "key-recovery" solutions that meet the requirements of business for strong encryption and could allow easing of restrictions of cryptographic import/export around the world. Among the aims of the alliance was to provide some kind of standardisation for key recovery mechanisms, with a particular focus on interoperability issues. However, public opposition to the general notion of key recovery restricted the Key Recovery Alliance's activities and, in a particular case, even forced a company to withdraw from the alliance. Key recovery was something that nobody wanted to talk about, regardless of the environment in which it was to be used, and this seemed to have an effect on the work carried out by the Key Recovery Alliance. It is worth mentioning, however, that almost all the companies that provide cryptographic solutions incorporate some-kind of proprietary key recovery within their products, not necessarily for LEA demands but for the end user. This is an indication that businesses, which must safeguard their

encrypted assets from loss of privately held keys, are starting to see key recovery as an essential tool, which must exist in their cryptographic products.

1.2 Contribution of this thesis

This thesis investigates certain issues that surround key recovery mechanisms when deployed specifically in a business environment. It looks at this topic entirely from the business perspective without considering LEA requirements.

A thorough analysis of the requirements that a key recovery mechanism should fulfil when deployed in a business environment is given, and an investigation of the applicability of existing mechanisms follows. What follows from this investigation is that the nature of the target data, i.e. whether the mechanism is used for communicated or archived data, can influence the criticality of the requirements. Such an investigation was considered necessary as businesses wishing to use key recovery have to be aware of the advantages and disadvantages of deploying a mechanism that was designed to serve law enforcement access.

The identification of different requirements for communicated and archived data has motivated research for a mechanism that can be deployed specifically for archived data. Two mechanisms of this type are proposed, one of which is an adaptation of a mechanism proposed by Maher in [53]. The mechanisms meet different requirements, with the main distinction being that one of them requires the existence of an on-line key recovery agent during key generation, which helps counter rogue user attacks.

The work within this thesis was carried out in parallel with the work of the Key Recovery Alliance, and the results of the latter's work have been closely monitored. In particular, consideration was given to the solution that the Key Recovery Alliance has proposed to solve the interoperability problems between dissimilar key recovery mechanisms. It is shown that the proposed model has some flaws and does not achieve

what the alliance promises. Therefore, further research on this important topic is necessary. A solution is proposed for the interoperability problem that arises from the use of dissimilar mechanisms incorporated within various cryptographic solutions. The proposed scheme gives the communicating parties the ability to negotiate the key recovery mechanism(s) to be used, thus avoiding the situation where the two entities are unable to establish a secure association because of interoperability issues between the corresponding key recovery mechanisms.

This thesis is not primarily concerned with potential government requirements for law enforcement access. However, in Chapter 9 a solution is proposed that allows law enforcement access to encrypted telecommunications; this was done to help the international deployment of the ASPeCT (Advanced Security for Personal Communications Technology [2]) protocol bearing in mind potential government demands for compulsory deployment of key recovery mechanisms. The objective was to put a mechanism in place that will guarantee time-bounded interception and protect users against unauthorised access to their communications.

1.3 Structure of the thesis

The thesis is divided into five parts. Part I introduces the concepts and principles of key recovery mechanisms and investigates the requirements of a key recovery mechanism deployed in a business environment. More specifically, Chapter 2 gives an introduction to key recovery mechanisms. It introduces the concept of key recovery and explains how key recovery mechanisms work using selected examples of mechanisms proposed by both the commercial sector and academia. Chapter 3 outlines the requirements for a key recovery mechanism when deployed specifically in a business environment, as opposed to the use of a key recovery mechanism for law enforcement access. Within this chapter the need for distinction between key recovery mechanisms for archived and for communicated data is outlined, while a thorough investigation of these two classes

follows in Parts II and III respectively.

Part II covers key recovery mechanisms specifically deployed for archived data. In Chapter 4 the requirements of such a key recovery mechanism are described and an analysis of the applicability of existing mechanisms follows. Moreover, two new mechanisms are proposed which fulfil the identified requirements. The first one is a variant of Maher's mechanism described in [53], which avoids certain problems. Chapter 5 gives the details of a prototype implementation of this scheme.

Part III covers issues surrounding key recovery mechanisms for communicated data and more specifically investigates interoperability of key recovery mechanisms. Chapter 6 gives an introduction to the interoperability problem that dissimilar key recovery mechanisms might face in encrypted communications and identifies those factors that can cause interoperability problems using a detailed description of a key recovery model. Within this chapter, a scheme proposed by the Key Recovery Alliance as a solution to the interoperability problem is also investigated and the reasons why this scheme fails to achieve its objectives are explained. In Chapter 7, a protocol for negotiating the key recovery mechanism(s) to be used by two communicating parties is described. This protocol aims to help overcome the interoperability problems that might arise from the intention of two communicating parties to use dissimilar key recovery mechanisms.

Part IV of this thesis looks at key recovery from a different perspective, namely the prevention of unfair key recovery by either of two communicating parties. More specifically, it looks at one of the standardised key agreement mechanisms of ISO/IEC 11770-3 [38] and considers the problem that either party might influence the generated key, while activating a covert channel for key recovery by a third party. The protocol proposed as a countermeasure can also be used as a certification protocol for Diffie-Hellman keys.

Finally, Part V gives an introduction to the ASPeCT *authentication and initialisation of payment* (AIP) protocols and describes certain modifications that give them a key re-

covery capability. The modified protocols enable time-bounded law enforcement access to encrypted communications. Two different solutions are proposed for this purpose.

Part I

KEY RECOVERY IN BUSINESS ENVIRONMENTS

Chapter 2

Information Recovery Using Key Recovery

Contents

The aim of this chapter is to give an overview of key recovery mechanisms. Some of the various types are illustrated with selected examples that have been proposed by both the commercial sector and academia. Attacks that seek to defeat the objective of key recovery, and weaknesses introduced into the security infrastructure by the use of key recovery, are also described, together with some techniques that can be used as countermeasures.

2.1 Data protection and key recovery

The dependence of organisations and enterprises on the global information infrastructure for conducting business with their partners or consumers is increasing very rapidly. Information has become one of the most valuable assets of a corporation, and hence the need for protection of this information has emerged. This underlines the importance of the security mechanisms used for the protection of sensitive information to prevent loss of proprietary data and to safeguard its integrity.

Among the protection mechanisms used are cryptographic techniques, which can help guarantee the confidentiality, integrity and authenticity of information. Key management techniques exist for the cryptographic keys used, covering the generation, distribution, validation, update, storage, usage, and destruction of keys. Key management techniques should, however, also deal with situations where keys become forgotten, damaged, rendered unavailable, or authorised third party access is required. Key recovery, backup and archival are techniques that can be used for this purpose.

Key backup refers to the backup of key material in independent, secure storage media, during operational use. *Key archival* is a form of backup where the target keys are no longer in normal use. It refers to long-term storage of post-operational keys and is used for key retrieval under special conditions, such as settling disputes involving repudiation. Both key archival and backup can be considered as key recovery techniques.

Although the term key recovery lacks a precise definition, and its interpretation may vary, within the context of this thesis *key recovery* (KR) is defined as the process that enables authorised entities to recover decryption keys, when they are not otherwise available, with the ultimate goal of recovering the plaintext from encrypted data. This process is typically an integral part of the key management infrastructure and is achieved through reconstruction and/or retrieval of the target key. Related terms that have been used in the literature include *key escrow*, *encryption control*, *commercial key*

recovery, *cryptographic backup and recovery*, *virtual addressing*, and *key encapsulation*, which are defined differently by various communities of interest and have poorly defined differences. We use the terms key escrow and key encapsulation to describe two fundamentally different approaches to key recovery.

One of the most critical components of a *key recovery mechanism* (KRM) is a *key recovery agent* (KRA), which is a *trusted third party* (TTP) that assists in the recovery of keys by preserving the appropriate key material. The KRA can be external or internal to an organisation, with associated advantages and disadvantages (these are described in detail in Chapter 3). Trust in the KRA is obviously a crucial requirement, as the KRA will potentially hold the key to all the users' encrypted information. If the KRA becomes corrupted or changes its policy, the users' privacy is endangered.

Alternative names for the KRA that have been used in the literature include *key recovery centre* (KRC), *data recovery centre*, *key escrow agent* or even just *agent*. Throughout this thesis these terms are considered synonymous, unless specifically distinguished, although there is a trend to use the term *key escrow agent* only for key escrow mechanisms.

2.2 An abstract model of a key recovery mechanism

An abstract model of a key recovery mechanism is depicted in Figure 2.1. This is a functional model, depicting the processes of a key recovery mechanism while emphasis is given to the entities that are involved. A similar model has been described by Denning in [17] and an entirely functional model can be found in [42].

The entities that are typically involved are the KRA and the two communicating parties, namely A and B, unless the mechanism is used for archived data in which case only A will be present (the ciphertext together with the potential key recovery field are simply sent to a storage device). The *Authorised entity* within whose domain the data

recovery takes place can be any entity authorised to recover encrypted data, such as the corporation management, or the user that encrypted the data.

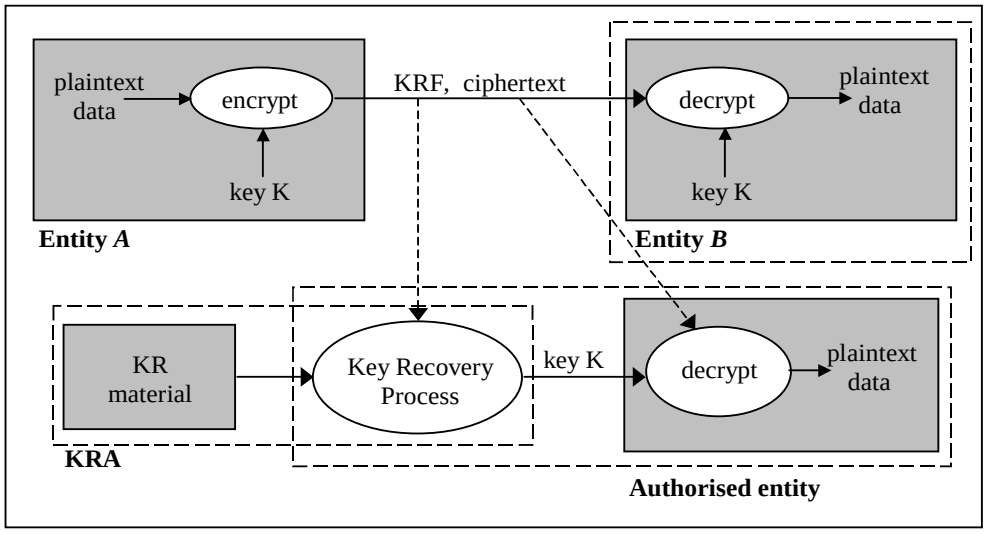


Figure 2.1: A typical KRM

In the model, A will encrypt the *plaintext data* with the session key K producing the *ciphertext*. A will also generate the *key recovery field* (KRF), which is information that can be interpreted by the `Key Recovery Process`.

The recovery of the target keys is performed by the `Key Recovery Process`. This might be part of the KRA's or the *Authorised entity*'s functionality (the latter possibility applies when the key recovery material is dispersed among multiple agents; e.g. the Escrowed Encryption Standard [70]). The `Key Recovery Process` takes as input information necessary for this process, namely 'KR information', which has the form of a *key recovery field* (KRF) or *KR material*. The *KRF* represents KR information generated by either or both communicating parties while the *KR material* can include cryptographic values that A escrows with *KRA*, or *KRA*'s own key material essential for the recovery of the requested keys.

For simplicity reasons, only one KRA is shown, which is assumed to be the one with which entity A is associated. Although interaction between A's and B's KRAs can take place during any of the communication phases, it is not shown in the figure as inter-

TTP communication is beyond the scope of the model given here. No communication, however, takes place between an entity and the peer's KRA unless the two entities share the same KRA. Finally, note that A might be associated with multiple KRAs, in which case the KR material is distributed among them.

2.3 Recoverable keys

For the purposes of this thesis we assume that the target keys that can be recovered using a KRM are keys used by a cryptographic mechanism deployed for data confidentiality (i.e. a cipher mechanism). Note that, in any event, it is usually considered bad practice to give third parties access to signature keys. If a user's private signing key is lost then a new key pair can be generated, and use of cryptographic mechanisms can continue uninterrupted without loss of service. Of course, if a user's private signature key is compromised, then different problems arise, but this is outside the scope of the thesis.

The situation, however, is rather different for private decryption keys for a public key cryptosystem. Third party access to private decryption keys is often necessary, a property that strongly suggests the maintenance of separate key pairs for signatures and for encryption.

Regarding the nature of the target keys, there are typically the following possibilities:

- *Data decryption keys.* These keys are used directly for the decryption of the encrypted data, and include session keys for encrypted communications and file keys.
- *Master keys or key decryption keys.* These keys typically enable access to multiple data encryption keys, e.g. master keys used for encrypting multiple session keys.

Given that the current trend is the deployment of symmetric ciphers for data encryption, while public-key cryptosystems are used for key management purposes, throughout this thesis it is assumed that the data decryption keys are those used by symmetric ciphers unless otherwise stated. In this context, the term "encryption key" will also denote the decryption key of the target data.

2.4 Classification of key recovery mechanisms

Although many categories have been introduced for the classification of KRMs (see [83]) key recovery mechanisms can typically be divided into *key escrow* or *key encapsulation* mechanisms.

In a typical *key escrow mechanism*, an escrow agent holds a copy of all or part of the user's keys (either data or key encryption keys). In [66] a key escrow mechanism is described as a method of KR in which "the secret or private keys, key parts, or key-related information to be recovered are stored by one or more key escrow agents". As a result, each user has to escrow directly with his agent his private keys, or each session key that he uses. A typical key escrow agent could be an on-line TTP acting as a *key distribution centre* (KDC) or a *key translation centre* (KTC), which keeps a copy of all keys that the user establishes. Variants of key escrow mechanisms exist and some of them are described in [83].

In another scenario, the user escrows an initial value, namely a Master Key, with his agent, which is subsequently used for the generation of all session keys (for example using a hash function and a time-stamp). A further alternative is when the user escrows with his agent the private key of an asymmetric key pair that can be used to compute the secret session keys [28]. An example of the latter is the JMW scheme [39, 40, 41] which is described in detail in Section 2.5.1.2. As can been seen from the above, a wide range of KR solutions can be classified as key escrow mechanisms. All of them

are characterised by the storage of key-related information with a trusted agent that gives the latter the ability to recover all the user's decryption keys.

In a typical *key encapsulation mechanism*, the user encloses the KR information (e.g. session keys or key parts) in an encrypted KR block that is made available to the agent(s) with which the user is associated, and which can be decrypted only by this agent(s). The KR information is typically encrypted using the agent's public encryption key, and attached to the encrypted data as a *key recovery field* (KRF) [91]. In a more general definition, [66], a key encapsulation mechanism is described as "a method of key recovery in which keys, key parts, or key related information are encrypted specifically for the KRA function and associated with the encrypted data", where the KRA function is "a key recovery system function that performs a recovery service in response to an authorised request".

There are also KR mechanisms that are difficult to categorise into one of the above two classes. There are, for instance, key escrow schemes that also require the transmission of a KRF for the KRA to be able to recover the keys. An example is a KRM where the user escrows his private decryption key with his agent, and uses the public key for the transmission of any key related material. Such a scheme can be considered as a *hybrid* mechanism, but it can also be classified as a key escrow scheme since it involves escrowing key-related material, regardless of the transmission of a KRF. There are also key encapsulation schemes for which the KRF is not restricted to the transmission of session keys encrypted under the KRA's public encryption key. As an example, consider the KRM proposed by Maher in [53]. Although the author claims that the proposed scheme is a key escrow mechanism, using the above definition the scheme should be classified as a key encapsulation mechanism, since the user does not escrow any key material with his KRA but instead makes this information available to the KRA function.

A KRM that cannot be classified under these two categories is *weak encryption*, which

is identified in [45] as an *encryption control* system. For the purposes of this thesis, however, we do not consider this as a type of KRM as it contradicts one of the basic requirements for a KRM deployed in a business environment (see Chapter 3), which is that the deployed KRM should not weaken security and, more specifically, the cryptographic mechanisms used. Besides that, and as mentioned in [45], this type of key recovery is unlikely to be attractive to any of the interested entities, i.e. users and/or corporations.

2.5 Examples

In this section we consider some representative examples of KRMs proposed either by industry or academia.

2.5.1 Key escrow

2.5.1.1 The US Escrowed Encryption Standard

The US *Escrowed Encryption Standard* (EES) [70] was developed and standardised by the US National Institute of Standards and Technology (NIST), a division of the US Department of Commerce. EES is probably the most well-known and controversial key escrow mechanism. It was designed to provide decryption of encrypted sensitive but unclassified communicated data, when their interception is lawfully authorised. For the purposes of this standard, data includes digitised voice, fascimile or computer information communicated in a telephone system.

The EES is implemented on a tamper-resistant hardware chip. Two devices have been produced for this purpose: Clipper and Capstone. The EES standard is based on a NSA-designed algorithm called SKIPJACK [82], and a method that allows for government access through a chip unique key and a *Law Enforcement Access Field* (LEAF)

transmitted with the encrypted communications. For this purpose, the following elements are included in the chip.

1. The SKIPJACK encryption algorithm.
2. An 80-bit *family key KF* that is common to all chips.
3. A *chip unique identifier* (UID).
4. An 80-bit *chip unique key KU*, which is the exclusive-or (XOR) of two 80-bit *chip unique key components* ($KU1$ and $KU2$).
5. Specialized control software.

Prior to the use of EES, an initialisation phase takes place, where two government-approved escrow agents generate key components $KU1$ and $KU2$ respectively, which are bitwise XORed to form the chip unique key KU, i.e. $KU = KU1 \oplus KU2$. The key components are encrypted with the escrow agents' secret key encrypting keys and escrowed along with the chip identifier *UID*. In particular, $KU1$ is encrypted with the key encrypting key $K1$, assigned to escrow agent 1, to produce $E_{K1}(KU1)$, where $E_K(X)$ denotes encryption of X with key K. Similarly, $KU2$ is encrypted with $K2$, which is assigned to escrow agent 2, to produce $E_{K2}(KU2)$.

The mechanism. Two entities that wish to communicate using a security device that contains an escrowed encryption chip should first establish an 80-bit session key *KS*, which will be used to encrypt the communications. Once *KS* is established, it is passed to the chip and the LEAF generation method is invoked. The 128-bit LEAF consists of the 32-bit *UID*, the session key *KS* encrypted with *KU*, and a 16-bit checksum, which is computed using the session key and an *initialisation vector* (IV) (which may be generated by the chip). The LEAF, encrypted with the unique family key *KF*, and the IV are then transmitted, together with the encrypted data, to the receiving entity as shown in Figure 2.2.

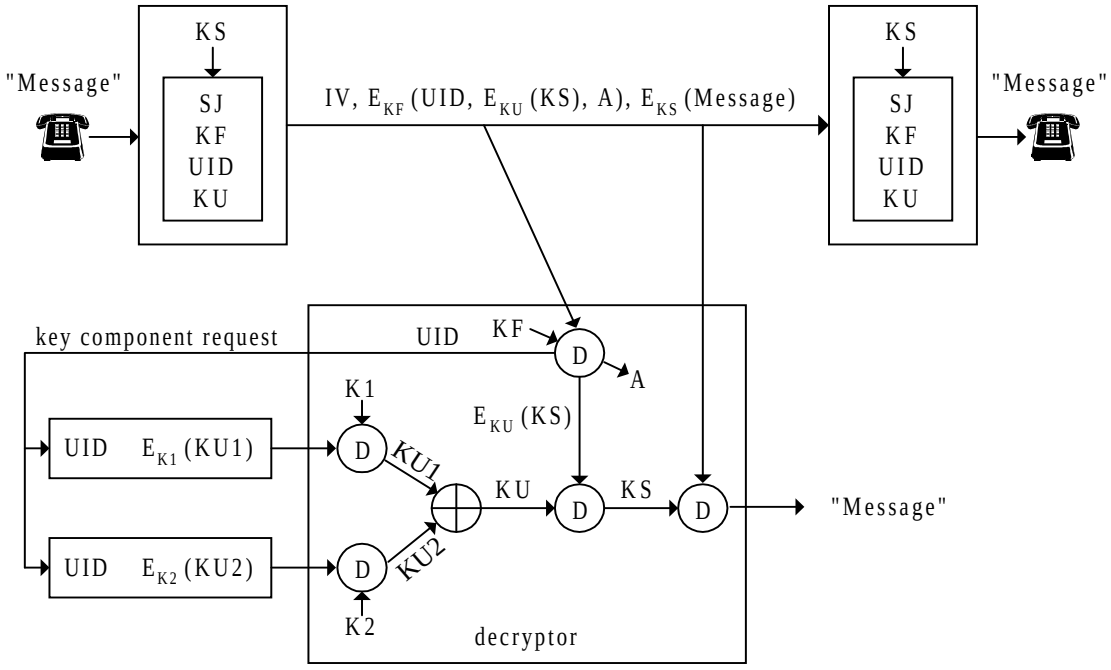


Figure 2.2: Escrowed Encryption Standard

Warranted interception. Lawful interception of communications that take place on a particular line is feasible after the authorised entity has obtained a court order to intercept it. The court order is passed to the telecommunications service provider in order to get access to the communications associated with that line. As described in [13], if encrypted communication is detected "the incoming line will be set up to pass through a special government-controlled decrypt device" as shown in Figure 2.2. "The decrypt device will recognize communications encrypted with a key escrow chip, extract the LEAF and IV, and decrypt the LEAF using the family key KF in order to pull out the chip identifier UID and the encrypted session key $E_{KU}(KS)$."

The chip identifier will be given to the escrow agents along with the request for the corresponding key components, documentation that certifies the authorisation of the surveillance, and the serial number of the decrypt device. In response to the request, the escrow agents will release the corresponding key components to the authorised entity. These components will then be used to form the key KU, as required to decrypt KS.

2.5.1.2 The JMW scheme

Another representative example of key escrow is the JMW scheme [39, 40, 41], named after its three inventors: Jefferies, Mitchell, and Walker. It is based on the Diffie-Hellman key exchange algorithm, [20], and is "intended to provide warranted access to user communications" via the TTPs with which the two communicating parties A and B are associated (TA and TB correspondingly).

Prior to use of the mechanism, an initialisation has to take place. More specifically, every pair of TTPs, whose users wish to communicate securely, have to do the following.

- Agree on public values g and p, where g is a primitive element modulo p and p is a large integer.
- Agree on the use of a digital signature algorithm, choose their own signature key/verification key pair, and exchange verification keys in a reliable way. Any user B wishing to receive a message from a user A, with associated TTP TA, must be equipped with a trusted copy of TA's verification key.
- Agree on a secret key $K(TA, TB)$ and a Diffie-Hellman key generation function f.

If A wishes to send a secure message to B, A and B have to be provided with certain parameters by their respective TTPs.

- Using the function f, and the secret key $K(TA, TB)$ and the name of B as input to f, both TA and TB generate the private integer b satisfying $1 < b < p-1$. This key is known as B's *private receive key* and it needs to be securely transferred from TB to B. The corresponding *public receive key* for B is set equal to $g^b \bmod p$. The key b will be used to protect all traffic from clients of TA to B.

- *TA* randomly generates a *private send key* for A, denoted a, and signs a copy of A's *public send key* (equal to $g^a \bmod p$). A's *private send key* and *public send key* will constitute A's *send key pair*, which A will use when sending confidential messages to any users, not just those associated with *TB*.
- A must also be equipped with a copy of B's public receive key. This can be computed by *TA* and transferred in a reliable way to A.

The mechanism. The information the two entities possess prior to the use of the mechanism, can be used to generate a shared key $g^{ab} \bmod p$ (where a is A's private send key and g^b is B's public receive key) for protecting the confidentiality of a message sent from A to B. This key can be used as the session key or, as the authors suggest, it can be better used as a key encryption key, which in turn can be used for protecting a suitable session key.

User A then sends the following information to user B.

- The message encrypted using the session key (either $g^{ab} \bmod p$ or a key encrypted using $g^{ab} \bmod p$),
- A's public send key $g^a \bmod p$ signed by *TA*, and
- the public receive key $g^b \bmod p$ for user B.

Upon receipt of the message B, using A's public send key $g^a \bmod p$ and B's own private receive key b, can generate the session key $g^{ab} \bmod p$ and decrypt the received encrypted message.

A representation of the scheme is given in Figure 2.3 (note that inter-TTP communication is not shown; $Cert_{TA}(g^a)$ denotes a copy of A's public send key signed by *TA*).

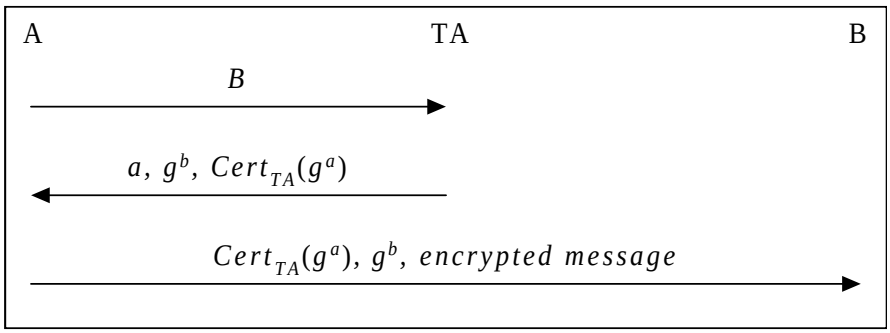


Figure 2.3: JMW scheme

Warranted interception. Warranted interception of encrypted communications is feasible as long as one of *TA* and *TB* operates within the intercepting authority's jurisdiction. The intercepting authority has the following options (assuming that *A*, served by *TA*, sends encrypted data to *B*, served by *TB*).

- Request *TA* to recover the shared key $g^{ab} \bmod p$. In that case, *TA* has the following options:
  - Combine *B*'s private receive key *b* (generated from $K(TA, TB)$ and the name of *B* using the key generating function *f*) with *A*'s public send key, sent with the message.
  - Combine *A*'s private send key *a* with *B*'s public receive key, sent with the message.
- Request *TB* to recover the key. Because *TB* does not have access to *A*'s private send key there is only one way that *TB* can recover the key, namely by combining *B*'s private receive key *b* and *A*'s public send key, sent with the message.

Properties of the mechanism. The JMW scheme has the following properties.

- A user can arrange for his/her *send key pair* to be changed at any time by simply requesting his/her TTP to generate a new key pair, which is then passed by the TTP to the user.

- No directories are required to make the system work. An entity wishing to send a message only needs to obtain the public receive key for the intended recipient from his/her own TTP, who can generate this information merely from the name of the recipient and the identity of the recipient's TTP. A recipient of an enciphered message will, given the information contained in the message, possess all the data necessary to obtain the session key, without further reference to any third parties.
- All receive key pairs can automatically be updated at regular intervals, by including date information (e.g. month and year) within the scope of the key generating function f.

2.5.2 Key encapsulation

We next describe three examples of key encapsulation schemes.

2.5.2.1 Commercial Key Recovery

Probably the most representative example of a key encapsulation mechanism is the one proposed by Walker, Lipner, Ellison, and Balenson in [91], which the authors refer as a "Commercial Key Recovery (CKR) System". The main objective for the design of this mechanism was to "respond to the objections to Clipper without sacrificing the government's law enforcement interests".

The proposed scheme employs a *data recovery centre* (DRC), which is a KRA either established by a corporation, for its own use, or by an external organisation which offers this service. Every time a key recovery enabled program encrypts a file or message using a session key, the corresponding key recovery information is generated. This is the *data recovery field* (DRF), containing the session key and the user's identity encrypted using the DRC's public key and the DRC's identifier. "This information is stored with the file or sent with the message, and these are the only places the session keys reside.

There is no database of escrowed keys, at the DRC or elsewhere".

If an authorised entity, e.g. a user or corporate management, is unable to decrypt a message, it needs to send the DRF together with appropriate authentication information to the corresponding DRC. The DRC, using its private key, will decrypt the DRF and return the session key to the requester via a secure channel.

2.5.2.2 TIS Commercial Key Recovery

The TIS CKR scheme, also described in [91], is a key encapsulation mechanism with enhanced features that allow authentication and authorisation of the entity requesting recovery. In particular, "the DRF holds more than merely the encrypted file key". It includes the following unencrypted fields:

- a version number (for the DRF itself),
- a name for the DRC, and
- a number for the DRC's public key (since each DRC can have multiple key pairs).

It also contains a field encrypted with the indicated DRC's public key, containing:

- an *access rule index* (ARI), which tells the DRC how to challenge any requester to make sure that he/she has permission to get access, and
- the file key (an individual encryption key for this encrypted data only).

The ARI is a number, chosen by the DRC, indicating an *access rule* (AR), which "defines the process for verifying permission to gain emergency access". ARs are defined by the user during the initial registration phase, and each user can have any number

of ARs, "to represent different sets of people or conditions for emergency access to different data".

The TIS CKR scheme uses four types of ARs.

1. A direct personal identification (such as a password, one-time password, or public key signature verification), which can be checked directly between the person and the DRC over a secure channel.
2. An indirect personal identification, in which a third party vouches for the identity of the individual.
3. Testimony about a condition, e.g. testimony by a trusted individual or organisation about an event, such as a death or transfer of funds that has cleared the bank.
4. A group of ARIs. This is a threshold group of n previously defined ARIs, such that k out of the n ARs in the group need to be satisfied in order to satisfy this AR. The ARIs involved can be of any of the four types. The threshold group allows specification of logical `AND`, `OR` and `majority` – as special cases.

The scheme has also an override capability that permits emergency access to data owned by the corporation.

2.5.2.3 IBM Secure Key Recovery

Another example of a key encapsulation mechanism is IBM's *Secure Key Recovery* (SKR) [27]. SKR differs from other key encapsulation mechanisms in that the data encryption key is not encrypted under the KRA's public key but under a symmetric key established between the user and the KRA(s). The proposed mechanism can be used for encrypted communications as well as for encrypted archived data.

For this scheme each user initially registers with two key recovery agents in his domain, while each KRA has at least one private and public key pair.

The mechanism. The SKR mechanism consists of a two-phase process as shown in Figure 2.4. The division of the mechanism into two phases allows the expensive public key computations to be avoided in all but the first communication between the communicating parties. More specifically, Phase 1, which must occur at least once before one or more communication sessions take place, consists of the following steps.

1. The communicating parties A and B establish a common random seed S. This is typically done using a key agreement protocol, e.g. Diffie-Hellman key exchange, or an RSA key transport protocol.
2. A uses S to derive random *key-generating keys* (KG), specific to each recovery agent, by hashing S and the respective agent's ID.
3. A encrypts KG using the respective agents' public keys.
4. A sends to B the SKR Block 1 (B1) which consists of: $T1$, $e_{PUa1}(KGa1)$, $e_{PUa2}(KGa2)$, $e_{PUb1}(KGb1)$, and $e_{PUb2}(KGb2)$, where
   - $T1$ is a public header containing IDs for A, B, and all key recovery agents,
   - e_P denotes public key encryption with key P,
   - $PUai$ is the public key for A's ith agent,
   - $PUbi$ is the public key for B's ith agent,
   - $KGai$ is the key generating key for A's ith agent, and
   - $KGbi$ is the key generating key for B's ith agent

In Phase 2, the two parties establish the session key K, which will be used for encrypting any subsequent communications, using any key transport or key agreement protocol.

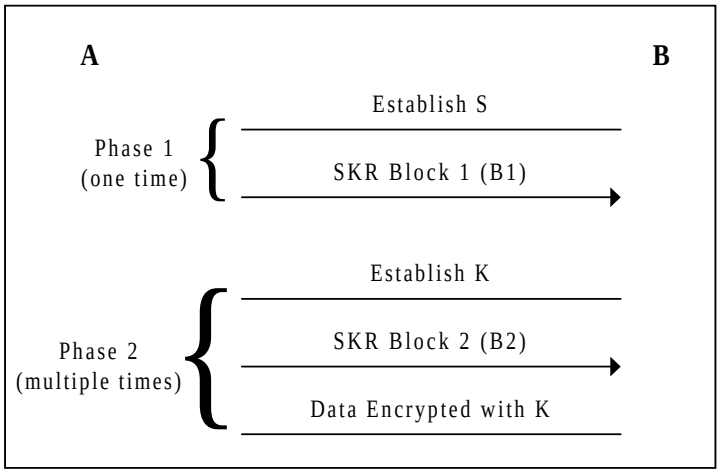


Figure 2.4: SKR communication process

Phase 2 occurs prior to the start of encryption for each encrypted session and consists of the following steps.

1. A and B establish session key K.
2. A derives for each agent a key generating value KH as the hash of the respective KG value, and a key-generating header. The KH key-generating keys are used to derive subordinate sets of pseudo-random key-encrypting keys KK (see [27, 35] for a detailed description of key generation in SKR).
3. A derives for each agent a session-specific key KK as the hash of the respective KH value, and the session's public header $T2$.
4. A nested encrypts K under the sets of session-specific KK key encrypting keys.
5. A sends to B the SKR Block 2 (B2) which consists of the values: $T2$, $h(B1)$, $E_{KKa1.1}(E_{KKa2.1}(K))$, and $E_{KKb1.1}(E_{KKb2.1}(K))$, where
   - $T2$ is a public header consisting of T1, the session ID, a timestamp e.t.c.,
   - E_K denotes symmetric key encryption using key K,
   - h is a hash function,
   - $KKai.1$ is the first key encrypting key for A's ith agent, and
   - $KKbi.1$ is the first key encrypting key for B's ith agent.

Key recovery phase. In the recovery phase, the authorised entity that requests the recovery of a key has to pass to a set of agents the block B1 and the public header T2 from B2. The agents are not given the rest of T2, and in particular the doubly encrypted key K, so that they will not be able to recover the key individually. The agents decrypt their respective blocks and recover the KG values, which they use together with the headers T1 and T2 to generate the values KK and KA for the session covered by the T2 header. Each agent returns to the Recovery Service the calculated KK, as shown in Figure 2.5, and the latter recovers the requested key by decrypting the encrypted recovery field $E_{KKa1.1}(E_{KKa2.1}(K))$ or $E_{KKb1.1}(E_{KKb2.1}(K))$.

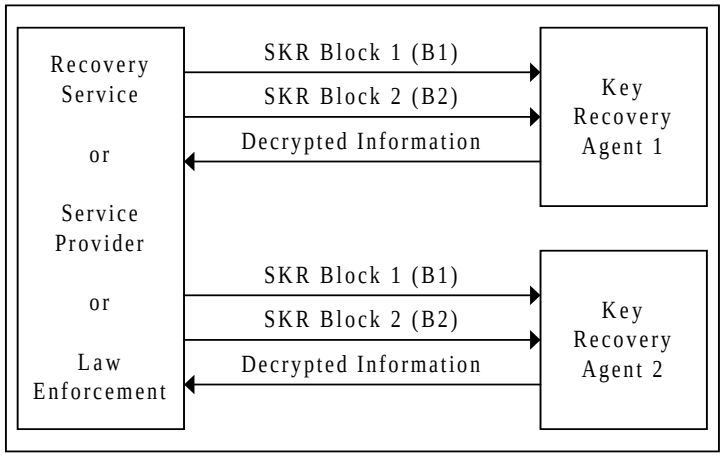


Figure 2.5: SKR key recovery process

As the agent does not return the long term KG value but only key-encrypting keys KK, the secrecy of the KG value is not compromised, and hence the requesting authority cannot recover data from other sessions. However, if the user and agent agree, it is possible to release the value KH to the requesting authority, which will give access to a set of sessions instead.

2.6 Attacks on KRM functionality

In this section we describe two methods of attack on KRM functionality. The term 'attack' is used here to describe means of defeating the objective of key recovery, i.e.

preventing the authorised recovery of keys to decrypt encrypted data.

2.6.1 Rogue user attacks

Rogue user attacks are amongst the most effective attacks against a KRM, and are probably the most difficult to prevent. Such an attack typically involves a *rogue user*, who wants to circumvent the KR system so that his encrypted data cannot be accessed by an authorised third party even when the KR system is used [67]. The rogue user bypasses or disables the key recovery functionality, either by passing the wrong key material to the agent or by altering or deleting the key recovery information generated by the mechanism. The way these attacks are mounted depends on the mechanism and in some cases they can be successfully prevented. However, no KRM can prevent dual rogue user attacks, where there is collusion by two or more dishonest users. In the case of archived data, it is impossible to prevent a rogue user from deploying his own cryptographic infrastructure [91], a situation that constitutes another form of rogue user attack.

2.6.2 Superencryption

Superencryption, i.e. a technique where data is encrypted more than once, can be considered as a special type of rogue user attack. In this case, the rogue user does not attempt to bypass the key recovery functionality using any of the methods previously mentioned, but he rather uses a non-escrow cryptosystem to pre-encrypt his data [4, 71]. Thus, the user makes proper use of the KRM but the agent recovers encrypted data instead of plaintext. The user, however, who appropriately manages the keys of the non-escrow cryptosystem, will have full access to the plaintext.

2.7 Making KRMs robust

In this section, we describe some techniques that can be used as countermeasures to the attacks on the KRM functionality identified in the previous section.

2.7.1 Integrity checks

An integrity check is a means whereby interested parties can ascertain that the key recovery functionality is properly invoked and has not been maliciously modified by a rogue user, i.e. the key recovery mechanism is not being circumvented.

In [4], a set of measures for achieving the integrity of software products are described, including the following.

- Build integrity checks at several points into the product. These typically involve the use of digest functions embedded in the product for verifying the integrity of the product functionality by comparing the computed digest with the one given by the software's vendor.
- Have different integrity checks verify different areas of the product.
- Use several different instruction sequences to perform the integrity checking function. This will prevent an adversary from performing an automated scan to identify and remove all integrity checks at once.

2.7.2 Implementation on trusted hardware

In order to achieve a higher degree of security, parts of the security functionality on the user's terminal could be implemented on a separate, trusted, tamperproof security module or token, such as a smart card. Such an approach can help to prevent rogue

user attacks by keeping the security functionality intact.

2.8 Weaknesses introduced by deployment of KRMs

Key recovery mechanisms, if they are not properly deployed, can introduce weaknesses into the security infrastructure, which an adversary might exploit.

2.8.1 Attacks on the agent

Key recovery systems provide an alternative means of access to decryption keys, and hence to the data encrypted with these keys. The concentration by some KRMs of key material at a single location, especially in a key escrow system, make this location a point of attack that an attacker might want to exploit [1]. Unless these keys are appropriately protected and access control mechanisms are in place, the deployed KRM will introduce a major vulnerability point in the security chain. Attacking single keys that give the attacker access to a large number of the users' keys also constitutes a major threat. This is the case with key encapsulation mechanisms where a vast number of users' keys are encrypted under a single agent's private key. If this key is compromised, all the data encryption keys encrypted with it will be revealed to the attacker.

2.8.2 Untrusted agents

Although one of the crucial requirements for an entity that acts as a key recovery agent is to be trusted, there is no guarantee that this will always be the case. There might be situations where the agent might relax its security policies, go bankrupt, or get corrupted. In situations like these, the secrecy of the users' keys, and hence their data, is critically endangered.

2.9 Strengthening the security infrastructure

In this section, several countermeasures that can be used to enhance the security infrastructure are described.

2.9.1 Dispersion

Dispersion is the property where "joint (but not necessarily unanimous) participation by multiple, designated entities is an essential step in key recovery" [83]. Such an approach typically requires that the recovery of the target keys is done by an independent authorised entity, such as a law enforcement agency or the management of the corporation. Each agent administers part of the required key recovery information. Participation of all, or k out of n agents in the case that a (k, n) threshold scheme is used [59], is essential for the recovery of the keys. An example is provided by the Escrowed Encryption Standard, where recovery of the requested keys is done by the authorised entity and not by the agents, while participation of 2 out of 2 agents is required for the recovery, i.e. EES is a $(2, 2)$ threshold scheme.

2.9.2 Collusion-resistance

In some environments, an additional goal to dispersion might be to ensure that even if the agents collude they will not be able to recover the required key material. This feature allows the user or an authorised entity to control the recovery of keys. Collusion resistance typically requires an authorised entity to retain critical information for the recovery of keys [83].

2.9.3 Residual work factor

A residual work factor requires the agent to possess only a portion of the information required for the recovery of keys [80]. Keys can then be recovered through trial-and-error techniques or cooperation with the entity that retains the rest of the required information, including the user or the entity that is authorised to recover the keys. As described in [83], "a residual work factor can be used to increase the overall work effort involved in key recovery, to discourage 'casual' recovery requests and to keep part of the overall security of key recovery in the hands of the user".

2.9.4 Making the trusted parties oblivious

Micali in [60] describes a method of avoiding unauthorised actions by trusted parties especially in the case of law enforcement access. In particular, Micali considers the situation where "a trustee requested by a court order to surrender his share of a given secret key may alert the owner of that key that his communications are going to be monitored". While, however, making trustees oblivious avoids this undesirable situation, several problems are introduced, the most important being the weakness of the system to unauthorised key recovery requests. That is, if the trustees do not know which user's key material they are revealing, they cannot easily verify the legitimacy of the request.

2.9.5 Using procedural means to protect the key recovery material

Almost any deployed cryptographic infrastructure (e.g. those containing Certification Authorities or Key Distribution Centres) concentrates trust in a small number of security-critical network entities. Compromise of such an entity will almost certainly severely damage security throughout the network that this trusted entity serves. In this respect, key escrow centres do not significantly worsen risks that are commonly

dealt with in existing security infrastructures which do not possess a KR capability.

The threat posed by such concentrations of trust is typically dealt with by well-established techniques, including physical and procedural security measures (combined with rigorous auditing of the use of security facilities). In particular, such measures can be used to enforce the principle of 'dual control' to critical security functions, whereby no single individual is ever granted access to such functions or data.

2.10 Summary

In this chapter we introduced the concept of key recovery and we gave a general model of a key recovery mechanism. We described several existing KRMs, covering both key escrow and key encapsulation, the two main categories of key recovery mechanisms.

Attacks both on KRM functionality and on weaknesses introduced by their deployment have been discussed as well as techniques that can be used as countermeasures.

Chapter 3

Matching key recovery mechanisms to business requirements

Contents

This chapter addresses the business needs for key recovery as a countermeasure to the threat of losing potentially valuable information. Requirements essential for a sound key recovery mechanism are described, and the applicability of the two main classes of key recovery schemes to a corporate environment is examined.

3.1 Protecting business information

Protection of information through the use of security mechanisms has become vital for business. Cryptographic keys, including key agreement keys, session keys used for encrypting communication sessions or stored data, and signature keys, are a crucial part of the security infrastructure protecting corporate data. Loss or unavailability of encryption keys will lead to an inability to access the encrypted information, a situation the corporation will typically not wish to tolerate. Within a business environment there are many cases where access to keys might be lost, arising from both deliberate actions and accidents. The former might originate from both outsiders and insiders, while accidents can be due to a failure of mechanisms.

As far as deliberate actions are concerned, it has been reported that more attacks to corporations' systems are likely to come from insiders rather than outsiders [15, 84]. This needs to be taken into account when establishing a cryptographic infrastructure offering services such as data confidentiality. Employees acting as the only holders of encryption keys might pose a threat to the corporation. Suppose a user's employment is terminated, and that the user is the only holder of keys used to encrypt business information. On leaving the company, the employee might withhold these keys, either deliberately or through simply forgetting to hand them to their legitimate owner. If there is no backup of these keys, and there is no way to recover or recompute them, then access to the information encrypted under them is infeasible (assuming that the cryptographic mechanisms used are strong enough to prevent a cryptanalytic means of decryption). Similar problems arise when an employee cannot be contacted because he is absent, e.g. on vacation. It is easier and safer to be able to recover the encryption key within the company's protected environment rather than having to contact the user, in some cases in insecure environments, bearing the risk of accidentally revealing the keys to untrusted third parties.

As far as failure of, or damage to, devices is concerned, this is always a threat in the

business environment. More specifically, if encryption keys are stored in a damaged device, and there is no backup or other means to recover the keys, then data encrypted under the inaccessible keys will be lost. This device could be a hard disk storing a file of passwords used to derive keys, or a smart card containing a key or key component. In the latter case, losing the token itself is also not unlikely.

In [15] it is mentioned that AccessData of Orem, Utah, a company that provides software and services to companies that have lost access to encrypted data, "reported in 1995 that they received about a dozen and a half calls a day from companies with inaccessible computer data. About a third of these calls resulted from disgruntled employees who left under extreme conditions and refused to cooperate in any transitional stage by leaving necessary keys. Another half-dozen resulted from employees who died or left on good terms but simply forgot to leave their keys. The remaining third resulted from loss of keys from current employees."

In addition to the possibility of key loss, companies may wish to monitor employees' communications, either external or internal, e.g. to track leaks of information. This is especially necessary in hierarchical environments where exchange of proprietary information, even within the company's domain, needs to be monitored. Corporate monitoring of communications can also deter employees wishing to break security policies governing the flow of classified information [35]. Corporations may further wish to have access to encrypted communications for non-repudiation purposes in the event of a dispute, or even for running checks on incoming traffic, e.g. for viruses or for intrusion detection.

Although key recovery can be used to deter employees, it can also be used to promote the use of cryptography. Unless they are sure that the data they encrypt can be recovered even if they lose the decryption keys, employees may be reluctant to use encryption, hence leaving their data unencrypted even though that information needs to be protected.

3.2 Distinguishing between a business environment and law enforcement access

The term key recovery, or more specifically key escrow, has attracted much unfavourable publicity mainly because of a number of government proposals for compulsory escrow of all private communications keys, see e.g. [92]. The intention of these proposals was to give governments the ability to decrypt intercepted communications to deal with criminal activities. However, this has been seen by a number of parties as a potential infringement of the rights of individuals and corporations to provide privacy for data stored and communicated electronically.

In a business environment, however, the situation is rather different [1]. Corporations cannot tolerate loss of potentially important information through the unavailability of encryption keys. Further, and most importantly, a company normally owns its information, and therefore the issues surrounding access to private communications through compulsory key escrow do not arise. KRMs deployed in a corporate environment can be thought of as part of routine disaster recovery planning.

Moreover, the requirements for KRMs used for law enforcement access and those deployed in a business environment are slightly different, making this distinction important. As described in [1], the requirements for a KRM designed for law enforcement access include the following.

1. **Access without end-user knowledge or consent**. While law enforcement typically requires the monitoring of users' communications without the latter being aware, recovery of keys in a business environment does not necessarily have this requirement. The company, as the owner of the encrypted data, has the right to recover keys should an emergency situation arise, with or without the user's knowledge or consent.
2. **Ubiquitous adoption**. Governments have been seeking the ubiquitous adoption

of "key recovery for all encryption, regardless of whether there is benefit to the end-user" [1]. In a business environment, however, deployment of a KRM will be in a restricted controlled environment where users can benefit from the existence of KRMs, as these can help prevent loss of access to their data.

3. **Fast access to plaintext**. Law enforcement access to encrypted communications demands service availability round-the-clock, and seeks the ability to obtain decryption keys quickly (in some cases within one or two hours). This is to help monitor fast-moving criminal or terrorist activities. Within a corporation, however, time restrictions are not expected to be such an important requirement when the KRM is used for archived data and the corporation can typically tolerate longer response times.

Granularity of keys is another strong requirement for KRMs deployed for law enforcement needs as it limits LEA access to those communications authorised by a valid warrant [45]. In a business environment, however, granularity is of minor importance as the corporation will typically have the right to access any corporate data.

A discussion of the need to distinguish between business and law enforcement requirements can also be found in [65]. Note, however, that in [65] the authors claim that *enforceability*, i.e. a requirement that the users cannot circumvent the KRM, is not strongly required in a business environment. We believe that this is not always true as enforceability (or *non-circumventability*, as it is identified in this thesis) is necessary for the prevention of rogue user attacks. The existence of a controlled environment, i.e. an organisation's infrastructure, does not necessary mean that the users cannot manipulate the generated key recovery information (in some KRMs), thus circumventing the key recovery functionality.

In the following section the requirements on a KRM deployed specifically in a business environment are described in detail.

3.3 Requirements for KRMs deployed in a business environment

Although key recovery mechanisms address problems arising from loss of decryption keys, they should always be deployed with extreme care. If the mechanism is not properly deployed it can seriously weaken security, as KR provides an alternative means of access to encryption keys that may be easier for an attacker to exploit than the original computation process. Thus, the fundamental security requirement for any KRM is that the effort to exploit and break the cryptographic infrastructure with KR added should not be less than the effort required if the cryptographic infrastructure lacks KR functionality. Moreover, the KRM deployed should not weaken the cryptographic mechanisms used. In particular, it should not necessitate the use of specific mechanisms and algorithms which may be weak.

Another obvious requirement is that honest users and agents should be able to successfully use the KRM, and, if possible, the deployed mechanism should be transparent to users and acceptable by users and agents. Moreover, the mechanism should not be vulnerable to rogue user attacks.

The following list gives requirements for a KRM deployed in a business environment. It is not unlikely, however, that most of the these requirements also apply in a LEA environment. For instance, some of these (in particular requirements R2, R3, R4, and R5) are identified in [45] where the authors consider them as a general framework for analysis of key recovery systems regardless of the environment that they are used. Detailed requirements regarding a general functional model of a key recovery system are also proposed by the National Institute of Standards and Technology in [66].

R1. *Non-circumventability*: The KRM should be infeasible to circumvent, i.e. users must not be able to use the cryptographic mechanisms while bypassing the KR information generation process. Further, the user should not be able to generate

invalid KR information or alter/delete it after its generation. It should be noted that a lot of mechanisms do not meet this requirement, making them vulnerable to rogue user attacks.

R2. *User completeness*: Honest users should succeed in making use of the KRM to produce valid KR information. This requirement covers any need for the availability of a server that will be involved in the KR information generation process, or of any public key material required by the KRM during this process. User completeness should also satisfy the users need for being able to use the KRM to recover their keys without the agent's intervention. This is particularly relevant to the use of a KRM for archived data, and is discussed in detail in Sections 3.5 and 4.1.

R3. *Agent completeness*: An authorised entity complying with the company's rules and policy should succeed in recovering the required keys. Any authorised attempts to recover keys, successful or not, should be logged for audit purposes.

R4. *User soundness*: As a result of the first requirement, any attempt to misuse the protocol by a dishonest user should be prevented, or at least be detectable. More specifically, a rogue user should not succeed in establishing a secure communication or encrypting stored data while producing invalid KR information without being detected and logged for audit purposes.

R5. *Agent soundness*: Any attempt by an agent to misuse the protocol or recover keys without complying with the company's policy should at least be detectable. Within this category of misuses fall both attempts by unauthorised entities and unauthorised attempts by authorised entities. Agent soundness can be achieved by the use of well-established access control mechanisms. Any unauthorised attempt to use the recovery process, successful or not, must be logged for audit purposes.

R6. *User acceptability*: The protocol should be acceptable to users. However, in a corporate environment this property is less important, since the use of the mechanism will typically be specified within the corporate security policy, and hence part of the conditions of employment. Nevertheless, factors that will help acceptability

include: making clear the benefits of its use, its flexibility, availability, compatibility and interoperability with existing schemes, and, probably most importantly, its efficiency of use (see property 11 below).

R7. *Agent acceptability*: The protocol should be acceptable to entities authorised to recover keys. The main factor that will lead to acceptability is the protocol's efficiency, i.e. it should be easy for agents to recover keys efficiently and quickly by using the protocol in accordance with the corporate security policy.

R8. *Policy compliance*: The protocol should operate within the corporate security policy, and should satisfy all relevant legal restrictions. Within the latter fall any constraints imposed by laws within the domain the corporation operates, such as those covering cryptographic algorithm use and export.

R9. *Flexibility*: The KR scheme should not prohibit the use of well-known and secure cryptographic mechanisms. This is closely related to the main security requirement that the KRM should not weaken or introduce any vulnerabilities to the cryptographic infrastructure by imposing restrictions on the mechanisms used.

R10. *Interoperability*: The mechanism should be capable of dealing with dissimilar KRMs, or cryptographic mechanisms that do not have KR functionality. This is particularly important for communications enhanced with a KR capability. A thorough analysis of the interoperability problem and its proposed solution is given in Chapters 6 and 7.

R11. *Mechanism transparency*: The KRM should be transparent to end users in that it should not introduce any significant computational overhead, or demand user interaction when the user employs the cryptographic mechanisms.

R12. *Negligible cost*: Deployment of the KRM should introduce only a negligible increase in the overall cost of the cryptographic services used and the security infrastructure. In particular, the cost of using the mechanism should not exceed the value of the information encrypted using an inaccessible key.

Although, as previously mentioned, some of these requirements are also identified as requirements for a LEA environment, others are of no relevance to a KRM deployed for LEA access. One example of such a requirement is interoperability of key recovery mechanisms. This is because law enforcement access typically seeks the wide (even global) deployment of a KRM which is potentially not expected to interact with other schemes. However, the variety of existing KRMs and their deployment in various business environments is expected to cause interoperability problems in encrypted communications.

Other requirements differ from those for LEA access in the degree that they should be met. For instance, while user acceptability has proven to be one of the crucial requirements for the deployment of a KRM for LEA needs, it is not expected that it will be an important factor in a business environment, assuming that the benefits from the use of a KRM will be made clear to the users. Cost is another issue that corporations are likely to consider, in contrast to governments which can typically tolerate higher expenses.

The above list of requirements can also serve as criteria for evaluating a KRM. All of them are likely to be essential for a sound mechanism, but factors that can influence their criticality should also be taken into account. For example, a slight weakening of the user and agent acceptability requirements is allowable if all the other requirements are satisfied and there is no alternative. The target of a KRM, that is whether it is used for communicated or archived encrypted data, can also affect the above requirements. A more detailed analysis of this distinction is given in Chapters 4 and 6.

There are other issues involved in using KRMs. KRMs require the existence of an infrastructure supporting key management techniques meeting the requirements of the mechanism. Clearly the administration and cost of the required infrastructure are factors that cannot be ignored. Moreover, if the mechanism is not appropriately deployed a number of security issues might arise. Storage of any KR information should be

carefully protected to prevent unauthorised access. If appropriate access control restrictions on the recovery process are not enforced, an adversary without the required privileges might be able to recover keys and monitor communications or decrypt stored encrypted files. More important still, if the infrastructure lacks the existence of an audit log mechanism where all attempts to recover keys are monitored and reviewed, then the system could be significantly more vulnerable to compromise than it would be if it lacked KR functionality.

These requirements may be more difficult to meet if the corporation decides not to perform key recovery internally but rather to outsource the service. While there are certain risks associated with outsourcing the service (the agent might relax its security policies, go bankrupt, or even be bought out by a competitor while retaining the ability to recover keys [1]) this solution might be more attractive to small and medium-sized enterprises that are not willing to deploy their own infrastructure.

3.4 Assessment of existing mechanisms

In the following sections, the applicability of existing KRMs to a business environment is investigated. More specifically, the two main types of KRMs are considered, i.e. *key escrow* and *key encapsulation*, taking into account only their general characteristics as described in Section 2.4.

3.4.1 Assessment of key escrow mechanisms

When key escrow mechanisms are used in a business environment, there is typically a need for a large storage capacity for the escrowed information and, more importantly, this information must be protected from unauthorised access. The latter is one of the main drawbacks of key escrow mechanisms, as pointed out frequently by their opponents. As mentioned in [1], the storage of all keying material at a single point

makes it a significant point of attack, and introduces a major vulnerability to the key escrow mechanism if appropriate countermeasures are not in place. The main protection mechanism that can be employed for this purpose is strong access control, ensuring that only authorised personnel can access the escrowed key material and recover user keys. Access control should be enforced in conjunction with appropriate audit log mechanisms that will enable the monitoring of all attempts to access the escrowed keys and make use of the recovery process. Dispersion of the key material to multiple locations, resistance to agents' collusion, and residual work factor (see Section 2.9) can be used as additional countermeasures. If all these protection mechanisms are in place, supported by an appropriate security policy, the likelihood of misuse of the KRM can be minimised.

Key escrow mechanisms are likely to be integrated into a cryptographic infrastructure, i.e. the KR functionality will be closely related to the key establishment process. For these mechanisms to work properly and not to face interoperability problems (requirement R10; see Section 6 for more details), there is a need for a common infrastructure meeting the mechanism's requirements. In other words, such mechanisms usually demand the existence of specific key establishment protocols, a requirement that can cause interoperability problems in communications sessions, making them difficult to deploy world-wide. As a consequence, if they are deployed by a corporation they may constitute a barrier to encrypted internet communications with other organisations. Even if the KR information can be generated solely at the user's end, with no requirement for interaction with the peer, the dependence on the cryptographic infrastructure would require the communicating parties to use cryptographic mechanisms compatible with the KRM.

Such mechanisms are, however, potentially more appropriate for intranet communications, where it is easy to establish a common infrastructure, and also for archived data encryption. If the KRM is part of the cryptographic infrastructure, and dependent upon it, circumventing it will typically require the rogue user to circumvent the whole

cryptographic functionality, and hence not use the provided mechanisms. In controlled environments, such as a corporation, it is not infeasible to restrict the user's resources, and hence requirements R1, R3, and R8 can be efficiently met. This is not the case for key encapsulation mechanisms as will be described later, or even for those key escrow mechanisms that are less dependent on specific key establishment protocols.

When assessing key escrow mechanisms it is useful to make a further distinction between those that at least sometimes require the on-line participation of a TTP acting as an escrow agent and which assists in the session key establishment process (such as the JMW scheme [39]), and those that do not. In the first category fall mechanisms such as the ones where the escrow agent acts as a *key distribution centre* (KDC) or *key translation centre* (KTC), and in the meantime escrows all the keys that the users establish. For this class of key escrow mechanisms, an on-line server, which must typically be able to deal with a large number of simultaneous requests, will be involved in all, or at least a significant number of, the key establishment processes. The agent's on-line participation makes these mechanisms the most difficult to circumvent, and rogue user attacks are difficult to mount. Furthermore, they give the agents more control over the KR information than with other mechanisms, a scenario that fits the business model (assuming that agents are internal to, and managed by, the organisation). These properties are particularly relevant to requirements R1, R3, and R7, which this type of key escrow mechanism can efficiently satisfy. Compromise of this server, however, would have unpredictable consequences. An adversary in control of the server functionality would typically be able to recover all the established keys and decrypt communicated or archived data. Moreover, agent unavailability would mean that users are unable to encrypt data, hence requirement R2 will not always be satisfied. Thus, protection of the server against unauthorised use and denial of service attacks becomes a fundamental issue.

In the second category fall mechanisms that are less dependent on an on-line escrow agent. For these mechanisms, the user escrows certain key-related information, typi-

cally during the initialisation phase, which enables the agent to recover session keys subsequently generated by the user. As an example, consider a scheme where the sender encrypts the session key under the recipient's public key, while the corresponding private key is escrowed with the user's agent. Although there is no need for the agent to be on-line for these mechanisms, avoiding any availability requirements, they are no more flexible than mechanisms of the first category, since they also typically need an infrastructure to assist in all the cryptographic computations. For instance, in the given example users need to possess each others' valid certificates. This means that a complete certificate management scheme is required, including an inter-organisational public-key infrastructure (PKI) (e.g. a certificate repository, and means to generate and manage certificate revocation lists).

Cost (requirement R12) is another important consideration, especially in a commercial environment. Although deployment costs might be acceptable, long term administrative costs cannot be ignored. Key escrow mechanisms require provisions to protect the escrowed key material, and in that respect are potentially expensive. Although the cost might be significantly reduced if an external agent is used, as previously mentioned there are clear potential disadvantages of such an approach.

Summarising the above, key escrow mechanisms can efficiently satisfy *non-circumventability* (R1), especially in cases where the key recovery agent controls the key establishment process. As a result *agent completeness* (R3) and *policy compliance* (R8) will also be satisfied. Problems, however, might arise if an on-line agent is used for the key establishment process, whose unavailability can affect *user completeness* (R2) and *user acceptability* (R6). Finally, key escrow mechanisms are likely to suffer from interoperability problems, and hence not satisfy requirement R10, and their cost (R12) cannot be considered negligible.

3.4.2 Assessment of key encapsulation mechanisms

When used in a corporate environment, the majority of key encapsulation mechanisms appear to be more flexible than key escrow schemes. Being independent of the key establishment technique means that protocols can easily be adapted to them and, unlike key escrow schemes, they are unlikely to suffer from key management related interoperability problems and hence, they can satisfy requirement R10. The data encryption keys will typically be encrypted under KRM-specified public key(s), with no restrictions on their nature or generation (requirement R9). Interoperability, however, can be affected in communication sessions by the interaction the mechanism might require with the remote party prior to generation and/or for verification of the KR information. This might include mechanism-specific public keys that the originating party needs to generate the KR information, or any verification checks the peer is required to perform on the received KR information prior to decryption.

Unlike some key escrow mechanisms, a key encapsulation infrastructure would typically not require a high powered on-line server, as there is no need for on-line interaction during the KR information generation process (of course, such a server may be necessitated by other key management requirements). Therefore, user acceptability (requirement R6), as far as the availability of the KRM is concerned, will always be satisfied. Deployment of such a mechanism in a corporate environment, however, might necessitate checks on transmitted KR information to ensure users comply with the company's policy (requirements R3, R4, and R8). This is because key encapsulation mechanisms are vulnerable to *cut-and-paste* attacks, where a rogue user can alter or delete the KR information after its generation to disable subsequent key recovery by an authorised entity. To prevent this, and hence satisfy requirements R1, R3 and R7, authorised entities could run checks on the generated KR information to ensure that the KRM has been properly used (assuming that such checks are supported by the KRM); such checks could be made at random, or only if rogue activity is suspected.

One way of preventing rogue user attacks is to check intercepted KR information. If the intercepted information is invalid, the transmitted data could be prevented from leaving the organisation's domain. Rogue user attacks can also be prevented by requiring the validation of the KR information by the receiving party prior to decryption of the received data. This latter solution, however, requires trust in the receiving entity, which is not always the case, especially if the latter is not within the company's domain. Thus, a drawback of key encapsulation mechanisms is that, in order to ensure that the mechanism is not circumvented, there is a potential need for on-line checks on the generated KR information. This checking can prevent both single rogue user attacks that are not prevented by the mechanism itself, and double rogue user attacks where the colluding entities agree to make use of the organisation's cryptographic mechanisms but bypass the key recovery process by tampering with the key recovery information.

On-line checks on the key recovery fields might be trivial if the key recovery agent operates within the company's domain. In that case a server that resides behind the organisation's firewall can be used for this purpose. However, verification of the KRF might be harder if the agent is external to the organisation. Only checks can be performed in this case, and then only if the information is intercepted during its transmission, or if all communications are routed through an agent's server. This is because the data may already have left the company's control and have reached their destination. Thus, in this case, *detection* of rogue users remains possible, but the capability for *prevention* is much more limited. Of course, it may be the case that detection is sufficient, for actions can be taken against rogue users as soon as a single instance of system misuse is detected. However, in situations where misuse must always be prevented, it will probably be appropriate to have an internal rather than an external agent.

Note that existing key recovery mechanisms typically do not support third party validation of KR information, a property that can help meet requirements R1, R3, R4, and R8. A model designed for this purpose was proposed in [89] but, as described in

[71], the proposed scheme suffers from superencryption attacks. Simple techniques can be employed, e.g. demanding the encryption of a data field known to the agent with the key used for encrypting the rest of the data. Decrypting this field with the key contained in the KRF would give the agent an indication as to the validity of the KRF. Even with these techniques, however, mechanisms can still be vulnerable to rogue user attacks. Probably the most effective solution is to decrypt the data, and search for expected patterns that should, or should not (in the case of malicious software), be present in the transmitted data.

Key encapsulation mechanisms are even more vulnerable to cut-and-paste attacks when the mechanism is used on encrypted archived data. It would typically be too costly to check the validity, either on-line or off-line, of every KRF produced for every encrypted file. Therefore, a rogue user could tamper with the KRF by simply deleting or modifying the field after its generation, thereby disabling any authorised KR attempt. This will typically constitute a breach of policy (requirement R8), and action could be taken against that employee, who might even lose his job. However, from the company's perspective the data are lost, and hence the KRM has failed. In such a case, it would be more appropriate to use a key escrow mechanism which will give the company the ability to have more control over the generated keys.

Another relevant issue is that, in key encapsulation mechanisms, the management of the keys is left with the user. In hierarchical environments this property might cause problems if different agent public keys are used to protect KRFs for different levels of classification of information. The user will have to decide which key he should use for encrypting the key recovery information, depending on the sensitivity of the data. This might lead to confusion, and even accidental or deliberate misuse of the KRM. This is not always the case with key escrow mechanisms, where the escrow agent can manage the generated data encryption keys.

Key encapsulation mechanisms are not inherently more secure than key escrow mecha-

nisms. Although for the latter there is a need to protect all the escrowed key material, and unauthorised access to it would typically give the attacker access to data encryption keys, the compromise of the agent's private decryption key in a key encapsulation mechanism would have the same unacceptable consequences.

Finally, key encapsulation mechanisms appear to have potentially lower management costs than key escrow schemes (requirement R12). For such a mechanism there only needs to be a cryptographic infrastructure. There is no need for on-line participation of the agent, which potentially requires a high powered server (unless the corporation demands on-line checks on generated KRFs), or for the secure storage of all the escrowed keys. However, these cost estimates might be altered, depending on the mechanism's implementation. For instance, if it is decided to deploy user smart cards, then the cost of issuing a card for each employee may not be negligible.

Summarising the above, key encapsulation mechanisms are more susceptible to rogue user attacks than key escrow schemes, and thus they do not always satisfy the *non-circumventability* requirement (R1). Susceptibility to rogue user attacks is likely to have the same effect on *agent completeness* (R3), *agent acceptability* (R7), and *policy compliance* (R8). However, as key encapsulation mechanisms can typically work with any key establishment protocol and key encryption algorithm, they are quite *flexible* (R9), and they cause less *interoperability* (R10) problems than key escrow mechanisms. Finally, their cost of deployment (R12), although it cannot be considered negligible, is likely to be less than key escrow mechanisms.

3.5 Distinguishing between communicated and archived data

The above analysis of KRMs in a business environment has not considered the target data. There are certain issues, however, that need to be addressed as far as the target of the KRM is concerned. This arises from the fact that there are different requirements

for KRMs for archived data and KRMs for communicated data.

The majority of existing key recovery mechanisms were designed for use with communicated data, and with the objective of giving access to LEAs. Giving user access was typically not a design requirement mainly because users would not benefit from such a property. As mentioned in [1], "there is hardly ever a reason for an encryption user to want to recover the key used to protect a communication session". If the session key is lost during an encrypted session, a new session can be established and a new key can be negotiated. This, however, does not rule out business demands for access to encrypted communications.

With communicated data, interoperability of the deployed KRMs is the most important requirement. Otherwise, use of dissimilar mechanisms might prohibit the establishment of a secure communication session. Interoperability is an issue that is covered in detail in Chapters 6 and 7.

With archived data, the requirements are rather different, making the distinction essential. With archived data, the focus of the design of the KRM should also consider the users' needs for recovery of data. In other words, apart from fulfilling third parties' needs, the KRM should also be the users' means for access to lost keys. It would be a waste of communications resources and processing power if the user had to contact his agent whenever he wants access to his encrypted data or requests recovery of a lost key. Moreover, interoperability is no longer an issue, as encryption of archived data will typically only involve one entity, and hence only one KRM. As a result of this, however, the mechanism will be more susceptible to rogue user attacks where the user might deliberately delete or alter the generated KR information.

Electronic mail is a special case because it has the characteristics of both communicated and archived data. If the decryption key for an encrypted e-mail is lost, access to the email will be infeasible unless the sender re-sends the message. The ability to recover

keys used for encrypting e-mail could be of a potential benefit to the user.

Therefore, the differences that necessitate the distinction between KRMs used for communicated data and those used for archived data are as follows.

- **Interoperability**. While interoperability is a critical requirement for KRMs used for encrypted communicated data it is of no importance to KRMs for archived data.
- **Susceptibility to rogue user attacks**. In encrypted communications, attacks where a rogue user tampers with the generated KR information can be prevented by requiring the receiving party to verify it prior to decryption (although this might not always be an efficient countermeasure against these attacks). This is not possible with archived data, however, as during a typical encryption of archived data there will be only one entity involved.
- **Users' ability to recover their keys unaided**. While users typically do not benefit from being able to recover keys used for their encrypted communications, the situation is rather different for archived data.

The next two parts of this thesis consider these two types of KRMs in more detail.

3.6 Summary

In this chapter, the possible dangers to a corporation arising from an inability to access keys used for encrypting data have been considered. An analysis has been made of the requirements for KRMs applied in a business environment, and the applicability of existing mechanisms was investigated. More specifically, the deployment of the two main types of KRMs, i.e. key escrow and key encapsulation, has been assessed on the basis of their general properties. As there is no panacea to the key recovery problem,

careful analysis of the business needs is necessary to identify appropriate solutions.

A further distinction was made between requirements for KRMs for communicated data and for archived data. This distinction has motivated the work described in the next two parts of this thesis.

Part II

KEY RECOVERY FOR ARCHIVED DATA

Chapter 4

Applying key recovery to archived data

Contents

In this chapter we identify the requirements for a key recovery mechanism used specifically for archived data, and we propose two schemes where keys used for stored data encryption can easily be recovered.

4.1 Requirements for KRMs for archived data

Keys used for encrypting stored data should be saved so that later decryption will be feasible. Storing these keys unencrypted makes them subject to unauthorised access. If, however, they are protected by a private key (e.g. password) a question arises as to where this key is stored. Requiring the user to memorise it is very risky (it is not unlikely that the user will forget it), while, if it is stored on a secure device such as a smart card, then there is a risk that the device will fail, or will be lost or destroyed.

Archived data management typically involves only one entity, which stores and retrieves data at distinct points in time [59]. Keys used by this entity to encrypt stored company data are likely to be possessed only by this entity. Therefore, decryption of archived data by a third party (a crucial requirement in a business environment) will be infeasible unless a KRM is in place that will enable recovery of the data decryption keys.

Previously proposed KRMs were mainly designed to provide KR functionality for encrypted communications. As we discussed in Chapter 3, when these mechanisms are applied to archived data (especially key encapsulation schemes [83, 91]), they suffer from the absence of a second party that can verify the generated KR information. As a result, they become particularly vulnerable to rogue user attacks, where a rogue user can tamper with (alter or delete) the KR information during or after its generation, making it invalid to third parties. While this kind of attack on some KRMs, such as key encapsulation schemes, is typically prevented in a data communication environment by requiring the receiving party to verify the KR information prior to decryption of the received data, this is infeasible for stored data encryption.

Another problem with most existing KRMs is that they do not offer the user the ability to recover his keys unaided, hence forcing him to contact his agent, who will recover the keys on the user's behalf. This problem arises from the fact that the intention of the majority of the proposed mechanisms was to give LEAs access to encrypted

communicated data. Designs have therefore typically focused on giving access to keys to the on-line agent rather than the user himself, i.e. the user is given no means to recover his data encryption keys unaided. As a result, applying these mechanisms to encrypted archived data introduces extra communications costs from the necessary interaction with the agent. User key recovery will also necessitate the existence of an on-line agent.

For example, consider a key encapsulation mechanism where the KRF contains the data encryption key encrypted under the agent's public key. Whenever the user wants to access an encrypted file, and assuming that a copy of the key is not kept locally, he has to ask the agent to recover the decryption key from the KRF. Problems will then arise when the user works off-line, or if there is no connection with the agent because of a network failure; the user will be unable to use the KRM to recover his keys because of the agent's unavailability. A solution to this problem is for the user to have a backup of the keys stored locally, and whenever he requires decryption of a file the copy of the key is used instead of the KRM. This solution, however, introduces new risks.

A key recovery mechanism that would be specifically designed for use with encrypted archived data while overcoming the aforementioned problems should typically satisfy the following requirements. Note that this is not a complete list of requirements; it rather highlights those requirements that are of particular relevance to the use of a KRM for archived data, and which are not satisfied by the majority of the existing mechanisms.

1. The KRA should have the ability to recover the required keys, even when the user tampers with the generated KR information. This is particularly relevant to key encapsulation mechanisms where the KRA typically does not have direct control over the key generation process.

2. The mechanism should give the user the ability to recover his keys unaided, i.e. without the KRA's intervention. This will ensure that the user has continuous

access to his keys while avoiding the need to keep a backup of them locally (an approach that introduces new threats to the secrecy of the keys).

Further to these requirements, it will be an advantage if the KRM makes no demands for an on-line agent or for storage of users' key related material.

4.2 Applicability of existing mechanisms

To examine the applicability of existing KRMs, we start by making the same distinction as in Section 3.4.1, that is, between key escrow mechanisms that require the existence of an on-line TTP and those that do not.

1. Mechanisms requiring on-line participation of the escrow agent for key generation appear to be more secure from the non-circumventability point of view. As they are typically integrated within the cryptographic mechanisms, circumventing the KRM would require the user to avoid making use of the cryptographic mechanisms, and hence make no use of encryption. One example of such a mechanism is provided by a KDC that generates keys on the users' behalf while a copy of the key is kept by the agent. Yet, as already mentioned, these mechanisms suffer from the requirement for a high-powered server that will participate in the key generation process.

2. KRMs that are less dependent on on-line agent participation are weaker as far as circumventability is concerned. Within this class, however, fall some mechanisms that do not have this problem. For example, consider a KRM where a user escrows a Master Key with his agent. He subsequently uses the Master Key, combined with some additional data that the agent is assumed to know in advance, to compute the session key. Such a scheme does not require the existence of an on-line TTP. Yet, the TTP has direct access to the generated key, and this is the characteristic distinguishing it from other mechanisms of the same category.

Both these types of mechanisms suffer from the problems previously mentioned regarding key escrow mechanisms. That is, deployment of such a scheme would require the existence of large storage devices for the administration of the escrowed keys, which require the use of strong security mechanisms for their protection.

Key encapsulation mechanisms used for encrypted archived data have the advantage that neither an on-line agent nor storage of any KR material by the user's agent is required. Although key encapsulation mechanisms increase the amount of data that have to be stored, as the encrypted key has to be attached to the encrypted data, we assume that this is not a problem in most applications. Moreover, the session key is adequately protected as it is encrypted, together with some mechanism-specific credentials, under the public key of the user's agent. Key encapsulation mechanisms, however, are vulnerable to cut-and-paste attacks where a rogue user alters or deletes the generated KR information, preventing access to the encrypted data by authorised entities. So, as far as circumventability is concerned, these mechanisms appear to be more vulnerable than key escrow.

A KRM designed for use with archived data, as well as with encrypted messages and session keys, was proposed by Maher, [53]. The Maher *Crypto backup* system provides corporations access to keys that are lost, destroyed, or withheld from legitimate access, by automatically making "a backup key through a controlled process". The system requires agents and users to share a value x and a function f with the following properties.

- The function is easy to compute, but its inverse in r is computationally infeasible. More specifically, r is hard to compute from the value $f(x, r)$ when the value x is known.
- For any two values r_1 and r_2, $f(f(x, r_1), r_2) = f(f(x, r_2), r_1)$.

One example of a suitable choice for f is obtained by putting $f(x, r) = x^r \bmod p$, where p is a large prime and x is chosen to be an element of large prime multiplicative order modulo p.

The Crypto Backup process is implemented by the following steps.

1. The value x and the function f are fixed.
2. The agent, namely *Trusted Backup Agent*, generates a random value r_M, which is the agent's private MasterKey, and computes the corresponding public MasterKey as $y = f(x, r_M)$.
3. The public MasterKey together with the agent's identity is signed by an appropriate authority, thus binding the agent's identity to the agent's public MasterKey.
4. A user U of the backup system, generates a random number r_U and constructs the key k to be used for the file or message encryption as $k = proj_n(f(y, r_U))$, where $proj_n$ selects the first n bits of its argument.
5. User U generates the Backup Recovery Vector (BRV), which is a vector that contains information identifying the agent and information sufficient to recover the corresponding key k for a single file or message, as $(id_A, id_y, f(x, r_U))$. Following its generation, either the BRV is placed in the file header or an appropriate pointer to a BRV file is provided to the agent.

Although the described mechanism does not require the agent's participation in session key generation, it suffers from the problem that a user can tamper with the generated key recovery information and prevent key recovery. It might be argued though, that if the key recovery process is automatic and transparent to the user, then rogue user attacks are relatively unlikely. However, in many circumstances users will nevertheless be aware that key recovery is taking place, and hence might wish to take steps to prevent key recovery operating correctly. Therefore, a rogue user can typically alter or

delete the attached BRV, or submit an invalid BRV to the BRV file, hence disabling key recovery by the agent. Moreover, the user cannot recover the keys used without the agent's participation, a property that makes the mechanism less attractive for use with archived data.

Two different schemes are proposed in Sections 4.3 and 4.4 that effectively overcome these problems, with the first being an adaptation of Maher's scheme. The two mechanisms have the following common properties.

- Users do not require their agent's intervention to recover keys previously used.
- The mechanisms are not vulnerable to rogue user attacks on the generated KR information.
- The mechanisms make no demands for the user to escrow with the agent any key related material.

Therefore, the proposed mechanisms avoid both the key encapsulation mechanisms vulnerability to rogue user attacks, and the key escrow mechanisms' demands for storage and protection of users' key related material. The main difference between the two schemes is that the second KRM requires user communication with the agent(s) during key generation, and mandates the use of smart cards. Use of smart cards in the first mechanism is optional, although, as discussed below, their deployment can help prevent rogue user attacks.

Both schemes are 'text expanding', i.e. the encrypted string will be longer than the cleartext string. Whilst this will not be a problem in most applications, there exist certain special circumstances, for example the encryption of individual database records, where this is a serious problem. Dealing with such special cases is outside the scope of this thesis.

As mentioned in Chapter 2, for rogue user attacks we assume that a rogue user, trying to disable authorised KR by his associated KRA, may tamper with the generated KR information by either altering or deleting it, or may even prevent its generation. However, we assume that the user will leave the encrypted data unchanged so that he can recover it when necessary (if the encrypted data is modified or destroyed, then no KRM can deal with the situation). For instance, the rogue user might simply detach the KR information from the encrypted file, and retain the KR information separately so that it is not available to the KRA. Through possession of this information the user can recover the required key but the KRA cannot.

4.3 A new KR scheme

4.3.1 Specification of scheme

Whenever a user wants to encrypt a file or message, instead of generating a random key and using it to encrypt the stored data, he uses the proposed mechanism which will also allow later recovery of the generated key. For this mechanism the following requirements must be satisfied:

1. All entities share public values p and g, where p is a large prime number and g is an element of large prime multiplicative order modulo p. We will write g^x for $(g^x \bmod p)$ throughout.

2. Each user and each KRA has a Diffie-Hellman [20] key agreement key pair. More specifically, user A has private key agreement key x with corresponding public key g^x, and KRA C has private key agreement key y with corresponding public key g^y. The user's key pair can be generated by either the user or a certification authority (CA) specified in the corporation's policy (or even by both of them [62]), and the public key is certified by the latter. If the CA and KRA are not the same entity, the KRA can also get its public key agreement key certified by

a CA in the hierarchy.

3. The entities share a one-way hash function h.

4. A repository for distributing the certified public keys has to be in place.

The *KR information generation phase* consists of the following steps:

1. When a user wants to encrypt archived data, a KRA is selected depending on the demands of the data to be encrypted (note that this can be an automated process), and obtains the KRA's public key certificate from the repository.

2. The user generates session key K as:

$$K = h((g^y)^x \bmod p \parallel \text{“data credentials”})$$

where $\parallel$ denotes concatenation, x is the user's private key and g^y is the KRA's public key agreement key. The “data credentials” can vary according to the application of the KRM. If the data is a file, the “data credentials” can be derived from the file's unique characteristics, e.g. the date of the file's last modification, and/or the name of the file. If the data is a message, a unique serial number that identifies the message is sufficient. The generated key, or part of it, depending on the requirements of the encryption mechanism, can be used for the encryption of the message or file.

3. A *key recovery field* (KRF) is generated and attached to the encrypted data (message or file). The KRF consists of the identity, id_{KRA}, of the agent whose public key agreement key g^y was used to compute the session key, a serial number id_y that identifies the agent's public key if the agent has multiple public keys, and the “data credentials” used to compute K. That is, the KRF is:

$$KRF = (id_{KRA}, id_y, \text{“data credentials”})$$

When a user wants to decrypt the data, he simply recomputes the key K by executing step 2 from the *KR information generation phase*. That is, during the *user key recovery phase*:

1. The user detaches the KRF from the encrypted data.
2. Using the agent's identity id_{KRA} and the key serial number id_y, both contained in the KRF, the user obtains from the repository the certificate for the appropriate agent public key (if he has not already got a valid copy stored locally).
3. Using his private key agreement x, the KRA's public key agreement key and the "data credentials" field, contained in the KRF, the user can re-compute the required key.

If an authorised entity wants to recover a key, he detaches the KRF and sends it to the KRA identified by the id_{KRA} field, along with the appropriate authorised request. The requesting entity must also inform the KRA who the file owner is, i.e. the user that encrypted the file, so that the KRA can retrieve the appropriate user's public key certificate. The latter task can be left to the requesting entity which, in this case, must pass the certificate to the KRA. If the recovery request complies with the relevant security policy the agent proceeds with the *agent key recovery phase*:

1. The agent extracts from the received KRF the "data credentials" and the value id_y, which will identify the agent's key used. Moreover, the agent retrieves the user's public key agreement key if this is not received from the requesting entity.
2. The agent is now in possession of all the key material required for the recovery of the requested key, which the agent can easily compute.

4.3.2 Properties and security analysis

From the above description it should be clear that, during the *user key recovery phase* there is no need for interaction with the agent, a property that simplifies the key recovery process for the user. More specifically, when a user wants to recover a key that he has previously used to encrypt archived data, the agent need not participate in the key recovery process. The user is able to recover the keys himself, without the KRA's intervention.

The majority of KRMs lack such a feature; the user's KRA is typically the only entity that can recover the key. With the proposed scheme the user will be required to contact his agent only if he has lost his private key agreement key, in which case the only entity that can recover the user's keys is the KRA. This property typically eliminates the requirement for an on-line agent and avoids the related communications overhead. Further properties of the proposed mechanism include:

1. The agent and the user are the only entities that can access the data encryption keys. Encryption of a file by a user that is not its owner, if this is allowed by the access control system, will give ownership of the encrypted file to that user. This will ensure that when the agent attempts to recover the corresponding key, it will use the appropriate user public key. If the operating system cannot assign ownership of the encrypted file or message, the user's identity can be included in the *KRF* as an alternative. Moreover, if a user can have multiple keys, the *KRF* should also contain an identifier for the one used.

2. There is no need for the agent to store any user-specific key material. The only storage requirement is that there must exist a repository from where the certificates, for both users' and agents' public key agreement keys, can be easily obtained. Each agent, however, has to ensure that its private key agreement key(s) are protected against disclosure to unauthorised third parties.

3. The "data credentials" field used in the key generation process ensures the uniqueness of the generated key for each file or message encrypted, assuming these data are unique. Thus, each key recovery request by an authorised entity will be restricted to a single file or message.

4. Although with the proposed scheme the encrypted data will be longer than the cleartext this is not a problem in most applications. Those special cases where 'text-expansion' might be a problem are outside the scope of this thesis.

5. The corporation's environment can be divided up and multiple agents' key agreement keys can be used, each one corresponding to a single domain, group of users, or level of information classification. The management can thus ensure that a single agent's key agreement key can only be used to recover specific types of data encryption keys.

6. The proposed scheme is not restricted to encryption of archived data. It can also be used in encrypted communications, given the existence of an independent means for conveying the generated key to the receiving party. In that case, however, the peer will not be able to verify the validity of the generated KR information, although this may not be a problem since the mechanism is not vulnerable to rogue user attacks.

7. Unlike many KRMs applied to encrypted archived data, the proposed scheme is not vulnerable to rogue user attacks. If a rogue user tampers with the KR information, aiming to disable any authorised KR attempt by the agent, the proposed scheme offers an alternative means for the agent to recover the keys. Specifically, if only a limited number of bytes are used as the "data credentials" for the key computation, the agent can mount an exhaustive key search, as described in Section 4.3.3, which will recover the required key in a reasonable time period.

The following remarks can also be made concerning the security of the proposed mechanism.

1. A rogue user who wants to disable key recovery on his encrypted files has to subvert the whole key establishment process. Appropriate software integrity checks [4] can ensure the proper use of the cryptographic infrastructure.

2. To avoid an attack where a rogue user generates his own Diffie-Hellman key pair and uses it to generate a session key, an on-line check can be run on the validity of the certificate for the public key agreement key used. That is, if necessary, the mechanism can pass the public key agreement key being used to the agent over a secure channel. The agent can thereby check its validity in a user-transparent way. This on-line check can be avoided, however, if the key is stored in a smart card, as described in Section 5.3, assuming the user is not able to create 'bogus', apparently legitimate, smart cards.

3. Use of smart cards can also prevent a rogue user from manipulating the key after its generation, e.g. the user can use the generated key as input to a keyed function while using the output of this function as the session key. To prevent such an attack the whole mechanism should be implemented on the card, i.e. encryption must take place on the card, although there are certain limitations associated with such an approach, as described in Chapter 5. An alternative is also proposed in the next chapter.

As with the mechanism proposed by Maher [53] this scheme can also be modified to reduce the trust that needs to be placed in one agent. Each user can use keys from multiple agents, thus eliminating any unauthorised attempt by a single agent to recover encryption keys. More specifically, assume, for simplicity, that the user wants to split his trust between two agents and that these two agents have private key agreement keys y_1 and y_2 and corresponding public key agreement keys g^{y_1} and g^{y_2} respectively. When generating an encryption key, instead of simply using one agent's public key agreement key, both agents' public keys are used:

$$K = h(g^{x(y_1+y_2)} \bmod p \parallel \text{“data credentials”})$$

During the *key recovery phase*, the user can reconstruct the key following exactly the same procedure as before. An agent that wants to recover a requested key must obtain the user's public key agreement key raised to the power of the other agent's private key agreement key. Thus both agents must participate in key recovery.

4.3.3 Defeating rogue user attacks

The agent, as previously mentioned, can mount an exhaustive key search for the requested key if the user tampers with the KR information. Performing a known-plaintext exhaustive key search would typically require the knowledge of at least as many bits of plaintext as there are bits in the unknown "data credentials" used to derive the secret key. Therefore, we propose prefixing the data to be encrypted with a fixed bit string, known to the KRA, of sufficient length to enable the correct key to be determined. Note that, in normal use, the encryption software will delete this fixed string during decryption. By this means, if there are 2^k possibilities for the "data credentials", then finding the correct key is expected to take approximately 2^{k-1} trials.

Given the above requirements, it should be clear that, for current technology the "data credentials" field should not exceed 6 bytes in length, i.e. $k = 48$, if a result is required within a short time period (not more than a couple of days) and at reasonable cost. Nevertheless, if the bytes making up the "data credentials" are highly redundant, e.g. if each byte is an ASCII-coded alphanumeric character, then using a larger number of bytes for these credentials will become possible. Ultimately, the size of the "data credentials" will depend on the resources and the time that a corporation is willing to spend on an exhaustive search for the keys. This, in turn, will depend on the availability of key searching technology – note that, for example, in [93] it is estimated that a $10,000 machine can find a 56-bit DES key in 2.5 days.

If the rogue user also attempts to delete the encrypted fixed string from the encrypted data, then a ciphertext-only key search can be attempted. As described in [59], "if the

underlying plaintext is known to contain redundancy, then ciphertext-only key search is possible with a relatively small number of ciphertexts". In that case the criteria for a correct key would be characteristics that are expected to be found in the decrypted text, e.g. the coding of the expected plaintext.

Further suppose that a rogue entity, in an attempt to circumvent the key recovery process, uses the cryptographic infrastructure to perform superencryption, at the same time deleting the key recovery information at each stage. If the encrypted fixed strings used to prefix the data to be encrypted are left in place, then a series of brute force searches, each of expected size 2^{k-1}, will be sufficient to recover the plaintext, even if different "data credentials" are used for each iteration of the superencryption process.

It could be argued that rogue users who do superencryption are smart enough to circumvent the whole key recovery process by other means. However, performing superencryption may only require a mouse click, and hence will be very easy to perform for even the most non-technical of users. Taking other measures to defeat key recovery (e.g. implementing another encryption algorithm) will not be an option for such non-technical users.

Finally, if a rogue user performed superencryption while also deleting the encrypted fixed strings, then recovery of the encrypted data may become infeasible, as it will now probably be necessary to perform a simultaneous brute force search over the entire set of keys used for superencryption. However, this seems a relatively minor threat, since the amount of work involved for the rogue user may be more than simply introducing a different cryptographic infrastructure.

4.4 A second new KR scheme

4.4.1 Specification of scheme

We now describe another mechanism for adding KR functionality to the encryption process for archived data. This mechanism requires the user's interaction with the agent during key generation, and it is this interaction that helps prevent rogue user attacks.

Three entities are involved in the proposed mechanism: the *user* who encrypts the data, the KRA which assists in the management of keys, and the *authorised entity AE* which is the entity authorised by the corporate policy to have access to users' data.

Whenever a user wants to encrypt a file or message, instead of generating a random key for data encryption, he uses the proposed mechanism for key generation, which will also allow later recovery of the generated key. For this mechanism, which necessitates the use of a smart card by each user, the following requirements must be satisfied:

1. The KRA and the user's card share a *message authentication code* (MAC) function $f1$, a one-way hash function h, and a key generating function $f2$ (this could potentially be based on a one-way hash function). $f2$ is used to generate the key K_{AC} that will be used by the MAC function $f1$, i.e. $K_{AC} = f2(K_M, id_A)$, where K_M is the KRA's master key and id_A is user A's identity. K_{AC} should be stored on the user's card, typically during card personalisation. The user must not have access to K_{AC}, otherwise rogue user attacks become possible.

2. The KRA, user, and authorised entity share a key generating function $f3$. As with $f2$, $f3$ can be based on a one-way hash function.

3. The user's card and the AE share a secret key K_{AM} which is generated as a function of K_A and a secret master key K_{AE} that the authorised entity possesses,

i.e. $K_{AM} = f3(K_{AE}, K_A)$. K_A is a master key specific to user A, which is generated as a function of KRA's key K_M and the user's identity id_A, i.e. $K_A = f3(K_M, id_A)$. As with K_{AC}, K_{AM} should also be stored on the user's card during the card personalisation.

4. The user has access to a random number generator. The generated random numbers are used to ensure key freshness so that a file will never be encrypted with the same key more than once.

5. The KRA administers a file consisting of indexes binding a unique file identifier with a random value generated for the specific file. The integrity of this file must be preserved.

6. All the entities trust the device where the encryption takes place, i.e. the user's PC or a server. If this is not the case then encryption has to take place on the card, although there are certain performance limitations associated with this approach (see Chapter 5).

When user A wants to encrypt a file, a session key K_S has to be generated using the following protocol.

1. A generates a random value $RAND$ either on his card or on the PC. Using his card, A computes a MAC on $RAND$ and the unique identifier *fileid* of the file to be encrypted, i.e. $MAC_1 = f1_{K_{AC}}(RAND, fileid)$. A then sends the following message to the KRA,

$$\text{A} \xrightarrow{idA \parallel RAND \parallel fileid \parallel MAC_1} \text{KRA}$$

   where $\parallel$ denotes concatenation.

2. Upon receipt of the message, the KRA uses the received user's identity id_A and the master key K_M to compute the key K_{AC}. The KRA then recomputes the message authentication code MAC_1 using the received values $RAND$ and *fileid*

and checks the result against the received MAC_1. If the check succeeds the KRA adds an entry to the index file consisting of the received *fileid* and the random value *RAND*, indexed by the user who sent it. The KRA then computes a message authentication code MAC_2 on A's identity id_A and the values *fileid* and *RAND*, i.e. $MAC_2 = f1_{K_{AC}}(idA, fileid, RAND)$, and sends it back to the user's card.

$$\text{A} \xleftarrow{MAC_2} \text{KRA}$$

This tells the card that the KRA, which is the only entity that can compute MAC_2, has successfully registered the received random value *RAND* for the file identified by *fileid*.

3. As soon as the card receives MAC_2 it recomputes it using the values *RAND* and *fileid* that it sent to the KRA, and checks it against the received MAC_2. If the check succeeds, the card uses the secret key K_{AM} and the random value *RAND* to generate the session key K_S as

$$K_S = f3(K_{AM}, RAND)$$

which is passed to the PC for the encryption process. The file is encrypted using K_S and a key recovery field *KRF* is attached to it. The *KRF* consists of the random value *RAND* (the KRA's identity should also be included if there are multiple KRAs), i.e.

$$KRF = \{RAND\}$$

Should an emergency access situation arise, the authorised entity will request the appropriate key K_A from the KRA. Having K_A and using the master key K_{AE} and the function $f3$, the authorised entity computes the corresponding user's key K_{AM}, i.e. $K_{AM} = f3(K_{AE}, K_A)$. Using K_{AM} and the value *RAND* attached to the file, the authorised entity can successfully recover the required key and the target data.

4.4.2 Properties and security analysis

With the proposed scheme, there is no need for interaction with the agent in everyday user access to the encrypted data, a property that simplifies the key recovery process for the user. More specifically, the user is able to recover the keys using his smart card, which can recompute the target key K_S using the value $RAND$ attached to the file.

The user will be required to contact his agent only if he has lost his card, in which case only the authorised entity AE can recover the user's keys. This property typically eliminates the requirement for an on-line agent for the recovery of keys (for the purposes of everyday user access to encrypted data) and avoids the related communications overhead. Moreover, the user does not have to back-up or archive the generated keys, thus avoiding the security hazards associated with secret key storage. Further properties of the proposed mechanism include:

1. The proposed mechanism is not vulnerable to rogue user attacks, as even if a rogue user deletes the generated KRF the KRA has the means to recover the requested key. Using just the index file and the identity of the file and the user, the KRA has all the needed values to compute the required key.

2. The KRA does not have to store or protect any of the user generated keys, thus avoiding certain problems that key escrow mechanisms face, e.g. preventing unauthorised access to the escrowed material. The only requirement, apart from the protection of the secret value K_M, is protection of the index file from unauthorised modification.

3. The mechanism benefits from the separation of the KRA from the authorised entity AE in that the KRA does not have access to users' generated session keys. The only entities that can recover the session keys are the users and the authorised entity AE. This allows the corporation to outsource the management of the KRM without endangering the confidentiality of the corporate data.

4. Dispersion of key material, a countermeasure that makes attacks on key recovery mechanisms more difficult, is properly enforced with the use of both K_M and K_{AE} for the computation of K_{AM} and, therefore, the generation of K_S. Even if K_M or K_{AE} is compromised an adversary cannot gain access to the users' keys. The attacker has to know both K_M and K_{AE} to be able to recover users' keys.

5. The random value *RAND* can be generated either on the card or on the user's PC and passed to the card. The security of the mechanism, however, does not rely on the randomness of this value, since it is only used to ensure freshness of the generated key. As a result, *RAND* can be generated on the PC to reduce the number of power consuming procedures that take place on the card.

6. As with the previous mechanism, to prevent a rogue user from manipulating the key after its generation, the whole mechanism should be implemented on the card, i.e. encryption must take place on the card. However, there are certain limitations associated with such an approach and alternative countermeasures are described in Chapter 5.

4.5 Comparison of the two schemes

As previously mentioned, the two schemes meet the two main requirements identified in Section 4.1 for key recovery mechanisms specifically deployed for archived data. Their main difference is the way that rogue user attacks are prevented. The first mechanism solves this problem by imposing a residual work factor on the agent recovering the keys when a rogue user attack occurs, a property that is likely to affect data availability given the time needed by the agent to recover the required keys. The second scheme requires the agent's participation in the generation of the session keys. Therefore, agent availability becomes a crucial requirement for the second mechanism, as otherwise it forces the user to leave his data unencrypted.

Although the first scheme does not necessitate the use of smart cards, as the second one

does, smart cards provide an effective means to prevent rogue user attacks. Therefore, in terms of cost of deployment of the mechanism the second scheme is no more expensive than the first one. Besides that, both mechanisms can be deployed using an existing multi-application smart card, which the employee might already use, and hence they may well not add significant costs to the existing infrastructure.

Both mechanisms offer the corporation the ability to outsource the service without endangering the privacy of the encrypted data. Although this constitutes one of the properties of the second mechanism, for the first mechanism use of multiple agents, with one of them being internal to the corporation, is essential if the corporation wishes to control the recovery process.

Finally, note that the second mechanism appears to be more secure against attacks targeting the compromise of the agent's key. The second mechanism requires an adversary to know two keys, i.e. the KRA's and the authorised entity's master keys, to get access to users' generated keys, while compromise of either of those keys does not endanger the privacy of the encryption keys. With the first mechanism, however, and if multiple agents are not used, compromise of the agent's private key would give the adversary access to all the users' generated keys (assuming that the adversary has access to the users' public keys).

4.6 Summary

In this chapter, the use of key recovery mechanisms for archived data has been investigated. More specifically, we identified the requirements that a KRM should fulfil when deployed to provide key recovery functionality for encrypted archived data. As most of the existing KRMs were designed for use with encrypted communicated data, using them with archived data causes problems, and thus there is a need for a mechanism designed specifically for archived data.

4.6 Summary

Two different KRMs were proposed that satisfy the identified requirements. The first involves a certain amount of work by the KRA to recover the keys if a rogue user deletes the generated KR information. The second mechanism defeats rogue user attacks by employing a secure user token and an on-line agent during key generation. Both mechanisms offer the user the ability to recover his keys without agent intervention. Neither requires the direct escrow of any user generated keys, avoiding the costs associated with the requirement for protection and administration of these keys.

Chapter 5

Implementation issues of the proposed KRM for archived data

Contents

A prototype implementation of the first of the two proposed KRMs for archived data is described in this chapter. The purpose of this demonstration is to examine the efficiency of the deployment of this mechanism using smart cards, and more specifically, using the Java Card technology.

5.1 Introduction to smart cards

A *smart card* or *integrated circuit card* (ICC) is a small device, about the size of a credit card, that contains memory, an input/output facility and, optionally, a microprocessor. Smart cards are capable of manipulating information, and hence they provide an attractive platform for implementing cryptographic functions. Tamper-resistance is another characteristic of ICCs which makes them a suitable medium for storing sensitive information. As a result, use of a smart card can be an important part of a security infrastructure, especially for operations that take place in untrusted environments. Smart cards are used extensively for a variety of applications including banking, mobile communications, and electronic purses.

Smart card applications consist of an *ICC-aware* application, or *client*, interacting with a *card-resident* component. The *ICC-aware* application is the software program that makes use of the functionality provided by the card, and which runs on some computing platform known as the *terminal*. A terminal can be a personal computer (PC), an automatic-teller machine, a personal digital assistant, or a network computer. The *card-resident* component comprises data and functions that reside on the smart card. The ICC communicates with the computing platform through the *interface device* (IFD), also known as *card acceptance device* (CAD) or simply a *smart card reader*.

The *ICC-aware* application interacts with the smart card by exchanging pairs of packets called *application protocol data units* (APDUs). The communication model is command-response based; the application sends a *CommandAPDU* to the smart card by handing it to the IFD's driver, which in turn forwards the APDU to the card. The smart card processes the CommandAPDU and returns the response in a *ResponseAPDU*. The set of Command–Response APDUs determines the smart card's functionality. Smart cards are reactive communicators, i.e. they never initiate communications.

A command APDU, as defined in ISO/IEC 7816-4 [36], consists of

- a mandatory header of 4 bytes (CLA, INS, P1, P2), and
- a conditional body of variable length.

Header				**Body**		
CLA	INS	P1	P2	[Lc]	[Data]	[Le]

Table 5.1 shows the contents of the command APDU (length is in bytes).

Table 5.1: Command APDU contents

Code	**Name**	**Length**	**Description**
CLA	Class	1	Class of instruction
INS	Instruction	1	Instruction code
P1	Parameter 1	1	Instruction parameter 1
P2	Parameter 2	1	Instruction parameter 2
Lc field	Length	1 or 3	Number of bytes present in the data field of the command
Data field	Data	Lc	String of bytes sent in the data field of the command
Le field	Length	1 or 3	Maximum number of bytes expected in the data field of the response to the command

A response APDU consists of

- a conditional body of variable length, and
- a mandatory trailer of 2 bytes (SW1 SW2).

Body	**Trailer**
Data field	SW1 SW2

Table 5.2 shows the contents of the response APDU (length is in bytes).

Table 5.2: Response APDU contents

Code	Name	Length	Description
Data field	Data	Variable	String of bytes received in the data field of the response. Its length depends on the Le value.
SW1	Status byte 1	1	Command processing status
SW2	Status byte 2	1	Command processing qualifier

5.2 Java Card

Among the various technologies that have been invented and used for smart cards, [30], is the *Java Card* [85], which is a smart card that can execute Java Card applications, namely *Java Card applets* or just *applets*. Java Card is a set of specifications issued by Sun Microsystems and based on the Java programming language. More specifically, as mentioned in [86], "Java Card technology combines a portion of the Java programming language with a runtime environment optimized for smart cards and related, small-memory embedded devices".

The software components that comprise the Java Card are depicted in Figure 5.1.

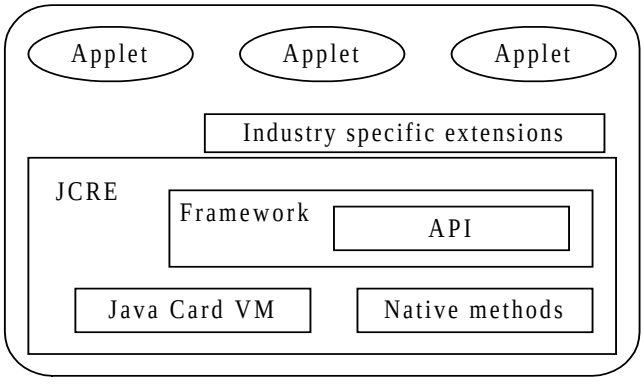


Figure 5.1: Java Card architecture

The heart of a Java Card is the *Java Card Runtime Environment* (JCRE) [86], which is the component that specifies how a Java Card manages its resources. More specifically, it defines how atomicity is to be achieved, how communication is to be handled,

how applications are to be managed (including how objects are to be shared between applets), and how security measures should be enforced. The JCRE includes native methods, that perform the input/output and memory allocation services of the card, and the framework, which is the set of classes that implement the *Application Programming Interface* (API). The API defines the calling conventions by which an applet accesses the JCRE and native methods.

A fundamental component of the JCRE is the *Java Card Virtual Machine* (JCVM) [87] which consists of two components; one that runs on-card and the other that runs off-card, i.e. on a PC or a workstation. The on-card JCVM executes bytecodes using the *bytecode interpreter*, manages classes and objects, enforces separation between applications (firewalls), and enables secure data sharing. The off-card JCVM contains a Java Card Converter tool, which converts the input class files into a *Java Card executable code format* (CAP file) during the development process. It also provides many of the verifications, preparations, optimizations, and resolutions that the Java virtual machine performs at class loading time. A schematic representation of this process and the relationship between the off-card and the on-card JCVM are depicted in Figure 5.2. This figure also briefly depicts the procedure followed for the preparation and download of an applet to the smart card.

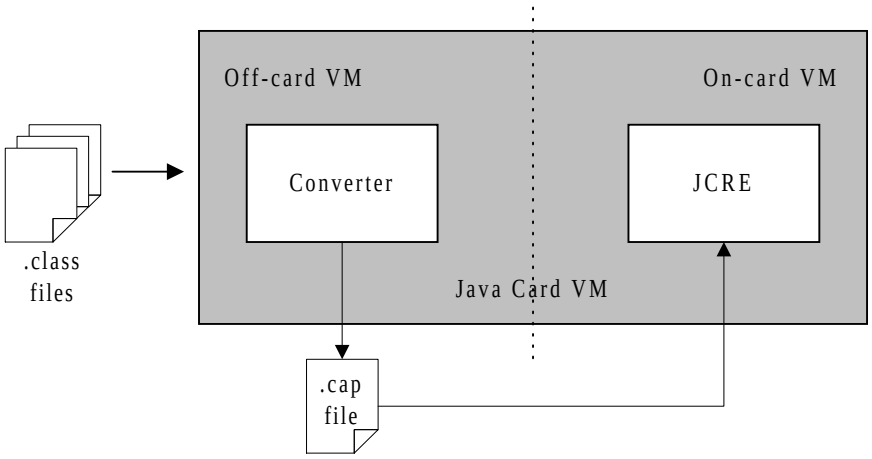


Figure 5.2: Java Card VM and converted classes downloading

Java Card technology offers many benefits including platform independence, multi-

application capability, and post-issuance of applications (i.e. loading applications onto a card after issue of the card). *Platform independence* ensures that Java applets will run on different vendors' cards (compliant with the Java Card technology). *Multi-application capability* allows multiple applications to run on a card and to be downloaded dynamically, while the inherent protection mechanism provides a secure environment for running multiple applications on a single card. Finally, *post-issuance* of applications provides card issuers with the ability to respond dynamically to their customers' needs.

There are certain limitations, however, associated with Java Cards compliant with the version 2.1 specification [85], including lack of support for dynamic class loading, multiple threads and multidimensional arrays, and non-mandatory garbage collection. These limitations, driven by the limited existing resources, in many cases make applet implementation a challenging process.

The Java Card technology has been used to implement the mechanism proposed in Section 4.3 to examine the efficiency of a smart card based version of the mechanism.

5.3 Implementation architecture

Implementation of the mechanism employing a user smart card, as mentioned in Section 4.3.2, offers benefits by securing the infrastructure against users' attempts to deploy their own, uncertified, key agreement keys, and to manipulate the generated session key. The high-level architecture of a prototype implementation, using a PC as the terminal, is depicted in Figure 5.3.

The Java Card applet contains all the functions accessible by the ICC-aware application, and provides an interface to the library functions that implement the KRM. User authentication is also part of the library functionality in the form of a secret 4-digit *personal identification number* (PIN). Once the user has been authenticated he can make

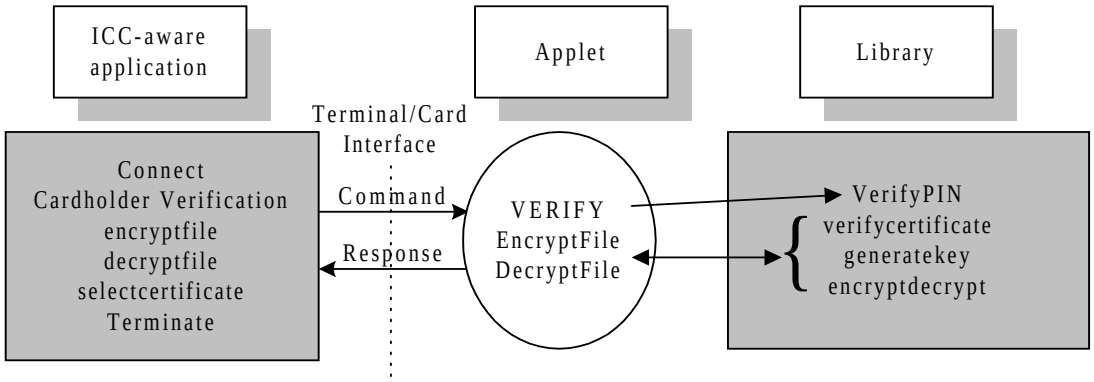


Figure 5.3: The implementation architecture

use of the functionality provided by the applet, unless the card has been removed from the IFD or a new session has started, in which case the user has to be authenticated again.

The proposed KRM was implemented using the Giesecke&Devrient Sm@rtCafé Java Card, partly compliant with the Java Card 2.1 API [85], while the interface technology used was the *OpenCard Framework* (OCF), developed by the OpenCard Consortium.

5.3.1 OCF overview

OCF is ICC middleware that sits between an ICC-aware application and the smart card reader. It is implemented in the Java programming language and seeks to provide functionality required by ICCs, IFDs, and PCs to allow portability and interoperability among compliant elements as provided by a variety of vendors.

The core architecture of the OpenCard Framework consists of two main components: the *CardTerminal* layer and the *CardService* layer. The *CardTerminal* layer provides access to physical card terminals and inserted smart cards. It also offers a mechanism to insert and remove card terminals through the use of the *CardTerminalRegistry*, which keeps track of the installed card readers [69]. The *CardService* layer represents a set of smart card services, each of which defines a particular API that can be used to access

a particular smart card function.

5.3.2 Applet functionality

The main applet functionality is provided by three library functions: `verifycertificate()`, `generatekey()`, and `encryptdecrypt()`. The `verifycertificate()` function verifies the certificate for the agent's public key. For this purpose a valid copy of the CA's public key has to be preloaded on the card. The `generatekey()` generates the session key K using the "data credentials" provided by the client, the user's private key stored on the card, and the agent's public key. The user's private key is accessible only by this applet, ensuring that the key cannot be accessed by a malicious applet.

The coding of the commands sent to the card is as follows (the command for PIN verification is not described). All the commands conform to the ISO/IEC 7816-4 format [36].

- **EncryptFile.** This command is used to request encryption of the data included in the "file data" field. The key will be generated using the data included in the optional "data credentials" field. If the command executes successfully the returned result is the encrypted data concatenated with the hex value '90 00'.

CLA	**INS**	**P1**	**P2**	**Lc**	**DATA**		**Le**
B0	50	GKCP	00	l	Data Credentials	File Data	00

GKCP – Generate Key Control Parameter. It can take the following values.

'01': The `generatekey()` function is invoked. The "data credentials" field is required.

'00': Encrypt with the existing key. There is no "data credentials" field. This mode is used when the data to be encrypted does not fit within the data field of a single APDU, and hence multiple APDUs have to be used.

l – the length of the DATA field.

- **DecryptFile.** This command is used to request decryption of the data included in the "file data". It has the same format as the 'EncryptFile' command except for the 'INS' field which, in this case, has the hex value of '60'. If the command executes successfully the returned result is the decrypted data concatenated with the hex value '90 00'.

- **GetCertificate.** This command is used to pass to the card the certificate for the agent's public key agreement key. The certificate received by the card will be verified and if this check succeeds the certified key will be used for the computation of K and the returned result will be the hex value '90 00'. The coding of the command is as follows:

CLA	**INS**	**P1**	**P2**	**Lc**	**DATA**	**Le**
B0	1A	00	00	l	Certificate	

The client part of the mechanism consists of three functions: `encryptfile()`, `decryptfile()` and `selectcertificate()`. The `selectcertificate()` selects an agent's public key and passes the appropriate certificate to the card. The result of the `encrypt()` function is the encrypted file concatenated with the KRF. The decryption is performed by extracting the KRF appended to the encrypted file, selecting the appropriate certificate, sending it to the card using the 'GetCertificate' command, and passing the "data credentials" to the card with the 'DecryptFile' command.

5.4 Implementation details

For the experimental implementation, dummy values were used for the user's private key x and the agent's public key g^y, and hence for the resulting key g^{xy}, as the desired cryptographic functionality was not available on the card. Moreover, the agent's public key was preloaded on the card. As a result, the key generation function included only the creation of a buffer which keeps the necessary key material, i.e. g^{xy} and "data

credentials". This buffer is used as input to a digest function and the result is passed to another function which converts the key to the format used by the encryption function.

The PKI functionality, including certificate upload and verification, was not part of the demonstration. Instead, the implementation focused on an examination of the encryption/decryption functionality efficiency and the handling of the generated key within the card. Figure 5.4 gives a flowchart of the mechanism's implementation.

The DES algorithm was used both in ECB and CBC modes for data encryption. The reason for using two different modes was to make a performance comparison, and to check how much more expensive the encryption process becomes when the CBC mode is used. Note that the code listed in Annex A makes use of DES in ECB mode.

Although, as mentioned earlier, the Sm@rtCafé Java Card that we used was only partly compliant with the Java Card 2.1 API, the lack of full compliance did not introduce any significant implementation problems. It was not possible, however, for the developed applet to be uploaded to a card fully compliant with the Java Card 2.1 specifications. Major modifications would be required to the existing code to make it fully compliant with the Java Card 2.1 API. Therefore, one of the main advantages of the Java Card technology, i.e. portability provided by platform independence, is lost. This problem, which arises from the fact that certain classes found in Sm@rtCafé Java Card are not supported by other cards, is of course easily solved by making Giesecke&Devrient Sm@rtCafé cards fully compliant with the corresponding specifications. As a result of this problem, however, cross-checking of the performance measurements with implementations on other cards was not possible.

5.5 Performance measurements and analysis

The main conclusion from the implementation of the proposed mechanism is that bulk data encryption on the card is still a prohibitively expensive operation. The perfor-

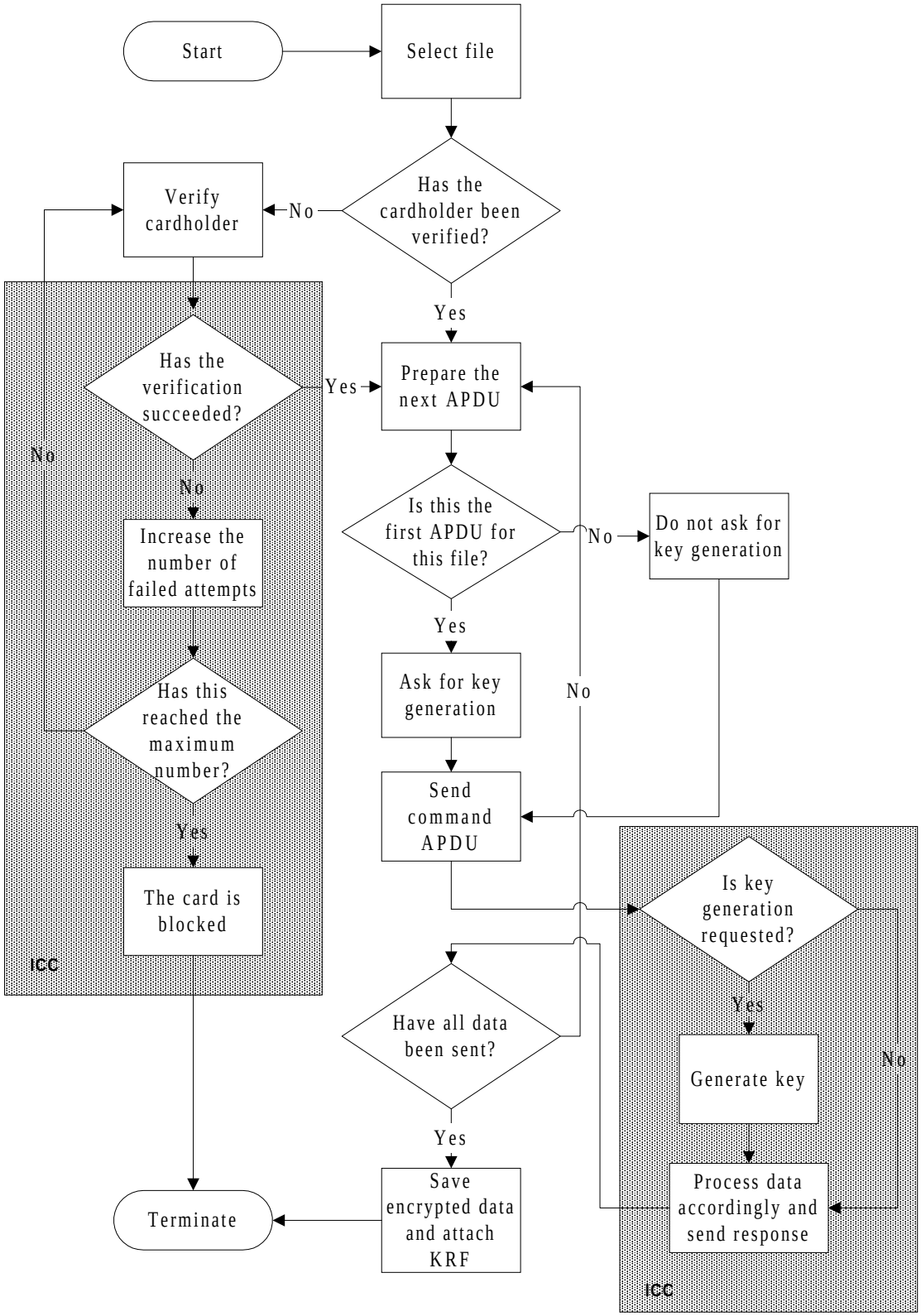


Figure 5.4: Implementation flowchart

mance measurements indicate that encryption on a card is a very slow process even for small amounts of data. Table 5.3 shows the results of ten consecutive encryptions of a data block of 248 bytes, which is the maximum amount of data that can be sent within a single APDU, given that the "data credentials" (6 bytes) used for key generation are included in the same APDU. Three different procedures have been measured to examine the efficiency of the implementation.

Table 5.3: Sm@rtCafé Java Card results using a block of 248 bytes of data

	Mean time (ms)	**Std. Deviation**
Encryption with key generation (DES-ECB)	2369	12.94
Encryption without key generation (DES-ECB)	2133	12.195
Encryption with key generation (DES-CBC)	2902	43.665
Encryption without key generation (DES-CBC)	2657	40.838
Transmission of 254 bytes	767	16.1

The "Encryption with key generation" figure indicates the time spent for the key generation together with the encryption of 248 bytes of data. Due to the lack of proper cryptographic functionality on the card, as mentioned in the previous section, the key generation stage includes only the computation of the digest of the key material and its preparation for use by the encryption function.

The "Encryption without key generation" row of the table involves the same operations as the previous figure except that no key generation takes place. The key used for encryption is the key computed when the first APDU for the specified file has been sent to the card. The difference between those two figures indicates that the actual key preparation lasts roughly 230ms.

The "Transmission of 254 bytes" figure gives an indication as to the delay introduced in the encryption/decryption process by the data transmission between the *ICC-aware*

application and the *card-resident* component. This figure includes the delay introduced by the use of OCF for the data transmission, and this gives an indication of OCF's performance as a middleware technology.

From the above figures, it is clear that with current smart card technology bulk data encryption is a very expensive operation. A better approach would be to use the card only for key management purposes, while data encryption can be done by the *ICC-aware* application. The key can be passed to the *ICC-aware* application using *secure messaging*, which is a method where cryptographic functions are used to protect and authenticate the data passed to and from the card [36]. Use of secure messaging can guarantee that the application that receives the key is the legitimate one and that this key originates from a legitimate card and has not been modified during transmission. Integrity checks on the *ICC-aware* application, [4], can also guarantee that the user will not tamper with the software and that the key is not manipulated in an unacceptable manner in an attempt to disable the key recovery functionality.

Alternatively, if the encryption is done by an *ICC-aware* application lacking integrity checks, then random checks on the encrypted data can ensure that the key recovery functionality is not bypassed by a rogue user.

5.6 Summary

In this chapter we considered the use of smart cards to implement one of the two key recovery mechanisms for archived data proposed in the previous chapter. The efficiency of the implementation was examined using a Java Card, in conjunction with the OpenCard Framework, which is an ICC middleware technology that sits between an ICC-aware application and the card reader. The outcome of this test implementation is that bulk data encryption on the card is still a very expensive procedure. With current technology it appears more sensible to use the card for key management, while

encryption takes place in the ICC-aware application.

The threats that arise from such an approach however, necessitate the introduction of extra countermeasures for the protection of the key recovery functionality against sophisticated rogue user attacks. Such countermeasures include secure messaging between the card and the ICC-aware application, and integrity checks on the ICC-aware application.

As the technology advances, however, it is possible that bulk data encryption on a smart card will become viable, especially if dedicated crypto co-processors are used for this purpose.

Part III

KEY RECOVERY FOR COMMUNICATED DATA

Chapter 6

Interoperability issues surrounding key recovery mechanisms

Contents

This chapter investigates the interoperability problems that arise from the use of dissimilar key recovery mechanisms in encrypted communications. The components of a key recovery mechanism that can cause interoperability problems are identified, as part of the development of a general model for key recovery mechanisms

6.1 Key recovery enabled communications

As previously mentioned, the issues surrounding key recovery for communicated data are somewhat different from those for archived data. Typically, the keys used for transient communications need not be retained, as once the communication session has finished such keys are no longer required and can be discarded. There is a need, however, for access to this key during its lifetime, i.e. during the communication session (or afterwards if the company logs any communications), especially if this communication takes place within certain environments, including companies and organisations. As described in Chapter 3, the company might want to have access to encrypted communications to check for malicious software or track leakage of sensitive information. Therefore, there are certain situations where the use of a KRM is required. The requirements that are of particular relevance to a KRM specifically deployed for communicated data are the following.

1. The KRM should give authorised entities the means to recover session keys used to encrypt the exchanged data, as well as keys used to encrypt communications-related data. This is vital for the company, as it may legitimately want to keep track of certain outgoing communications, to use this information for non-repudiation purposes in the case of a dispute, or even monitor incoming traffic for malicious software or for intrusion detection purposes. Thus, individual keys used by employees, or established during a communication session, must be recoverable. This includes keys generated by entities outside the company and used to protect messages sent to the company.

2. The KRM should provide the ability for on-line and real-time decryption of the intercepted communications. That is, if suspicious communications take place between an employee and an outsider (or between two employees) then the management should be in position to monitor them. This includes the ability to verify the validity of the KR information generated by the employee, or even decrypt

the communicated data at the time they cross the company's domain.

3. The KRM should be interoperable. That is, use of the mechanism should not prohibit the establishment of a secure communication session.

Assuming that the first two requirements are universal for KRMs, our investigation concentrates on the interoperability problems that are likely to arise from the use of dissimilar KRMs in encrypted communications. This has also been one of the primary concerns of the work carried out by the Key Recovery Alliance (see Section 1.1). Before looking in depth in the interoperability problem, it is worth examining how the two main categories of KRMs apply in encrypted communications.

6.2 Applicability of existing mechanisms

Most *key escrow* mechanisms designed for encrypted communications depend on a common key management infrastructure. While this might not be a problem in a corporate environment, i.e. for intranet communications, it is a major drawback when using the mechanism for communications that span company domains. The main reason is that there would be a need for the two communicating parties to deploy the same, or at least compatible, key establishment protocols. Otherwise, the establishment of secure communications is likely to be prevented.

Use of a *key encapsulation* mechanism, on the other hand, typically requires the addition of data fields to the communicated data, to carry the KR information. A receiving party in a different domain can simply modify its existing infrastructure so that additional fields are discarded without being interpreted. This allows secure communication to take place, and provides KR for the communicating party that wants it. However, such a configuration allows rogue users to mount cut-and-paste attacks (see Section 3.4.2), as the other party will not check the validity of the generated KR information.

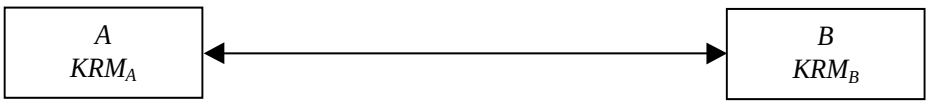
6.3 Interoperability

The word interoperability, as in [28], means the ability of entity A, which uses KRM KRM_A, to establish a KR-enabled cryptographic association with entity B, which uses KRM KRM_B. If A knows that B uses a different key recovery mechanism, then A may not know whether B can meet the requirements of A's KRM, and vice versa. This uncertain situation may force the parties to avoid key recovery, with associated increased risks especially when the entities operate in a business environment.

The interoperability issue arises from the variety of KRMs proposed by industry and academia, in conjunction with the lack of a standard for KRMs. Note, however, that even if KRMs were standardised there is no assurance that interoperability problems would be eliminated. Standards tend to provide a variety of sound mechanisms but not necessarily interoperable ones.

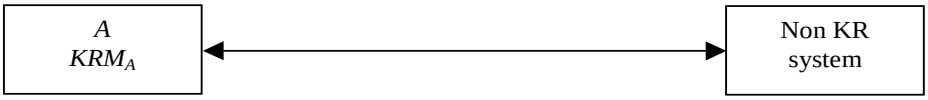
In the context of communications between two entities, A and B, we consider two possible scenarios where the use of a KRM might affect the establishment of a cryptographic association.

1. Entities A and B make use of KRMs KRM_A and KRM_B respectively, which might be identical, compatible or dissimilar mechanisms. In the case of identical or compatible mechanisms the two entities are not expected to face any problems. Problems, however, might arise if the two entities make use of dissimilar mechanisms. They are unlikely to be able to establish a secure communication while making use of their respective KRMs as this would typically demand each entity to fulfil the requirements of the peer's KRM.



2. Entity A uses KRM_A while B does not make use of KR. The issues that arise in

this case are whether B will be able to meet KRM_A's requirements, and whether A will be able to generate valid KR information. For the two entities to be able to communicate, assuming that A manages to generate valid KR information, B should at least be aware that A makes use of a KRM. This is important as B should not discard incoming traffic because of unrecognised additional fields (carrying the KR information and attached to the incoming data), which B cannot interpret. Another problem that is likely to arise is whether A's policy will permit the acceptance of incoming traffic that does not make use of KR. If A operates within a corporate environment, this requirement is likely to be crucial, as the company might want to run checks for malicious software on incoming data before they reach their destination.



Communicating entities who want KR functionality in their encrypted communication sessions and who wish to avoid the above problems, must use interoperable KRMs. Also, all deployed cryptographic mechanisms with embedded KR functionality should be compatible with cryptographic products that do not make use of KR. These requirements will ensure that neither of the above scenarios will prevent the establishment of a secure session.

6.4 Detailed description of the key recovery model

To identify those factors that can cause interoperability problems there is a need to give a more detailed description of the KR model described in Section 2.2. In particular, we need to consider the two most important processes that take place during a communication session, i.e. `key generation` and `KR information generation`.

6.4.1 The key generation process

Prior to an encrypted communication session, data encryption keys must be established. There are typically three options for this process.

1. Both entities contribute to the key generation process.
2. The key is generated by one of the communicating entities and either it is transmitted to the peer by conventional means, or the peer is able to generate the same key using some pre-agreed cryptographic values.
3. The session key is provided by a TTP, which in this case also serves the role of the KRA, and the key is transmitted to the communicating party(s) via a secure channel.

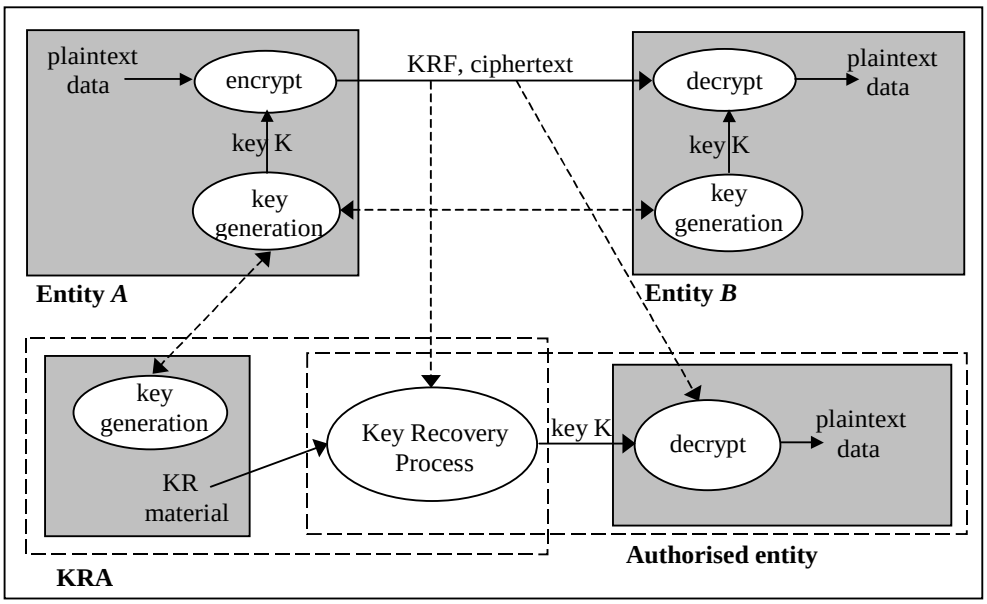


Figure 6.1: A typical KRM in relation to the key generation process

The above alternatives are represented in Figure 6.1 by the dashed arrows (which denote optional communications) between the `key generation` components of the three entities that might contribute to the establishment of the data encryption keys. If the KRA generates the session key on behalf of the user, *A*'s `key generation` component

will only serve the role of communicating with the KRA's key generation component. The key will be provided by the KRA to entity A and transferred to B by some secure means.

6.4.2 The key recovery information generation process

Every KRM, at some stage in its operation, will generate 'KR information' (which as described in Section 2.2 has the form of a *KRF* or *KR material*), as required for key recovery. The process of generating the KR information, in a form interpretable by the KRA's `Key Recovery Process`, is performed by the `KR information generation` component, depicted in Figure 6.2.

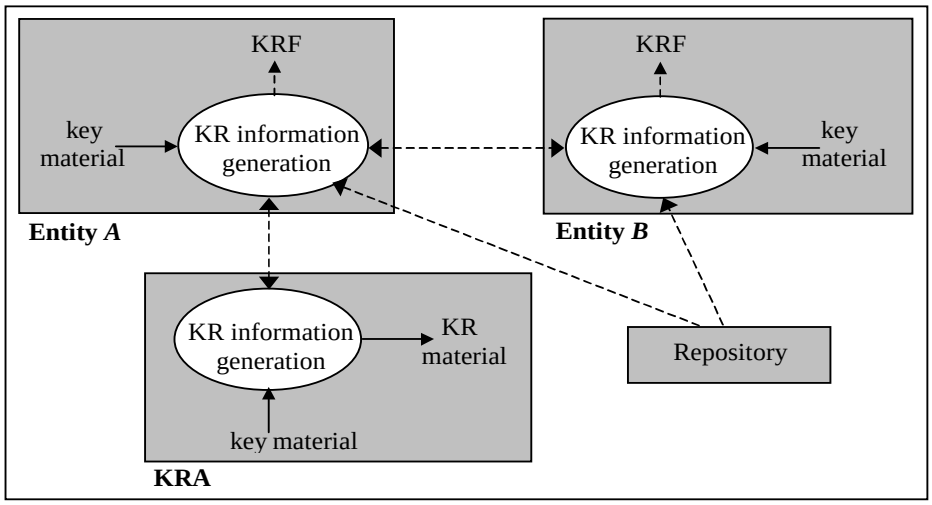


Figure 6.2: A typical KRM in relation to the KR information generation process

The dashed arrows to the *KRF* indicate that in some cases only one entity might be required to generate the *KRF*, which is not restricted to conveying key material – it can simply be data that uniquely identifies the session or the data encryption keys. The *KR material* generated by the KRA can include data encryption keys or key material that is not bound to a specific session (such as master keys, public key agreement keys or private decryption keys). The interaction between the entity's and the KRA's `KR information generation` might take place in every communication session, occasionally, or only once, e.g. during the initialisation of the KRM.

The *key material* in either A's or B's domain, which is input to the `KR information generation`, represents either the output of `key generation`, i.e. the generated data encryption key, or the input to this process (e.g. key components, master key, and/or random values). The first case includes KRMs that make use of a key generated by the underlying key establishment protocol, and as such can be independent of the key establishment method. The second case includes mechanisms that are tightly bound to the underlying key generation process, e.g. mechanisms where the KRA relies on a particular key establishment process which it can reproduce and thereby recover keys. Thus, depending on whether the *key material* is the output of, or the input to, the `key generation` process, KRMs can be divided into those that are bound to this process and those that are independent of it.

Unlike A's and B's `KR information generation`, which typically has to be executed in every communication session, the corresponding process in the KRA's domain might have to be executed only during the initialisation phase of the scheme, or only a limited number of times during the lifetime of the mechanism. In these latter cases the output of `KR information generation`, i.e. the *KR material*, is likely to be used for the recovery of multiple keys.

Finally, the *Repository* holds any additional information required by the *KR information generation* process, such as A's and B's public key certificates.

6.5 Factors that can affect interoperability

Many KRMs, as previously mentioned, require the use of a specific mechanism for the generation of the session keys, and as such they can be considered as part of the key establishment protocol. This restriction is one of the factors that can cause interoperability problems in the use of a KRM. A KRM with this property requires compatibility of the communicating parties' underlying key establishment protocols,

a requirement that is not always fulfilled. Key escrow mechanisms suffer more from this problem, as most of them demand the use of a specific key establishment protocol. By contrast, key encapsulation schemes appear to be more adaptable in this respect, since they simply wrap the generated data encryption key under the KRC's public encryption key (and hence potentially work with any key establishment protocol).

Flexibility of key encapsulation mechanisms with respect to the underlying key establishment protocols does not necessarily imply that interoperability problems do not arise. Interoperability very much depends on what additional requirements exist on their use. For example, interoperability problems arise if the recipient of encrypted data needs to validate KR information, or the receiver relies on the sender to generate KR information. These needs, which are likely to arise from policy requirements, will typically demand interaction between the two communicating parties, e.g. exchanging cryptographic values during the generation or verification of KR information. If either party's mechanism cannot cope with the demands of its peer, interoperability problems are likely to arise. This problem is not restricted to the use of key encapsulation schemes. In the case of key escrow mechanisms, a requirement for participation of both entities in the generation of KR information will have the same effect. We can therefore divide KRMs into two classes, depending on their communications requirements during the generation and verification of KR information.

1. KRMs where each entity generates KR information merely for its own needs without the peer's assistance. If neither party requires verification of the peer's KR information prior to decryption, interoperability issues become of minor importance and the two parties will be able to make use of their respective KRMs.

2. KRMs that require interaction between the two entities, which might be needed in the following cases.

    - Exchange of cryptographic material is required for KR information generation.

- The sender generates KR information both for his own and his peer's needs. This is particularly relevant to single-message communications, such as email or file transfer.
- Either party wishes, e.g. for policy reasons, to verify the KR information generated by the peer.

In situations like these, interoperability is an issue that has to be taken into account. Otherwise, it is likely to result in a failure to establish secure communications.

In summary, the two factors that are likely to affect the interoperability of KRMs in encrypted communication sessions are:

1. the KRMs' dependence on the underlying key establishment protocol, and
2. the interaction requirements between the communicating parties for the generation and/or verification of KR information.

6.6 Interoperable mechanisms

Based on the above analysis, a mechanism that is neither dependent on the underlying key establishment protocol, nor needs any interaction with the peer for generation or verification of KR information, will always be interoperable with a KRM with the same requirements. The two mechanisms can work independently, and the two parties can make use of them regardless of the underlying key establishment protocol. A key encapsulation scheme where each entity encrypts the key generated by the underlying key establishment protocol using the agent's public key has this property. We assume, of course, that neither entity requires its peer to verify the key recovery information, and that the transmission of the KRF will not cause any problems to the recipient.

A mechanism with these properties can also inter-operate with one that is dependent on the underlying key establishment protocol, as long as the mechanism does not require peer interaction for KR information generation or verification. As an example, consider the scenario where entity A makes use of a key escrow mechanism, where the session key is generated using an escrowed master key and some additional information made available to A's agent, and the remote entity B uses a key encapsulation mechanism, while the generated key is transferred to B via a secure channel. As B has no requirements for the key establishment process, and there is no interaction required to generate the KR information, interoperability problems are not likely to arise, assuming that verification of KR information by B is not required.

Yet, if B used the same key escrow mechanism there would be a conflict and the two mechanisms could not inter-operate as the receiver would have no means to escrow the generated key (which in this case is provided by the sender). The problem here is that both KRMs demand the use of specific key establishment protocols, and therefore the requirement for compatibility of the key establishment protocols becomes crucial for the interoperability of the KRMs (a requirement that in this case is not fulfilled). Thus, there are cases where even the same mechanism cannot be used simultaneously by two parties. Hence, if both parties mandate the use of such a KRM then it will potentially act as a barrier to secure communications.

Interoperability problems are likely to arise in the following cases (we assume that the communicating parties can deal with all possible underlying key establishment protocols):

1. Both KRMs demand the use of specific key establishment mechanisms regardless of requirements for interaction. In this case, the interoperability of the KRMs depends on the compatibility of the respective underlying key establishment protocols.

2. At least one KRM demands peer participation in the generation and/or verifi-

cation of KR information. For instance, if policy restrictions demand that the KR information for the receiver's needs should be generated by the sender, it is apparent that the sender must be able to cope with the receiver's mechanism. Otherwise, it is likely that establishment of secure communications will fail.

Given these scenarios, the chances of interoperability problems arising are considerable and a solution has to be found.

6.7 A scheme proposed by the Key Recovery Alliance

To overcome the interoperability problems, the Key Recovery Alliance has proposed a mechanism [28], which, when adopted, enables the establishment of secure communications between dissimilar KRMs. The mechanism introduces the *common key recovery block* (CKRB), that "serves as a container for a single KR mechanism–specific KRF". According to [28] the CKRB achieves two main objectives; it "provides a means to identify the KRM used to construct the KRF", and "provides a range of validation techniques, including those that allow validation of the KR information in generic, KR mechanism–independent ways". (Note that, in line with [28], throughout we simply refer to KRB instead of CKRB)

Brief descriptions of the KRB and the KRB validation techniques are given below. For full details and explanations of the mechanism, see [28]. The KRB consists of the following fields:

- **KRB Version Number**: Set to 1 for the version of the common KRB format specified in [28].
- **KRB Length**: The number of words in the entire KRB.
- **Object Identifier (OID) for KRF**: The OID for the KR mechanism used to generate the KRF, as registered with a central authority.

- **Reserved**: A 16-bit field reserved for future use.
- **KRF Length**: Number of words in the KRF.
- **Key Recovery Field**: The proprietary KRF whose format and contents are indicated by the OID.
- **Validation Field Type**: Identifies one of the following eight techniques used to compute the Validation Field.
    1. **NONE (Type 0)**: No *Validation Field Value* (VFV) is calculated; KRF validation is unnecessary at the decrypting side.
    2. **SEMANTIC (Type 1)**: No VFV is calculated; the KRF should be validated semantically using the mechanism-specific algorithm.
    3. **PROTOCOL (Type 2)**: No VFV is calculated; the KRB need not be checked for validity since the carrier protocol provides integrity protection for the KRB.
    4. **CONF-HMAC-SHA–1-96 (Type 3)**: The VFV is a MAC of the KRB using HMAC [47] and SHA-1 [52] and the confidentiality key associated with the KRF.
    5. **CONF-HMAC-MD5-96 (Type 4)**: The VFV is a MAC of the KRB using HMAC [47] and MD5 [51] and the confidentiality key associated with the KRF.
    6. **INTEG-HMAC-SHA–1-96 (Type 5)**: The VFV is a MAC of the KRB using HMAC and SHA-1 and the integrity key associated with the KRF.
    7. **INTEG-HMAC-MD5-96 (Type 6)**: The VFV is a MAC of the KRB using HMAC and MD5 and the integrity key associated with the KRF.
    8. **SIGNATURE–PKCS7 (Type 7)**: The VFV is a PKCS7 signature block that carries a digital signature, which is calculated on the hash of the KRB.
- **Validation Field Length**: Number of words in the VFV.

- **Validation Field Value**: It is calculated over the entire KRB. If some of the KRB fields are unknown at the time the Validation Field Value is calculated, then:
  - the KRB Length field is set to zero,
  - the Validation Field Length is set to zero, and
  - the Validation Field Value is omitted (it has a length of zero).

  The length of the Validation Field Value is variable, but it is padded with zeros to be an integral number of 32-bit words.

6.8 Remarks on Key Recovery Alliance's CKRB format

As mentioned above, the mechanism proposed by the Key Recovery Alliance is intended to promote interoperability between dissimilar mechanisms. However, two problems can be identified with this mechanism. The first relates to difficulties arising from the generation of proprietary KRFs, while the other concerns the fact that, in many cases, the mechanism fails to provide interoperability.

6.8.1 KRF generation

The paper describing the Key Recovery Alliance mechanism, [28], makes the implicit assumption that a KRF has already been generated and therefore is always available for KRB generation. However, this is true only in a limited number of combinations of the various KR schemes. In the case of non-interoperable mechanisms, the sender might not be able to generate a KRF because the receiver does not fulfill the requirements of the sender's KRM. A simple example of this problem is the case where KRF generation requires the encryption of the secret session key (data encryption key) using the receiver's public key, which is escrowed with the receiver's escrow agent. If the receiver does not have a public key meeting the requirements of the sender's KRM, e.g.

because he is not using a key escrow mechanism, then the sender cannot generate a KRF.

6.8.2 Interoperability issues

Among the Validation Field Types proposed in [28], there are five (types 2–6) which provide validation of the KRB. According to [28], using the generic validation mechanisms supported by the KRB, entity B would be able to validate the proprietary KRF sent by A and vice versa, "even though B did not understand how to parse the KRF". However, the placement of the KRF within the KRB does not enable this validation. The receiving entity still has to parse the KRF to check its validity. Moreover, the integrity and/or authenticity of the KRB, and therefore of the content of the KRB, i.e. the KRF, does not guarantee the KRF's validity.

As a consequence, the validation techniques proposed are vulnerable to a single rogue user scenario. Consider the case where two parties communicate using dissimilar (non-interoperable) KRMs. Sender A generates a KRF for receiver B, who is not able to verify the proprietary KRF using the method required by A's KRM because he is using a dissimilar KRM. Assuming further that A is a rogue user, then the following scenario might arise.

Rogue user A generates an invalid KRF. However, the VFV is generated using the valid session key (validation field types 3-6), which the receiver knows in advance. A genuine receiver B will validate correctly the KRB, as this was correctly generated by A. However, the validation technique and therefore the validation of the KR information will fail because the KRF is not genuine, and B has no means to verify it (we assume that the two mechanisms are not compatible and therefore B does not know the semantic details of A's KRM). Thus B will not be compliant with his policy, which requires validation of the KRF prior to decryption of the encrypted data.

Moreover, the agent with which A is associated will not be able to recover the key. However, the KRB was accepted as valid by B. This problem arises because the KRB validation mechanism only enables the recipient to verify the integrity and/or origin of the KRF. The KRB scheme is not a "mechanism for verifying the validation of the enclosed KRF" [28]. There is no way to recover the key from the KRB using only the information provided by the KRB itself (excluding the KRF because this can be manipulated only by the users that deploy the same KRM).

In the case of Validation Field Type 7, the same problem holds, as the VFV is a digital signature on the KRB which can carry a nonvalid KRF. The only Validation Field types that do not suffer from this problem are types 0 and 1 which demand "no validation" and "mechanism specific validation" respectively.

Therefore, the solution proposed here does not achieve one of the two major objectives mentioned above, which is to provide "the ability to validate the KR information in a way that does not require knowledge of the exact semantic properties of the KRF". The solution fails to achieve its objective in situations where it is most desirable, i.e. in environments where there is a lack of trust.

The KRMs where the KRF is equal to the key exchange block do not suffer from the above problem, because in such a case the KRF has to be processed to obtain the decryption key. However, these mechanisms still face the problem where the sender might not be able to generate a KRF and, if one is generated, the receiving end might not be able to parse it and get the session key. The KRB proposed does not offer an alternative to this situation, since the receiver cannot obtain the session key using interoperable components.

A different approach to the solution of the interoperability problem is described in the next chapter of this thesis. This enables some of the difficulties described above to be avoided.

6.9 Summary

The introduction of a large number of KRMs and their use in encrypted communications is likely to result in interoperability problems between key recovery enabled cryptographic products. In this chapter, a detailed key recovery model has been described and the factors that can cause interoperability problems have been identified.

A study of the model proposed by the Key Recovery Alliance to solve the interoperability problem has revealed that this scheme does not achieve its major objective, which is to provide interoperability between dissimilar mechanisms. This motivates the study of alternative schemes, the subject of the next chapter in this thesis.

Chapter 7

A protocol for negotiating key recovery mechanisms

Contents

In this chapter, a protocol is proposed which allows two communicating entities to negotiate the key recovery mechanism(s) to be used, with the ultimate goal of providing the parties the means to agree either on a mutually acceptable KRM or on different, yet interoperable, mechanisms of their choice.

7.1 High-level description of the KRM negotiation protocol

We now describe a protocol designed to enable two communicating parties to negotiate the KRM(s) to be used in an encrypted communication session. Its main objective is to deal with situations where the two parties might otherwise wish to make use of different, non-interoperable, KRMs.

A similar protocol, specifically designed to allow the negotiation of key recovery mechanisms using the Internet Security Association and Key Management Protocol (ISAKMP), is described in [3]. This protocol, however, uses the CKRB (described in the previous chapter) for the transmission of key recovery information, with which problems have been identified (see Section 6.8). A more general protocol is described here that considers the differing requirements of various mechanisms and potential additional requirements for the exchange of cryptographic certificates. Moreover, the proposed protocol can be used to provide key recovery functionality at the application layer, in contrast to the Key Recovery Alliance scheme which targets the IP layer.

Note that in the protocol description we refer to the two parties as 'Client' and 'Server', as opposed to our previous labels of A and B; this is so as to follow the client-server model terminology as closely as possible. The approach is similar to the negotiation of cryptographic parameters and establishment of a secure communication session in TLS [19].

7.1.1 The proposed scheme

The protocol consists of the following steps (messages in Small Caps are optional; more detailed descriptions of the exchanged messages are given in the next section).

- The client initiates the protocol by sending the *ClientHello* message to the server.

$$\text{Client} \xrightarrow{\texttt{ClientHello}} \text{Server}$$

- The server responds with his *Hello* message, the *ServerHello*.

$$\text{Client} \xleftarrow{\texttt{ServerHello}} \text{Server}$$

With the *ClientHello* and *ServerHello*, the two entities exchange the parameters necessary for KRM negotiation. After the *ServerHello*, the server sends the *Certificate* message to the client (if required by the selected KRM(s)) containing the appropriate certificates

$$\text{Client} \xleftarrow{\textsc{Certificate}} \text{Server}$$

and requests the corresponding client's certificates with the *CertificateRequest* message.

$$\text{Client} \xleftarrow{\textsc{CertificateRequest}} \text{Server}$$

Following that, the server sends the *ServerHelloDone* message, which indicates that the server has completed his *Hello* messages.

$$\text{Client} \xleftarrow{\texttt{ServerHelloDone}} \text{Server}$$

- The client responds with the optional *Certificate* message, which contains the certificates specified in the *CertificateRequest*,

$$\text{Client} \xrightarrow{\textsc{Certificate}} \text{Server}$$

the optional *KRParameters* message which contains any additional information required by the selected KRMs,

$$\text{Client} \xrightarrow{\textsc{KRParameters}} \text{Server}$$

and finally the *Finished* message, which indicates that the client has completed the negotiation of the mechanisms.

$$\text{Client} \xrightarrow{\texttt{Finished}} \text{Server}$$

- The server verifies the received *Finished* message, and if the check succeeds it responds with a similar *Finished* message.

$$\text{Client} \xleftarrow{\texttt{Finished}} \text{Server}$$

On receipt of this message the client verifies it and, if the verification succeeds, the negotiation protocol terminates successfully.

7.1.2 Exchanged messages

In the following sections the exchanged messages are described in detail.

7.1.2.1 Client Hello

The client, as previously mentioned, initiates the protocol by sending the first message of the negotiation protocol (*ClientHello*). The *ClientHello* contains a list of KRMs (from a complete list of mechanisms the protocol supports) that the client is able and willing to use, in decreasing order of preference. A default mechanism that all parties are assumed to be able to cope with, and which can serve as a worst case solution, can be included in the list.

With the *ClientHello*, the client must also inform the server whether he wants to resume a previous session, by including the appropriate identifier in the corresponding field. If this field is left empty, a new session id should be assigned by the server.

7.1.2.2 Server Hello

If the client does not request the resumption of a previous session, or if the server wants to initiate a new one, the server must assign a new session id, which will be included in the corresponding field and sent with the *ServerHello* message. Otherwise, the server will respond with the session id included in the *ClientHello* and proceed with the *Finish* message.

If the server initiates a new session, he sends the selection of the KRM that he wants the client to use, and the KRM that the server will use. The two mechanisms, if not identical, have to be interoperable. For this purpose, a list of all possible matches of interoperable mechanisms has to be kept by both communicating entities. If an acceptable match is not found in the list, the server can either terminate the negotiation protocol unsuccessfully, or choose the default mechanism if this was included in the list received from the client. Otherwise the server can drop the session.

If the selection of the mechanism for both entities is a KRM that can itself handle the exchange of certificates and related KR parameters, the two parties can terminate the negotiation protocol and leave this KRM to take charge. To achieve this, the server will send a *Finish* message (after the *ServerHello*) to indicate that control is now to be passed to the negotiated KRM(s).

Finally, within the *ServerHello*, the server also includes the *KRParameters* field, which carries any additional information that the client has to possess to be able to deal with the server's KRM.

7.1.2.3 Certificate and KRM related information exchange

Depending on the selection of the KRM, and if the server has not sent a *Finish* message, the server proceeds with the *Certificate* message. This message is optional and

contains the required certificates (for the chosen mechanisms) for the generation and/or verification of KR information. Following that, and depending on the requirements of the chosen KRM(s), the server can also send a request for the corresponding client's certificates using the *CertificateRequest*. The purpose of the *CertificateRequest* is to give the client a list of specific types of certificates needed by the server, and a list of certification authorities trusted by the server. After the *CertificateRequest*, the server sends the *ServerHelloDone* message, which indicates that the server has completed his *Hello* messages.

On receipt of the *ServerHelloDone*, if the client has received a *CertificateRequest* he responds with his *Certificate* message, which contains the requested certificates, assuming that he is in possession of the appropriate ones. Further, the client sends in the optional *KRParameters* message any additional information required by the selected for the client KRM. Note that the corresponding *KRParameters* for the server's KRM is sent as part of the *ServerHello* message.

7.1.2.4 Finish messages

If the client is satisfied with the current selection of mechanisms he sends the *Finish* message, which indicates that the client is willing to proceed with the current selection of mechanisms. Subsequently, the client waits for the corresponding server's *Finish* message, whose receipt indicates successful execution of the protocol.

7.2 Avoiding modification of the *Hello* messages by an adversary

After the execution of the above protocol, the two entities are not sure whether any of the exchanged messages have been altered during transmission by an adversary, as the specified protocol includes no proper integrity checks. Moreover, neither of the

communicating parties authenticates the other. Assuming, however, that the KRMs that can be negotiated are sound, the protocol does not introduce any vulnerabilities to the secrecy of the session key. The only attack that an adversary can mount against the protocol is to alter the *Hello* messages exchanged between the two entities in an attempt to downgrade the negotiated KRM(s) to one(s) that the attacker regards as weaker. Such an attack will only force the two entities to make use of less favourable mechanisms.

To avoid such problems, we propose enhancing the previously proposed protocol. These enhancements provide the following security services.

- Integrity of the exchanged messages.
- Assurance that the *Hello* messages exchanged are not a replay from a previous session.

Additionally, the mechanism provides mutual authentication of the communicating parties. Note, however, that mutual authentication is not a requirement for the negotiation protocol. It is a property derived from the use of digital signatures. The modifications proposed are as follows.

- The client generates a random value $randC$, which he sends to the server with the *ClientHello* message.
- The server generates a random value $randS$, which he sends to the client with the *ServerHello* message.
- The client's *Finish* message becomes

$$S_C(ClientHello \parallel ServerHello \parallel randC \parallel randS)$$

and the server's *Finish* message becomes

$$S_S(ServerHello \parallel ClientHello \parallel randS \parallel randC)$$

where $S_U(M)$ is U's signature on data M and "$\parallel$" denotes concatenation.

The rest of the messages remain as previously defined. On receipt of the respective *Finish* messages the two entities check the signatures and if either of the two verification checks fails the protocol terminates unsuccessfully (this indicates that at least one of the *ClientHello*, *ServerHello* might have been altered during transmission). The modified protocol deals with the threat of modification of the exchanged *Hello* messages by an adversary. The generated random values prevent against replay attacks, i.e. where an adversary uses old exchanged messages to subvert the protocol. The cost of this countermeasure, however, is the introduction of signatures, which have to be supported by an appropriate public key infrastructure.

In a practical implementation of the protocol the two variants can actually co-exist and the two parties will be able to select the mode to be used. More specifically, an extra field in the *Hello* messages can be used to indicate whether the two entities possess the appropriate certificates and therefore are willing to use the second variant of the protocol.

7.3 Properties and discussion

The proposed protocol offers the communicating parties a means of negotiating the KRMs to be used for their encrypted communications. Given that this negotiation will affect the selection of the key establishment protocol, execution of the proposed protocol must take place before the establishment of any session keys. It might also be the case that the two entities are obliged to use a specific key establishment protocol. This will simply restrict the number of mechanisms that the two entities will be able to negotiate.

The negotiating entities will be able to choose different KRMs as long as there are no conflicts between the underlying key establishment mechanisms. Therefore, the choice of the key recovery mechanism(s) will only be affected by the compatibility of the underlying key establishment protocols. If these are compatible, the two parties will be able to use the negotiated KRMs, overcoming efficiently any interoperability problems that the two parties would have otherwise faced.

Finally, note that in order to achieve the degree of agreement needed, the KRM negotiation process and the KRMs to be negotiated need to be subject of a standardisation process of some type (e.g. via the IETF). This standard will need to include agreed identifiers for a large set of KRMs.

7.4 Summary

A new KRM negotiation protocol has been described which follows a different approach to the scheme proposed by the Key Recovery Alliance. The new protocol gives the communicating parties the ability to negotiate the KRM(s) to be used. The protocol allows the communicating parties to use different, yet interoperable, KRMs.

Part IV

AVOIDING UNFAIR KEY RECOVERY

Chapter 8

A fair certification protocol

Contents

In this chapter a 'fair' key generation and certification protocol for Diffie-Hellman keys is proposed, which is intended for use in cases where neither user nor certification authority are trusted to choose the user's key on their own. This protocol also ensures that key agreement mechanism 1 in ISO/IEC 11770-3 [38] provides 'fair key agreement' [63] and prevents unfair key recovery by either of the communicating parties.

8.1 Fair key agreement

In the multi-part international standard ISO/IEC 11770, a number of key establishment techniques are described. *Key establishment*, as defined in ISO/IEC 11770-3 [38], is "the process of making available a shared secret key to one or more entities", and it can be subdivided into key transport and key agreement. *Key agreement* is "the process of establishing a shared secret key between entities in such a way that neither of them can predetermine the value of that key" [38]. This definition implies that the shared secret is derived as a function of the information contributed by, or associated with, all the communicating parties such that none of them can predetermine the value of the key [59].

Key agreement mechanisms are used in environments where the communicating parties, who may not trust one another, wish to be sure that a session key used to protect communications between them is derived so that neither of the communicating parties can predetermine some number of bits in the session key or all of its value. As briefly discussed in [63], the mechanisms described in ISO/IEC 11770-3 [38], clause 6, and 11770-2 [37], clauses 5.5 and 5.6, do not provide 'fair key agreement', as they do not prevent one of the communicating entities from choosing part of the shared secret key.

The basic idea behind most key agreement protocols, and certainly all the protocols described in ISO/IEC 11770, is that both parties provide a 'key component', and the two components are combined in some way to give the key. The method used to combine the components is typically a one-way function. As mentioned in ISO/IEC 11770-3 [38], certain checks, depending on the particular key agreement mechanism and/or cryptographic functions used, should also be enforced to prevent the use of weak values (key components).

However, in the mechanisms in the above standards, neither the use of a one-way 'combiner' function nor these checks can prevent one entity gaining an advantage over

the other. Suppose, as is the case with most such schemes, one entity (A say) sends its component to the other entity (B say) before B sends its component back to A. There is then nothing to stop B from working on the key component received from A prior to choosing its own component, allowing B to choose part of the shared secret key. Specifically, "if B is prepared to perform approximately 2^s computations of the one-way function used to combine key components prior to sending a response to A, then B will be able to choose s bits of the shared secret key" [63]. The computation of the combinations has to be performed within the limited space of time that B has prior to sending a response back to A. Yet, if a fast hash function, such as SHA-1, is used to combine components, B may be able to test as many as $10^4 - 10^5$ key components in a second or so, allowing him to choose as many as 16 bits of the shared secret key.

Before proceeding, we consider why allowing one of the two entities to choose a few bits of the shared key might be a threat. Suppose entity B has agreed to allow party C to have access to his keys for key recovery purposes, but B does not wish to let A know. Moreover, although this could be achieved by having B pass a copy of every key to C, this is potentially costly in communications and storage, and B and C wish to find an alternative. Suppose that the keys agreed between A and B are 64 bits long. Then, using essentially the same technique as described in [63], B may be in a position to choose every key shared with A such that the last 16 bits are a fixed function (known only by B and C) of the first 48 bits. When C wishes to recover a key, C needs only test at most 2^{48} possibilities for the key (e.g. using a known plaintext/ciphertext pair).

8.2 The *commitments* solution

To avoid the above problem, the use of *commitments* is proposed in [63]. The notion of *commitments* in cryptographic protocols is a well-established one, particularly in the context of zero-knowledge protocols (see, for example, [59]). By a commitment here we mean the disclosure of a value by an entity which binds that entity to a related value,

without revealing that related value (in some contexts the disclosed value is referred to as a *witness*). The main idea behind using commitments to make key agreement schemes fair is to ensure that both parties choose their own key component before seeing the other party's component. This is achieved by requiring one entity, say B, to hash its key component using a one-way hash function, and to send the resulting hash-code as the commitment to the other entity, say A, before A sends its own key component to B. Assuming that the key component contains sufficiently many bits, A cannot calculate B's key component before choosing its own. Therefore, A has to generate and send its own key component without seeing the exact value of B's component. After that, B can pass its key component back to A, who hashes the value and checks whether the newly computed hash-value matches the previously provided commitment.

Most of the standardised key agreement mechanisms described in ISO/IEC 11770-2 [37] and 11770-3 [38] can easily be adapted, using *commitments*, to provide 'fair key agreement'. However, of the seven key agreement methods specified in ISO/IEC 11770-3, the use of the commitment-based solution only applies to four of them; it does not work for the mechanisms involving use of pre-established key agreement key pairs (key agreement mechanisms 1–3 in ISO/IEC 11770-3). The main reason is that if one of the communicating parties has a public key agreement key certified by a *certification authority* (CA), the other party can work on it for a long period of time before choosing its key component.

In this chapter, we consider key agreement mechanism 1 in ISO/IEC 11770-3 and provide a solution to the 'fair key agreement' problem for this scheme. The proposed solution could more generally serve as a 'fair certification protocol' for Diffie-Hellman keys where the user does not choose his private key, but neither is the private key released to the CA. The mechanism could be deployed in environments where neither the CA nor the user trust each other to choose the user's key.

8.3 Key agreement using public key cryptography

Before describing the new method for 'fair key agreement', we consider the existing mechanisms in more detail.

In the key agreement mechanisms described in ISO/IEC 11770-3 [38], both communicating parties contribute to the shared secret key, which is computed as a one-way function of the key components that the parties have chosen. The requirements for use of the mechanisms are given in ISO/IEC 11770-3 clauses 5 and 6. Most importantly, entities A and B using one of these protocols must have agreed on a function $F : H \times G \rightarrow G$, with the following properties.

1. F satisfies the commutativity condition $F(h_A, F(h_B, g)) = F(h_B, F(h_A, g))$.
2. It is computationally intractable to find $F(h_1, F(h_2, g))$ from $F(h_1, g)$, $F(h_2, g)$ and g. This implies that $F(\cdot, g)$ is a one-way function.

Also A and B must share an element g in G, which may be publicly known, and A and B must be able to efficiently compute values $F(h, g)$, and must also be able to generate random elements in H. One 'obvious' candidate for F is to choose a large prime p, put $G = Z_p$, put $H = Z_p^*$, let g be an element of large prime multiplicative order modulo p, and define

$$F(h, g) = g^h \bmod p.$$

The seven key agreement mechanisms specified in ISO/IEC 11770-3 [38] can be divided into three classes.

- In one scheme (mechanism 1) the shared secret key is generated as a function of the two parties' pre-established key agreement keys. According to mechanism 1

two entities A and B non-interactively establish a shared secret key using their key agreement key pairs. Use of the mechanism requires each entity X to have a private key agreement key h_X in H and a public key agreement key $p_X = F(h_X, g)$, and both entities to have an authenticated copy of each other's public key agreement key [38, clause 6.1]. The mechanism involves the following steps:

1. A computes, using its own private key agreement key h_A and B's public key agreement key p_B, the shared secret key as $K_{AB} = F(h_A, p_B)$.
2. B computes, using its own private key agreement key h_B and A's public key agreement key p_A, the shared secret key as $K_{AB} = F(h_B, p_A)$.

- In two schemes (mechanisms 2 and 3) the shared secret key is generated as a function of one party's pre-established key agreement key, and the other party's dynamically generated component.
- In the other four schemes (mechanisms 4–7) the shared secret key is computed as function of two dynamically generated components, one for each party.

As already discussed, the 'commitments' solution only applies to the third class of mechanisms. The fairness problem arises in the first class of mechanisms because one party may select his/her key agreement key pair so as to choose part of the key shared with another specified entity. Suppose entity A generates and publishes his public key agreement key. When B chooses his key agreement key he can ensure that the key established between A and B has certain properties. Of course, B has no control over keys established between himself and other entities, so the problem is restricted in scope. Nevertheless, B potentially has a long time in which to work on A's key before choosing his own, and in this respect the problem is worse than for mechanisms 4–7. We now propose a solution to this problem for the first class of mechanisms, based on the idea of preventing a user choosing his/her key pair, whilst preserving the secrecy of the user's private key.

Finally note that we do not have a solution to the problem for the second class of

mechanisms. This case appears particularly intractable, and if the lack of 'fairness' is a major problem then a mechanism from one of the other two classes should be used.

8.4 A fair key generation and certification protocol

We present a protocol between an entity and a CA which provides the entity with a private key and a certified public key, where the user cannot choose his/her private key but also no other entity (including the CA) knows the private key. Use of the proposed certification and key derivation mechanism requires the user and CA to share a secure channel and have agreed a modulus (a large prime number p), an element g of large prime multiplicative order modulo p, and a collision-resistant hash-function h. The values p, g and h would typically be shared by a large domain of users, and could be distributed as part of an implementation of the scheme. Alternatively, the agreement of these values could be done using one of the mechanisms proposed in [59]. Note also that the secure channel might simply be a physical link between a user PC and the CA's registration system, set up at the time the user physically registers with the CA.

The proposed protocol consists of the following steps.

- User U chooses a private key component x, computes $h(g^x)$, and sends it to the CA.

$$U \xrightarrow{h(g^x)} CA \tag{8.1}$$

- The CA chooses a second private key component y, and also computes y^{-1} mod q (where q is the multiplicative order of g modulo p). The CA sends y to the User.

$$U \xleftarrow{y} CA \tag{8.2}$$

- U computes its private key as $xy \bmod q$ and sends its public key $g^{xy} \bmod p$ to the CA.

$$U \xrightarrow{g^{xy}} CA \tag{8.3}$$

- The CA computes $h((g^{xy})^{y^{-1}} \bmod p)$ and checks that the result equals the value sent by U in the first message. If the check is successful the CA accepts $g^{xy} \bmod p$ as the public key of U, certifies it, and returns the certificate to the user.

$$U \xleftarrow{\mathrm{Cert}_{CA}(g^{xy})} CA \tag{8.4}$$

Theorem 1 The above protocol has the following properties, assuming that the discrete logarithm problem is intractable, that g has multiplicative order q modulo p where q is a large prime, that the CA chooses y uniformly at random from the set of possible values, and that the hash-function h is one-way.

1. The user cannot choose his private key.
2. The CA does not know the value of the user's private key.

Proof

1. The user chooses x before y is chosen, and because of the one-way property of h cannot change x once it has been chosen. As y ranges over the full set of possible values, so does $(xy \bmod q)$, and hence xy is equally likely to be any of the set of possible values.
2. Suppose that, given $h(g^x)$, y and $g^{xy} \bmod p$, it is feasible to find xy (for any x and y), i.e. suppose claim (2) is not true. This is equivalent to saying that given $g^z \bmod p$ and a value y such that $z = xy$ for some (unknown) x, then it is feasible to find z. (Note that $h(g^x)$ is of no value since it can be computed from y and

$g^{xy} \bmod p$). However, the information regarding the value y is of no value at all, since given any value y^*, there will exist a value x^* such that $x^*y^* = z$. Hence the assumption is equivalent to saying that given $g^z \bmod p$ then it is feasible to find z, which contradicts our assumption that the discrete logarithm problem is hard [59].

Use of this protocol for the establishment and certification of all the users' key agreement keys ensures that a user cannot influence a key established with another user. In other words, the protocol prevents the user choosing his key agreement key to have certain specific properties [63]. Thus, in particular, B will not be able to choose part of the shared secret key established between A and himself, even if B has a long time to work on A's public key agreement key. However, although the user cannot control the generation of his key agreement key, the user's privacy is protected as neither the CA nor any other entity get knowledge of the generated private key agreement key. It is also clear that, through the use of 'commitments', the CA cannot choose part of the final key. This mechanism, as mentioned earlier, could more generally serve as a fair certification protocol for Diffie-Hellman keys in cases where neither user nor CA is trusted to choose the user's private key 'on their own'. As long as one party's contribution is random, the resulting key will be 'good'.

The value of g will typically be fixed for a particular application. If the multiplicative order of g (q say, where $q \mid (p-1)$) is non-prime, as would be the case if g were chosen to be primitive, then the user can have an influence on the value of their private key, albeit at the cost of choosing a rather 'weak' key. To see how this might arise, suppose r is a small prime dividing q. The user now chooses x so that g^x has order r. This is easily achieved by choosing a random value z ($0 < z < r$) and putting $x = zq/r$. Whatever the CA chooses as the value y, the final private key will be one of a set of r possible values. To avoid this pathological case it is necessary to choose q to be prime, and also for the CA to check that $g^x \neq 1$.

8.5 Summary

In this chapter we looked at key recovery from an entirely different perspective, which is the prevention of unfair key recovery. A protocol has been proposed that prevents either of two communicating parties using one of the key agreement mechanisms described in ISO/IEC 11770-3 [38] from choosing the generated key, while enabling a covert channel for key recovery by a third party. The protocol provides fair key agreement for this standardised mechanism, for which the *commitments* based solution proposed in [63] does not apply.

The new protocol can also serve as a certification protocol for Diffie-Hellman keys. The protocol enables a user to be provided with a certificate for a Diffie-Hellman public key such that the user does not choose his/her private key, but neither is this private key known to anyone other than the user.

Part V

LAWFUL INTERCEPTION OF ENCRYPTED TELECOMMUNICATIONS

Chapter 9

Key recovery in ASPeCT authentication and initialisation of payment protocol

Contents

In this chapter certain modifications to the ASPeCT (Advanced Security for Personal Communications Technology) authentication and initialisation of payment protocols are described that give them a key recovery capability. The two proposed solutions fulfil potential government demands for lawful interception of encrypted communications, while protecting the user from unauthorised disclosure of his/her communications.

9.1 Lawful interception of telecommunications in UMTS

Almost since the first introduction of public telecommunications networks, governments have demanded a lawful interception capability, mainly for the investigation of serious crime and for national security reasons. Laws of many countries and regional institutions (e.g. European Union) mandate the existence of a capability for lawful interception of traffic and related information, and this is typically part of the licensing and operational conditions of network operators [21]. Until the widespread employment of encryption for the protection of communications became possible, access to data was just a matter of line-tapping or listening to the air interface. The introduction of confidentiality services for protecting communications, however, has driven governments to try and find means of accessing encrypted data. One possible solution is the use of key recovery (escrow) services.

In a *Universal Mobile Telecommunications System* (UMTS) [88] environment many types of data might be involved, not just the 'voice' data found in GSM. In UMTS, communicated data comprises all forms of data that can be generated by user applications (emails, messages), signalling data (charging and billing), or control data (routing data, network access data). Recovery of transient keys, used for protection of the communicated data, is designed to meet LEA needs for lawful interception of communications. Intercepted information comprises both the *content of communication* and *intercept related information*, which is call related and non-call related information [23].

Among the authentication schemes proposed for a *third generation mobile communication system* (3GMS) were the ones designed and implemented by the collaborative research project ASPeCT [2]. In this chapter certain modifications to the ASPeCT authentication and initialisation of payment protocols are proposed that give them a key recovery capability. The solutions were proposed to assist the international deployment of the ASPeCT protocol, by meeting potential government demands for lawful

interception of encrypted communications.

9.2 The ASPeCT authentication and initialisation of payment protocol

The ASPeCT *authentication and initialisation of payment* (AIP) protocol [34] was developed for authentication between a user U and a *value added service provider* (VASP) V in UMTS environments. One of the basic properties of the AIP protocol is the establishment of a secret session key K which can be used to encrypt subsequent communications between the two entities. Two basic models have been designed for this purpose (B and C variants). The main difference between the B and C variants is the use in the C variant of an on-line TTP which serves as U's certification authority.

9.2.1 Preliminaries

Both variants are presented using similar notation, and both rely on certain pre-agreed 'domain parameters'. In particular, it is assumed that U and V share Diffie-Hellman key exchange parameters, including a modulus p and a 'base' g (where g is an element of large prime multiplicative order modulo p). For simplicity of presentation we write g^a for (g^a mod p) throughout; U and V must also share a set of agreed cryptographic functions as follows.

- Collision-resistant hash-functions $h1$, $h2$, and $h3$;
- A (pseudo–)random number generator;
- A symmetric encryption function where $\{M\}_K$ denotes encryption of message M using key K;
- A public system parameter T which gives the maximum number of 'ticks', i.e. the maximum transaction value, to which the user can commit himself by one

signature;

- A family of length-preserving one-way functions F_x, where a 'length-preserving one-way function' is a one-way function that inputs and outputs bit strings of the same length. For any positive integer i we write F_x^i for the ith power of F_x, i.e. the result of iteratively applying F_x a total of i times. Note that T and F are used only for the needs of the payment protocol which is performed after the AIP protocol described here.

Finally note that, as elsewhere in this thesis, we use $\|$ to denote concatenation of data items.

9.2.2 Authentication without an on-line TTP (B–Variant)

This variant of the AIP protocol assumes that U is in possession of a valid certificate for V's public key agreement key and V has a valid certificate on the public key of U's asymmetric signature system. A detailed description of this model is given in [34] and the messages exchanged are specified in Figure 9.1.

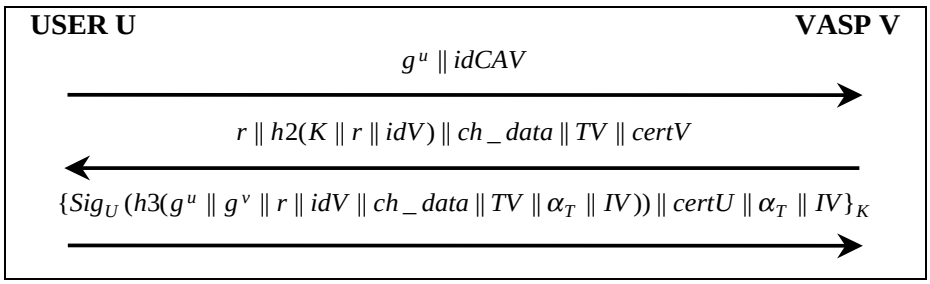


Figure 9.1: ASPeCT AIP Protocol (B–Variant)

In this protocol, U generates a random number u, computes g^u and sends it to V, together with the identity $idCAV$ of the authority whose certificates U can verify. On receipt of the first message, V generates a random number r and computes a session key $K = h1((g^u)^v \| r)$, where v is V's secret key agreement key. V then sends U

the number r, the hash value $h2(K \parallel r \parallel idV)$ (where idV is an identifier for V), its certificate $certV$, a time-stamp TV, and charging-relevant data ch_data. On receipt of the second message, U computes the key $K = h1((g^v)^u \parallel r)$, recomputes the hashed value $h2(K \parallel r \parallel idV)$ and compares it with the one received. If the check succeeds, U generates random numbers IV and α_0, computes $\alpha_T = F_{IV}^T(\alpha_0)$ and signs the message shown in Figure 9.1, including IV and α_T. U then concatenates the signature with his certificate $certU$, α_T, and IV, encrypts the concatenated parameters with K, and sends the encrypted message to V.

9.2.3 Authentication with an on-line TTP (C–Variant)

The second authentication protocol involves an on-line TTP. The main role of the TTP is real-time verification of certificates and to ensure that both the user and the VASP are able to verify each other's certificates. A brief description of the ASPeCT authentication protocol with on-line TTP is provided here. It is based on the protocol published in [33] and has been enhanced to prevent a content verification attack (see [56] and [32]). The messages exchanged are specified in Figure 9.2 and a full description and analysis of the protocol is given in [34].

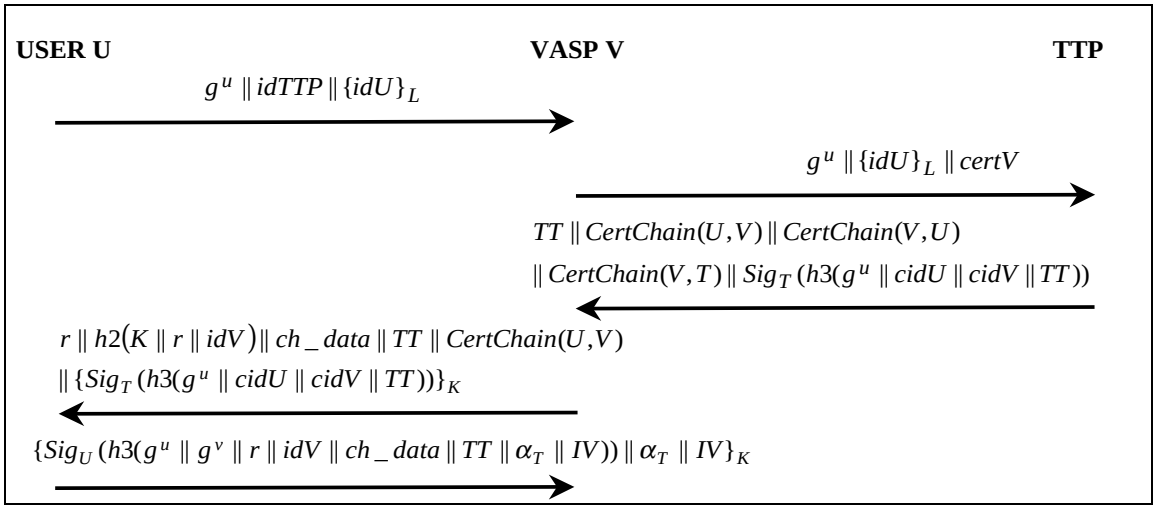


Figure 9.2: ASPeCT AIP Protocol (C–Variant)

In this variant of the protocol, U generates a random number u, computes g^u and sends

it to V, together with the identity $idTTP$ of his TTP and his own identity idU encrypted under session key $L = g^{uw}$, where g^w is the TTP's public key agreement key. As soon as V receives the first message it connects to U's TTP and forwards the message sent by U together with its certificate $CertV$. On receipt of the second authentication message the TTP checks whether U's, and optionally V's, certificates have been revoked. If both certificates are still valid, the TTP generates the certificate chains and sends them back, together with a time-stamp TT, and a signature on the concatenation of the certificate identifiers $cidU$ and $cidV$, the time-stamp TT and the random number g^u. V verifies $CertChain(V, U)$ and the signature using the TTP's public key which it retrieves from $CertChain(V, T)$. It then generates a random number r and computes the session key K and a hash value on K, concatenated with the random number r and V's identity idV. V also encrypts the signature with key K.

This encryption was incorporated in order to prevent a content verification attack in the air interface. V then forwards to U the encrypted signature together with the hash value $h2(K \parallel r \parallel idV)$, the cross-certificate for V's public key $CertChain(U, V)$, the random number r, the time-stamp TT and charge data ch_data. On receipt of the fourth authentication message U decrypts the signature, checks its validity and that of the cross-certificate. If the checks are successful U responds with the fifth authentication message (exactly as in the B-variant).

9.3 Requirements and goals for key recovery in the ASPeCT protocol

As mentioned earlier, the AIP protocol establishes a secret session key $K = h1(g^{uv}, r)$. This key can be used to encrypt subsequent communications between U and V. The aim of the modifications proposed below is to give authorised entities access to the generated session key K. This transient key will be the only target of the key recovery mechanism.

One of the main requirements for the key recovery mechanism is to minimise any additional computational overhead at the user end. This is desirable because all the user computations are typically performed by a smart card. An effective solution would therefore be to make the key recovery mechanism part of the key establishment process. The proposed key recovery mechanism, however, should not affect the key establishment process nor any of the desirable properties of the existing protocol. More importantly, it must not reduce the strength of the cryptographic system nor introduce any vulnerabilities into the protocol. The system should also be able to provide the recovered material within a very limited time period.

A key recovery mechanism that enables LEA access to plaintext should also protect the user against unauthorised access to subsequently, or previously, communicated data. Such a risk could arise if the recovered key has a greater lifetime than the period for which the LEA has authorized access to communicated data [22].

Two different solutions to the key recovery problem are proposed. Although both solutions apply to both basic models of the ASPeCT protocol (B and C variants), for brevity we apply only one solution to each model.

9.4 Giving a key recovery capability to the B-variant

The first variant of the protocol examined is the B-variant described in Section 9.2.2. This protocol can be given a key recovery capability by slightly modifying the way that U's key component u is generated. Note that, in the existing variants of the protocol, the value u is chosen at random by U prior to the start of the protocol.

The user's key component generation becomes a two-phase procedure. First, there is a *key recovery registration* phase where the user registers an initial secret value k_u with his TTP in an escrow-like mechanism. Second, each time the user wants to generate a key component he generates a random (or serial) number s and combines s and k_u to

get the key component u. That is, $u = f(k_u, s)$ where f should be a one way function. In order for the TTP to be able to compute the value u, U has to send the TTP its identity idU and the value s encrypted under $L = (g^w)^u$, where g^w is the TTP's public key agreement key. Hence, the modified scheme requires the TTP to have a key agreement key, as in the C-variant. The modified first message is as specified in Figure 9.3 (the other messages are as previously; see Figure 9.1).

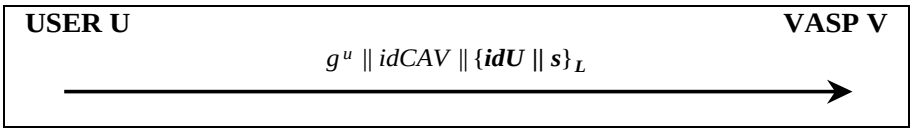


Figure 9.3: Modified B-variant Protocol

In U's domain, the keys can be recovered as follows.

- The entity requesting key recovery has to pass the following intercepted values to U's TTP, which acts as a KRA:

  1. The one-time random value g^u, V's certificate $certV$, the random value r and the encrypted string $\{idU \parallel s\}_L$. The TTP, using the value g^u and its private key agreement key w, can compute the session key L and therefore decrypt the encrypted string $\{idU \parallel s\}_L$. The value idU will help the TTP identify the user and hence retrieve the stored secret k_u. This will enable the TTP to compute the value u and, already having the values r and g^v, to recover the key K and send it to the requesting entity.
  2. The last authentication message sent by U to V together with the charging data ch_data and the time-stamp TV. These values will help the TTP verify U's signature so that it can check that the request is within the scope of the warrant.

In V's domain, however, the procedure is slightly different. This is because it would typically not be desirable to send the user's secret key component to V's TTP (especially when U's and V's TTPs are in different domains or simply if V's TTP is not

trusted by the user). Therefore, V has to register the private key agreement key v with its TTP. This can be done at the time that a certificate on the public key agreement key g^v is requested and issued. However, the key recovery procedure followed by V's TTP is almost the same as the one described above. The only difference is the way that V's TTP recovers the session key, i.e. it uses V's private key v and the value g^u to directly compute the session key K.

It should be noted that the value s could also be sent in clear (and not encrypted under L). In such a case the function f must have the property that, given the input value s, an adversary cannot get any information on the output u (without knowledge of k_u).

More generally, if s is sent in clear, a second one-way function f^* could be employed to increase flexibility. The user would keep a long term secret k_u^* (also known to the user's TTP). From this value the user would compute a 'fixed term' secret k_u, by combining k_u^* and a date stamp using f^*. In such a case the TTP could disclose the value k_u for a particular time period to the intercepting authority, and would thereby only reveal the user's key values u for a fixed time interval. However, the flexibility provided in the user's domain is not available in V's domain, since if V's private key agreement key v is revealed, then all previous and subsequent communications to and from the VASP can be decrypted. In most scenarios this will be inappropriate, so the TTP must only pass the session key K to the entity requesting recovery.

9.5 C-variant protocol with key recovery capability

In this section another solution to the key recovery problem is proposed which, as mentioned earlier, can also apply to the B–Variant. Essentially, this variant can be given a key recovery capability simply by passing the TTP the key component u encrypted under the secret key L. This gives the TTP the ability to recover the key K. With this change, the first two messages of the enhanced protocol are as shown in Figure 9.4

(where the other messages are as previously; see Figure 9.2).

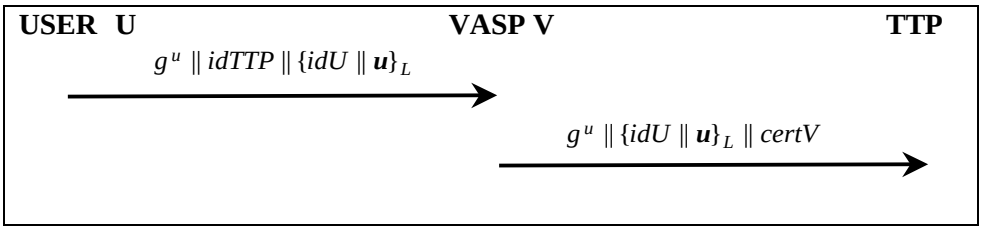


Figure 9.4: Modified C-Variant Protocol

As previously mentioned, in this solution U simply passes to its TTP the generated key component u encrypted under L. Thus, when intercepting the communication between the user and the VASP, all the information needed by the user's TTP to compute the session key K is available. The key recovery procedure is almost the same as in the previous solution in both U's and V's domain except for the session key K computation and the fact that the TTP's signature is sufficient to check that the request is within the scope of the warrant. Thus, in the user's domain the entity requesting key recovery has to give the following intercepted values to the TTP:

1. The one-time random value g^u, the certificate chain $CertChain(U, V)$, the random number r, and the encrypted string $\{idU \parallel u\}_L$. The TTP, using the value g^u and its private key agreement key w, can compute the session key L and therefore decrypt the encrypted string $\{idU \parallel u\}_L$. Having also the values r and g^v, the TTP will be able to recover the key K and send it to the requesting entity.

2. The time-stamp TT together with the TTP's signature. These will enable the TTP to verify that the request is within the scope of the warrant. As mentioned earlier, this signature contains all the information the TTP needs to make this verification and there is no need to give the TTP the last authentication message.

In V's domain the key recovery procedure is the same as in U's domain. However, as with the previous solution, V has to register the private key agreement key v with its TTP.

9.6 Properties and discussion

The main aim of the two solutions described is to give authorised entities access to transient keys and therefore access to communications. It should be noted that it is not only LEAs that might benefit from such a property. Consider an employee who is using a company's device for his communications. It is clear that the company could legitimately wish to discover what purposes this device is being used for. Key recovery for the session key could come to serve this purpose and therefore protect business.

One of the main concerns in the design of key recovery mechanisms that give LEAs access to plaintext, is the protection of the user from subsequent unauthorised access to his/her communicated data. Problems could arise if the recovered key has a greater lifetime than the period for which a LEA has authorised access to communicated data. The solutions described prohibit such unauthorised listening to communications. In the second solution only session keys are recovered, which means that LEAs can decrypt only the communication sessions they are authorised to. However, if the value s is sent in clear, the first solution gives more flexibility in the user's domain in terms of time-bounding recovered keys.

Finally note that the existence of an on-line TTP helps avoid single rogue user attacks in U's domain. If there is a strong requirement for the prevention of such attacks, the TTP might check whether the encrypted value u corresponds to the public value g^u it receives in the second authentication message. This check is not possible if there is no on-line TTP (B–Variant).

9.7 Summary

In this chapter two mechanisms were proposed that give the ASPeCT authentication and initialisation of payment protocols a key recovery capability. The modified pro-

tocols enable warranted law enforcement access to encrypted communications while protecting the user from further unauthorised disclosure of his/her data.

The intention of the proposed solutions was to assist the international deployment of the ASPeCT infrastructure, given potential government requirements for provision of warranted access to encryption keys used to protect mobile telecommunications.

Chapter 10

Discussion and conclusions

Contents

This final chapter summarises the primary contributions of this thesis and concludes with suggestions for future work.

10.1 Contributions and findings

This thesis has investigated the use of key recovery mechanisms primarily in a business environment. Within this context we identified the threats that motivate the deployment of a key recovery mechanism as a countermeasure. More specifically, we discussed the threat of lack of access to decryption keys arising from deliberate actions by disgruntled employees, or because of accidents, such as the failure of devices used for storing decryption keys. Loss of decryption keys leads to an inability to access potentially valuable information, a situation that corporations will typically not wish to tolerate. In addition, authorised access to encrypted communications might also be needed by a corporation wishing to run checks, either on incoming traffic for malicious data or on outgoing communications for leakage of sensitive information.

The outcome of this introductory investigation was that key recovery can be an essential tool for corporations and enterprises, and can be considered as part of their disaster recovery planning. Key recovery in a business environment can provide benefits to individual users as well as to the business as whole. The ethical issues arising from the use of key recovery in a business environment also appear less difficult than those arising from its proposed use by law enforcement agencies, which have aroused considerable disquiet.

Using the identified threats, we specified the requirements that a key recovery mechanism should satisfy when deployed in a business environment. While most of these requirements are likely to be important, some of them are less important than others, and their criticality is mainly affected by the nature of the target keys. More specifically, we concluded that there should be a distinction between those mechanisms deployed for communicated data and those used for archived data. These differing requirements, and the fact that most of the existing mechanisms were designed for communicated data, have motivated a study of the use of existing key recovery mechanisms for archived data. This study showed that these mechanisms might not constitute an ideal solution

when they are used on archived data.

Among the plethora of existing key recovery mechanisms is the one proposed by Maher in [53]. Although this mechanism has been designed to be used on archived data, it does not satisfy all the requirements identified in this thesis. More specifically, although the mechanism has the advantage that it does not require an on-line agent (unlike many key escrow mechanisms), it has the disadvantage that is vulnerable to rogue user attacks. A rogue user can alter or delete the generated key recovery information, thus preventing authorised recovery of the decryption key. This is one of the problems that we identified as a weak point of most existing key recovery mechanisms when applied to archived data.

The other problem that existing key recovery mechanisms face is that they do not offer the user the ability to recover his keys on his own, thus forcing the user either to keep a backup of these keys locally or to contact his agent whenever access to encrypted data is required. The first approach might introduce problems associated with the management and protection of the backup of keys, while the second introduces a communication overhead, and might also lead to an inability to access the required keys if the communications channel between the user and the agent is not available.

Given the lack of mechanisms that can be used for encrypted data, the design of a mechanism that can efficiently overcome the two aforementioned problems is highly desirable. As a result, two new schemes were proposed that fulfil the requirements identified for a key recovery mechanism used with archived data. Both mechanisms give the user the ability to recover his keys without the agent's intervention, while being robust against certain rogue user attacks. More specifically, we assume that in an attempt to bypass the key recovery functionality a rogue user might alter or delete the generated key recovery information, or even prohibit its generation. It is very difficult to prevent attacks where a rogue user deploys his own cryptographic mechanisms to encrypt his data.

The two proposed mechanisms prevent rogue user attacks by different means. The first imposes a residual work factor on the agent recovering the keys when a rogue user deletes the generated key recovery information. That is, the agent will have to recover the required key through a trial-and-report technique. The other mechanism requires the agent's participation during key generation and mandates the use of smart cards. The agent can be external to the corporation without endangering the secrecy of the keys.

The feasibility of deployment of the first mechanism was studied by implementing it using a Giesecke&Devrient Sm@rtCafé Java Card. One finding from this prototype implementation is that bulk data encryption on the card is still a prohibitively time-consuming operation. With current technology it is thus most appropriate to use the card for key generation, while encryption takes place on another platform, such as a PC. Secure messaging between the card and the PC, and certain integrity checks on the software that runs on the PC, are two techniques that can help ensure that a rogue user will not alter the key after its generation to prevent authorised recovery of his keys.

Another major theme of this thesis was the investigation of interoperability issues that arise from the use of dissimilar key recovery mechanisms in encrypted communications. It is likely that the use of dissimilar mechanisms will force communicating parties either to abandon the establishment of a secure communication or not to use key recovery. Both situations introduce potential problems, with the first case leading to a denial of service, and the second giving rise to potential data loss through lack of a key recovery capability.

The investigation of interoperability issues was performed in parallel with similar work carried out by the Key Recovery Alliance. One of the outcomes of this latter work was the proposal of the 'Common Key Recovery Block Format' designed to enable interoperation between dissimilar mechanisms. Detailed study, however, showed that

the Key Recovery Alliance scheme fails to achieve one of its main objectives, and that it is not always applicable. The varying properties of existing key recovery mechanisms render a single model solution, such as the one proposed by the Key Recovery Alliance, inappropriate for use on every mechanism.

Instead, we proposed a simpler approach where the two entities negotiate the key recovery mechanism(s) to be used in the encrypted communications. Negotiation of cryptographic parameters is an approach that has been empirically proven to work efficiently in environments where there is a variety of incompatible mechanisms available. Following this model, we proposed a protocol where the two parties can negotiate the key recovery mechanism(s) to be used, with the ultimate goal of agreeing on a mutually acceptable mechanism or on different, yet interoperable, mechanisms. The proposed protocol can be used to provide key recovery functionality at the application layer in contrast to a protocol proposed by the Key Recovery Alliance, which targets the IP layer. Note also that because the latter approach uses the Common Key Recovery Block Format mechanism, it suffers from the problems already identified with this mechanism.

Another issue that has been investigated within this thesis is the prevention of unfair key recovery by either of the parties in an encrypted communication. A protocol to enable 'fair key agreement' in one of the standardised mechanisms in ISO/IEC 11770-3 [38] was given. This protocol can also serve two additional purposes; it can be used as a fair certification protocol for Diffie-Hellman keys, and can also be used to prevent unfair key recovery. The term "fair certification" is used to describe the issue of public key certificates by a certification authority where the subject cannot choose his private key, and hence prevent unfair key agreement, while the private key is not disclosed to the certification authority. Unfair key recovery is used to describe the situation where one of the communicating parties chooses part of the negotiated session key, and hence enables a covert channel for recovery of this key by another entity. This would typically be an undesirable situation for the other party, especially in environments where the

two entities agree not to use key recovery.

Although, as already mentioned, this thesis is primarily concerned with the use of key recovery in a business environment, the last main section was devoted to the addition of a key recovery capability to one of the authentication protocols designed in the context of third generation mobile communications. This work was motivated by the possibility that governments would require future mobile telecommunications protocols to provide for warranted access to encryption keys. The proposed modifications were designed to protect users against unauthorised access to their communications. Access to user decryption keys is restricted to those protecting data to which the requesting entity is authorised to have access.

10.2 Discussion and suggestions for future work

One of the problems that the work carried out for this thesis has faced is the lack of standardisation for key recovery mechanisms. The existence of a plethora of proprietary key recovery schemes proposed by industry and academia, with no standards, presents a challenging problem to future users of key recovery techniques.

This problem becomes worse when one considers the interoperability issues that arise from the use of dissimilar mechanisms. Dealing with interoperability problems between non-standardised key recovery mechanisms requires consideration of the properties of the majority, if not all, of the existing schemes. This process would be much easier if the wide spectrum of key recovery schemes could be reduced to a small number of standardised mechanisms.

The adoption of some of the existing mechanisms as de facto standards does not necessarily imply that there is no need to take further action. It is vital to check the soundness of these mechanisms before they start being used extensively.

Although the Key Recovery Alliance has worked on the standardisation of key recovery mechanisms and on issues related to their deployment, the alliance was dissolved without having the results adopted by any of the existing standardisation bodies. One way of progressing standardisation efforts would be to put the negotiation protocol proposed in this thesis forward for adoption by an appropriate body such as the IETF. We believe that such an action, in conjunction with the work that has been presented in this thesis, would help to raise the awareness of the need for standardisation in this area.

It would appear that the general suspicion of research on key recovery, which mainly results from potential government requirements for access to encrypted communications, is holding back progress on key recovery mechanisms and their use in a business environment. Unless the benefits of the use of key recovery in the commercial environment are better understood, companies are likely to continue using proprietary mechanisms. Key recovery will potentially continue to be an issue that experts will be reluctant to talk about, yet many cryptographic products will transparently use it.

As the use of IT continues to advance, the threats that corporations will face from disgruntled employees are likely to increase. The deployment of key recovery, however, is not just a countermeasure to these threats. Key recovery is a security mechanism that can work to the benefit of both users and corporations. It can encourage employees to use encryption, which is vital for the protection of information, and it can also give businesses continuous access to their encrypted data, as well as access to their encrypted communications. In the future we expect that key recovery will be widely seen as a crucial component of the security infrastructure for all organisations.

Appendix A

Code Listings

A.1 KeyRecDeclarations

```
/*****************************************************************
***         Royal Holloway University of London           ***
***            Information Security Group               ***
*****************************************************************
  FILE NAME   : KeyRecDeclarations
  CLASS       : KeyRecApplet
  PACKAGE     : keyrecovery

  AUTHOR      : Konstantinos Rantos                                  10
  DATE        : May 2000
  DESCRIPTION : Part of the Applet. Declarations of the CLA and
                INS bytes, and of the PIN related constants
*****************************************************************/

package keyrecovery;

public interface KeyRecDeclarations
{
  // codes of CLA byte in the command APDU header                   20
  final static byte KeyRec_CLA =(byte)0xB0;
  final static byte ISO_CLA = (byte)0x00;

  // codes of INS byte in the command APDU header
  final static byte VERIFY = (byte) 0x20;
  final static byte EncryptFile = (byte) 0x50;
  final static byte DecryptFile = (byte) 0x60;

  // maximum number of incorrect tries before the
  // PIN is blocked                                                 30
  final static byte PinTryLimit =(byte)0x03;
  // maximum size PIN
  final static byte MaxPinSize =(byte)0x04;

  // status word (SW1-SW2) to signal that PIN verification has failed

  final static short SW_PIN_FAILED = (short)0x63C0;
}
```

A.2 KeyRec

```
/*************************************************************
***         Royal Holloway University of London           ***
***            Information Security Group                 ***
*************************************************************
  FILE NAME   : KeyRec
  CLASS       : KeyRec
  PACKAGE     : keyrecovery

  AUTHOR      : Konstantinos Rantos                                    10
  DATE        : May 2000
  DESCRIPTION : Part of the Applet. Contains the implementation of
                the core functions of the key recovery applet
*************************************************************/

package keyrecovery;

import javacard.framework.*;
import com.gieseckedevrient.javacardx.crypto.*;
                                                                       20

public class KeyRec implements KeyRecDeclarations{

  /* Constants declarations */
  static byte PIN_VALUE[] ={0x34,0x33,0x32,0x31};

  protected OwnerPIN pin;

  //The following are dummy values used for the user's and agent key pairs
  private short userprivate = (short) 21;                              30
  private short userpublic = (short) 285;
  public short agentpublic = (short) 251; //the corresponding private is 16;
  private short keycomponent = (short) 89;

  private byte[] buffertobehashed;
  private byte[] m_hashedbuffer;

  private CipherECB         m_cipherECB;
  private MessageDigest      m_sha1alg;
  private SymmetricKey       m_ECBkey;                                 40



  public KeyRec() {

    JCSystem.makeTransientShortArray( (short) 1, JCSystem.CLEAR_ON_DESELECT );

    buffertobehashed = new byte[8];
    m_hashedbuffer = new byte[20];
                                                                       50
    m_cipherECB = CipherECB.getInstance(Cipher.ENGINE_DES);
    m_ECBkey = new SymmetricKey((short) 0x08, JCSystem.CLEAR_ON_DESELECT );
    m_sha1alg = MessageDigest.getInstance(MessageDigest.ENGINE_SHA1);

    pin = new OwnerPIN(PinTryLimit, MaxPinSize);
    pin.reset();
```

```
  pin.update( PIN_VALUE, (short)(0), (byte)(4));
}

//The generatehashdata() function prepares the data that will be used
//by the generatekey() function for key generation

public byte[] generatehashdata(byte[] m_buffer){
  //put the exponentiation g^xy mod p into the beggining
  //of the buffertobehashed array
  Util.setShort(buffertobehashed, (short) 0x00, (short) keycomponent);
  //Concatenate the ''data credentials'' contained in the m_buffer
  Util.arrayCopy(m_buffer, ISO7816.OFFSET_CDATA, buffertobehashed,
             (short) 0x02, (short) 0x06);

  return(buffertobehashed);
}

//The generatekey() function provides the key for the file encryption.

public byte[] generatekey(byte[] m_buffer) {

  m_sha1alg.generateDigest(generatehashdata(m_buffer), (short) 0x00,
                     (short) 0x08, m_hashedbuffer, (short) 0x00);

  return(m_hashedbuffer);
}

//The encryptdecrypt() function encrypts/decrypts ''bytestoencrypt'' bytes of data
//found in m_buffer. The data to be encrypted/decrypted are positioned after the
//''data credentials'' at the m_inoffset position of the m_buffer.
//''encordec'' denotes whether the data are to be encrypted or decrypted.

public byte[] encryptdecrypt(boolean encordec, byte[] m_buffer, short m_inoffset,
                     short bytestoencrypt, boolean generatenewkey) {
  if ( ! pin.isValidated() )
    ISOException.throwIt(ISO7816.SW_PIN_REQUIRED);

  if(generatenewkey)
    m_ECBkey.setValue(generatekey(m_buffer), (short) 0x00, (short) 0x08);

  m_cipherECB.setKey(m_ECBkey);

  if (encordec)
    m_cipherECB.encrypt(m_buffer, m_inoffset, bytestoencrypt,
                  m_buffer, (short) 0x00, RunCipher.PADDING_ISO00);
  else
    m_cipherECB.decrypt(m_buffer, m_inoffset, bytestoencrypt,
                  m_buffer, (short) 0x00);
  return(m_buffer);
} // end encryptdecrypt()


public void VerifyPIN(byte[] m_buffer, byte m_byteRead) {
  // validate pin provided by the user
  if (pin.check(m_buffer, (short) ISO7816.OFFSET_CDATA, m_byteRead) == true)
  return;

  // If the PIN check failed, throw the apropiate ISOException.
  short sTries = pin.getTriesRemaining();
```

```
    ISOException.throwIt( (short) (SW_PIN_FAILED + sTries) );
  } // end VerifyPIN()
}
```

A.3 KeyRecApplet

```
/*************************************************************
***         Royal Holloway University of London           ***
***            Information Security Group                 ***
*************************************************************
  FILE NAME   : KeyRecApplet
  CLASS       : KeyRecApplet
  PACKAGE     : keyrecovery

  AUTHOR      : Konstantinos Rantos                                    10
  DATE        : May 2000
  DESCRIPTION : Contains the key recovery applet functionality
*************************************************************/

package keyrecovery;

import javacard.framework.*;


public class KeyRecApplet extends Applet implements KeyRecDeclarations{    20

  /* Constants declarations */
  private KeyRec m_AppletKeyRec;
  private byte buffer[]; // APDU buffer

  public static void install(byte[] buffer, short offset, byte length){
       new KeyRecApplet();
  } // end of install method

  //Constructor for KeyRecApplet                                          30
  private KeyRecApplet() {
       // create a KeyRec applet instance
       m_AppletKeyRec = new KeyRec();
       register();
  } // end of the constructor

  public boolean select() {
       return true;
  }// end of select() method
                                                                         40
  public void process(APDU apdu) {

    if( selectingApplet() )
    return;
    // APDU object carries a byte array (buffer) to
    // transfer incoming and outgoing APDU header
    // and data bytes between card and CAD
    buffer = apdu.getBuffer();
    // check whether the applet can accept the incoming APDU message
    if (buffer[ ISO7816.OFFSET_CLA] != KeyRec_CLA &                      50
      buffer[ISO7816.OFFSET_CLA] != ISO_CLA)
    ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);

    switch(buffer[ISO7816.OFFSET_CLA])
    {
      case ISO_CLA:
```

```
switch (buffer[ ISO7816.OFFSET_INS])
{
  case VERIFY: {
    // retrieve the PIN for validation.
    byte byteRead = (byte)(apdu.setIncomingAndReceive());
    m_AppletKeyRec.VerifyPIN(buffer, byteRead);return;
  }
  default: ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
}break;

case KeyRec_CLA:
switch (buffer[ ISO7816.OFFSET_INS])
  {
    case EncryptFile: {
      // Lc byte denotes the number of bytes in the
      // data field of the comammd APDU
      short numBytes = (short) (buffer[ ISO7816.OFFSET_LC] & 0x00FF);
      // indicate that this APDU has incoming data and
      // receive data starting from the offset ISO.OFFSET_CDATA
      short byteRead = (short)(apdu.setIncomingAndReceive());
      // it is an error if the number of data bytes
      // read does not match the number in Lc byte
      if (byteRead != numBytes)
        ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);

      if (buffer[ISO7816.OFFSET_P1] == (byte) 1)
        //generate new key
        Util.arrayCopy(m_AppletKeyRec.encryptdecrypt(true, buffer,
                    (short) (ISO7816.OFFSET_CDATA + 0x06),
                    (short) (byteRead - 0x06), true),(short) 0x00, buffer,
                    (short) 0x00, (short) (byteRead - 0x06));
      else
        //use the generated from the previous transaction key
        Util.arrayCopy(m_AppletKeyRec.encryptdecrypt(true, buffer,
                    (short) (ISO7816.OFFSET_CDATA + 0x06),
                    (short) (byteRead - 0x06), false), (short) 0x00, buffer,
                    (short) 0x00, (short) (byteRead - 0x06));

      apdu.setOutgoing();
      //indicate the number of bytes in the data field
      if ( (byteRead - 0x06) % (short)8 != 0) {
        apdu.setOutgoingLength((short)((((byteRead - 0x06)/8)+1)*8));
        apdu.sendBytes((short)0, (short) ((((byteRead - 0x06)/8)+1)*8));
      }
      else {
        apdu.setOutgoingLength((short)(byteRead - 0x06));
        apdu.sendBytes((short)0, (short) (byteRead - 0x06));
      }
      return;
    } //end case EncryptFile

    case DecryptFile: {
      // Lc byte denotes the number of bytes in the
      // data field of the comammd APDU
      short numBytes = (short) (buffer[ ISO7816.OFFSET_LC] & 0x00FF);
      // indicate that this APDU has incoming data and
      // receive data starting from the offset
      // ISO.OFFSET_CDATA
      short byteRead = (short)(apdu.setIncomingAndReceive());
```

```
          // it is an error if the number of data bytes
          // read does not match the number in Lc byte
          if (byteRead != numBytes)
          ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
                                                                                    120
          if (buffer[ISO7816.OFFSET_P1] == (byte) 1)
            //generate new key
            Util.arrayCopy(m_AppletKeyRec.encryptdecrypt(false, buffer,
                        (short) (ISO7816.OFFSET_CDATA + 0x06),
                        (short) (byteRead - 0x06), true), (short) 0x00, buffer,
                        (short) 0x00, (short) (byteRead - 0x06));
          else
            // use the generated from the previous transaction key
            Util.arrayCopy(m_AppletKeyRec.encryptdecrypt(false, buffer,
                        (short) (ISO7816.OFFSET_CDATA + 0x06),                      130
                        (short) (byteRead - 0x06), false), (short) 0x00, buffer,
                        (short) 0x00, (short) (byteRead - 0x06));

            apdu.setOutgoing();
            //indicate the number of bytes in the data field
            apdu.setOutgoingLength((short)(byteRead-0x06));
            apdu.sendBytes((short)0, (short) (byteRead - 0x06));
            return;
          } // end case DecryptFile
                                                                                    140
          default: ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
        }
      }
  } // end of process method
} // end of class KeyRec
```

A.4 KRAppletProxy

```
/*****************************************************************
                Royal Holloway University of London
                   Information Security Group
*****************************************************************
  FILE NAME   : KRAppletProxy
  CLASS       : KRAppletProxy
  PACKAGE     : services

  AUTHOR      : Konstantinos Rantos                                  10
  DATE        : May 2000
  DESCRIPTION : Part of the client. Contains the implementation
                of the two main functions provided by that service,
                i.e. encryption and decryption
*****************************************************************/

package services;

import java.util.Hashtable;
import opencard.opt.management.ApplicationID;                        20
import opencard.core.service.SmartCard;
import opencard.core.service.CardChannel;
import opencard.core.service.CardServiceScheduler;
import opencard.core.service.CardServiceException;
import opencard.core.service.CardServiceUsageException;
import opencard.core.service.CardServiceInvalidParameterException;
import opencard.core.service.CardServiceInvalidCredentialException;
import opencard.core.service.CardServiceOperationFailedException;
import opencard.opt.service.CardServiceUnexpectedResponseException;
import opencard.core.terminal.CommandAPDU;                           30
import opencard.core.terminal.ResponseAPDU;
import opencard.core.terminal.CardTerminalException;
import opencard.core.terminal.CardTerminalIOControl;
import opencard.core.terminal.CHVControl;
import opencard.core.terminal.CHVEncoder;
import opencard.core.util.Tracer;
import keyrecovery.KeyRecDeclarations;
import com.gieseckedevrient.opencard.util.Hex;
import com.gieseckedevrient.opencard.services.SmartCafeService;
                                                                     40
// KRAppletProxy is a Card Applet Proxy for the KeyRecovApplet.
public class KRAppletProxy extends SmartCafeService
                        implements KeyRecovApplet
{

  public static final ApplicationID KEYREC_APPLET_AID =
              new ApplicationID( "KeyRecApplet".getBytes() );

  private Tracer tracer;
  private static final String where = KRAppletProxy.class.getName();  50

  private static final byte[] bzero4 = { (byte)0, (byte)0, (byte)0, (byte)0 };
  private static final byte[] bzero8 = { (byte)0, (byte)0, (byte)0, (byte)0,
                                (byte)0, (byte)0, (byte)0, (byte)0 };

  public KRAppletProxy() {
```

```
  tracer = new Tracer(this, KRAppletProxy.class);
} // end KRAppletProxy()

protected void initialize(CardServiceScheduler scheduler,                    60
                    SmartCard card, boolean blocking)
throws CardServiceException{
  super.initialize( scheduler, card, blocking );
  try {
    allocateCardChannel();
    CardChannel channel = getCardChannel();

    Hashtable ht = (Hashtable)(channel.getState());
    if( ht == null ) {
      ht = new Hashtable();                                                  70
      channel.setState( ht );
    }

    KRAppletState state = (KRAppletState)(ht.get( KEYREC_APPLET_AID ));
    if( state == null ) {
      state = new KRAppletState();
      ht.put( KEYREC_APPLET_AID, state );
    }
  }
  finally {                                                                  80
    releaseCardChannel();
  }
} // end initialize().

public byte[] encryptfile(byte[] keycredentials, byte[] datatoencrypt,
                    boolean generatekey)
throws CardServiceInvalidCredentialException, CardServiceUnexpectedResponseException,
     CardServiceException{

  try {                                                                      90
    allocateCardChannel();

    // Perform Card Holder Verification if necessary
    if (!getKRAppletState().isEncryptionAllowed()) {
      performCHV(getCardChannel(), KRAppletState.ENCRYPT_CHV);
    }
    byte m_generatekey;
    if (generatekey)
    m_generatekey = (byte) 0x01;
    else                                                                    100
    m_generatekey = (byte) 0x00;

    CommandAPDU cmd = new CommandAPDU(5 + keycredentials.length +
                              datatoencrypt.length +1);

    cmd.append( new byte[]{ KeyRecDeclarations.KeyRec_CLA,  // CLA.
                        KeyRecDeclarations.EncryptFile,  // INS.
                        m_generatekey,                  // P1.
                        (byte)0x00,                  // P2.
                        (byte)(datatoencrypt.length + keycredentials.length), // Lc.   110
                        } );
    cmd.append(keycredentials);
    cmd.append(datatoencrypt);
    cmd.append((byte)0x00); //Le
```

```
    ResponseAPDU rsp = sendCommandAPDU(getCardChannel(), KEYREC_APPLET_AID, cmd);
    int sw  = rsp.sw();
    if( sw == 0x9000 )
      return rsp.data();
    else
      throw new CardServiceUnexpectedResponseException
        ( where + "::encrypt: " + "Unexpected response, SW = "
          + Hex.short2hex( (short)(sw & 0x0000FFFF) ) );
  }
  catch( CardTerminalException x ) {
     tracer.error( "encrypt", "Caught terminal exception - " + x.getMessage() );
     return -1;
  }
  finally {
    releaseCardChannel();
  }
} // end encryptfile()


public byte[] decryptfile(byte[] keycredentials, byte[] datatodecrypt,
                    boolean generatekey)
throws CardServiceInvalidCredentialException, CardServiceUnexpectedResponseException,
     CardServiceException{

  try {
    allocateCardChannel();

    // Perform Card Holder Verification if necessary
    if (!getKRAppletState().isEncryptionAllowed()) {
      performCHV(getCardChannel(), KRAppletState.ENCRYPT_CHV);
    }
    byte m_generatekey;
    if (generatekey)
      m_generatekey = (byte) 0x01;
    else
      m_generatekey = (byte) 0x00;

    CommandAPDU cmd = new CommandAPDU(5 + keycredentials.length +
                                datatodecrypt.length + 1);

    cmd.append( new byte[]{ KeyRecDeclarations.KeyRec_CLA,   // CLA.
                        KeyRecDeclarations.DecryptFile,  // INS.
                        m_generatekey,                // P1.
                        (byte)0x00,                  // P2.
                        (byte)(datatodecrypt.length + keycredentials.length), // Lc.
                        } );
    cmd.append(keycredentials);
    cmd.append(datatodecrypt);
    cmd.append((byte)0x00);    //Le

    ResponseAPDU rsp = sendCommandAPDU( getCardChannel(), KEYREC_APPLET_AID, cmd );
    int sw = rsp.sw();
    if( sw == 0x9000 )
      return rsp.data();
    else
      throw new CardServiceUnexpectedResponseException
        ( where + "::decrypt: " + "Unexpected response, SW = "
          + Hex.short2hex( (short)(sw & 0x0000FFFF) ) );
  }
```

```
  catch( CardTerminalException x ) {
     tracer.error( "decrypt", "Caught terminal exception - " + x.getMessage() );
     return -1;
  }
  finally {
   releaseCardChannel();                                                    180
  }
} // end decryptfile()


/**
 * Performs Card Holder Verification.
 *
 * The p_bNumCHV parameter can be used for the case that more than one
 * types of PINs used. For instance an administrator PIN can  be used
 * which can give the user privileged access to the card.                   190
 **/
protected void performCHV( CardChannel p_channel, byte p_bNumCHV )
  throws  CardServiceInvalidCredentialException,
        CardServiceOperationFailedException,
        CardServiceUnexpectedResponseException,
        CardServiceException,
        CardTerminalException
{
  byte bLength = 4;
                                                                           200
  String message = new String();
  if(p_bNumCHV == 1)
   message = "  Please enter  " +
           "   ENCRYPTION/DECRYPTION PIN    " +
           "     ";

  CommandAPDU cmd    = new CommandAPDU( 5 + bLength + 1 );

  cmd.append( new byte[]{ KeyRecDeclarations.ISO_CLA,     // CLA.
                    KeyRecDeclarations.VERIFY,      // INS.                  210
                    (byte)0x20,                  // P1.
                    (byte)(0x80 | p_bNumCHV),      // P2.
                    bLength } );                 // Lc.
   cmd.append( bzero4 );                          // Data.

  CardTerminalIOControl ioCtrl  = new CardTerminalIOControl( bLength, 30, null, null );
  CHVControl          chvCtrl = new CHVControl( message, p_bNumCHV,
                                        CHVEncoder.STRING_ENCODING,
                                        0,
                                        ioCtrl );                          220


  ResponseAPDU rsp = sendVerifiedAPDU( p_channel, KEYREC_APPLET_AID, cmd,
                                chvCtrl, getCHVDialog(), 30000 );

  int sw = rsp.sw();
  if( sw == 0x9000 )
  {
   getKRAppletState().setCHVPerformed( p_bNumCHV, true );
   return;                                                                 230
  }
  else
   if( (short)(sw & 0x0000FFF0) == KeyRecDeclarations.SW_PIN_FAILED )
```

```
      throw new CardServiceInvalidCredentialException
             ( "PIN verification failed, "
               + (sw & 0x000F)
               + " retries remaining." );
      else
        throw new CardServiceUnexpectedResponseException
                 ( where + "::verify: "                                    240
                   + "Unexpected response, SW = "
                   +  Hex.short2hex( (short)(sw & 0x0000FFFF)  ) );

  } // 'KRAppletProxy.verify()'.

  protected KRAppletState getKRAppletState()
    throws CardServiceException
  {
    Hashtable ht = (Hashtable)(getCardChannel().getState());
    KRAppletState state = (KRAppletState)(ht.get( KEYREC_APPLET_AID ));     250

    if( state == null )
      throw new CardServiceException
               ( where + "::getCardState: "
                     + "No card state object available!" );
    return state;
  } // 'KRAppletProxy.getCardState()'.

  protected void appletSelected()
  {                                                                        260
  // Reset the proxy state of the applet.
    try
    {
      getKRAppletState().resetState();
    }
    catch (CardServiceException x) {};

  } // 'SmartCafeService.appletSelected()'.
} // class 'KRAppletProxy'.
```

A.5 KRAppletState

```
/*****************************************************************
                  Royal Holloway University of London
                     Information Security Group
*****************************************************************
  FILE NAME   : KRAppletState
  CLASS       : KRAppletState
  PACKAGE     : services

  AUTHOR      : Konstantinos Rantos                                  10
  DATE        : May 2000
  DESCRIPTION : Part of the client. Represents the state
                of the key recovery applet on the card
*****************************************************************/

package services;

public class KRAppletState {

  /* Constants to indicate which type of CHV is performed. */       20
  public static final byte ENCRYPT_CHV = 1;
  public static final byte ADMIN_CHV = 3;

  /* Which type of successful card holder verification
     has been performed last. This is for the case that
     administrator functionality is implemented on the card. */

  protected boolean m_zEncryptCHVPerformed = false;
  protected boolean m_zAdminCHVPerformed = false;
                                                                    30
   /*
   * Check if encrypt operations are allowed.
   * return true if allowed, false otherwise.
   */
  public boolean isEncryptionAllowed() {
    return m_zEncryptCHVPerformed;
  } // KRAppletState.isEncryptionAllowed()

   /*
   * Check if PIN administration operations are allowed.            40
   * return true if allowed, false otherwise.
   */
  public boolean isAdminAllowed() {
    return m_zAdminCHVPerformed;
  } // KRAppletState.isAdminAllowed()

   /*
   * Sets the card holder verification flag to the given value.
   * param p_bCHVType Indicates which type of CHV has been performed
   * param p_zCHVPerformed Indicates whether a successful           50
   *                     card holder verification has been performed.
   */
  public void setCHVPerformed(byte p_bCHVType, boolean p_zCHVPerformed) {
    resetState();
    switch (p_bCHVType)
    {
```

```
    case ENCRYPT_CHV:
      m_zEncryptCHVPerformed = p_zCHVPerformed;
      break;
    case ADMIN_CHV:                                                    60
      m_zAdminCHVPerformed = p_zCHVPerformed;
      break;
    }
  } // KRAppletState.setCHVPerformed()

  /* Reset the state of the key recovery applet. */
  public void resetState() {
    m_zEncryptCHVPerformed = false;
    m_zAdminCHVPerformed = false;
  } // KRAppletState.resetState()                                      70

} // class KRAppletState
```

A.6 KeyRecovApplet

```
/*****************************************************************
***         Royal Holloway University of London           ***
***            Information Security Group               ***
*****************************************************************
  FILE NAME   : KeyRecovApplet
  CLASS       : KeyRecovApplet
  PACKAGE     : services

  AUTHOR      : Konstantinos Rantos                                    10
  DATE        : May 2000
  DESCRIPTION : Part of the client. Contains the interface
                to the functions implemented in KRAppletProxy
*****************************************************************/

package services;

import opencard.core.service.CardServiceException;
import opencard.core.service.CardServiceUsageException;
import opencard.core.service.CardServiceInvalidParameterException;       20
import opencard.core.service.CardServiceInvalidCredentialException;
import opencard.core.service.CardServiceOperationFailedException;
import opencard.opt.service.CardServiceUnexpectedResponseException;
import opencard.opt.service.CardServiceInterface;

public interface KeyRecovApplet extends CardServiceInterface{

  public byte[] encryptfile(byte[] keycredentials, byte[] datatoencrypt,
                        boolean generatekey)
    throws CardServiceInvalidCredentialException,                        30
          CardServiceUnexpectedResponseException,
          CardServiceException;


  public byte[] decryptfile(byte[] keycredentials, byte[] datatodecrypt,
                        boolean generatekey)
    throws CardServiceInvalidCredentialException,
          CardServiceUnexpectedResponseException,
          CardServiceException;
} // interface 'KeyRecovApplet'.                                         40
```

Bibliography

[1] H. Abelson, R. Anderson, S.M. Bellovin, J. Benaloh, M. Blaze, W. Diffie, J. Gilmore, P.G. Neumann, R.L. Rivest, J.I. Schiller, and B. Schneier. The risks of key recovery, key escrow, and trusted third party encryption. http://www.cdt.org/crypto/risks98.

[2] Advanced Security for Personal Communications Technologies (ASPeCT). http://www.esat.kuleuven.ac.be/cosic/aspect/.

[3] D. Balenson and T. Markham. ISAKMP key recovery extensions. *Computers & Security*, **19(1)**:91–99, 2000.

[4] D.M. Balenson, C.M. Ellison, S.B. Lipner, and S.T. Walker. A new approach to software key escrow encryption. In L.J. Hoffman, editor, *Building in Big Brother, The Cryptographic Policy Debate*, pages 180–207. Springer-Verlag, New York, 1995.

[5] M. Blaze. Key management in an encrypting file system. In *Proceedings Summer 1994 USENIX Technical Conference*, Boston, MA, June 1994.

[6] M. Blaze. Protocol failure in the escrowed encryption standard. In *Proceedings of Second ACM Conference on Computer and Communications Security*, pages 59–67, FairFax VA, November 1994.

[7] E.F. Brickell, D.E. Denning, S.T. Kent, D.P. Maher, and W. Tuchman. SKIPJACK review: Interim report. In L.J. Hoffman, editor, *Building in Big Brother,*

The Cryptographic Policy Debate, pages 119–130. Springer-Verlag, New York, 1995.

[8] W.J. Caelli. Commercial key escrow: An Australian perspective. In E.P. Dawson and J. Golic, editors, *Proceedings of Cryptography: Policy and Algorithms, International Conference Brisbane, Queensland, Australia, July 1995*, pages 40–64. Springer-Verlag (LNCS 1029), Berlin (1996).

[9] W.J. Caelli and D. Longley. Key recovery - a perspective for the rest of the world. Presented at 5th Annual IT Security Summit, Sydney NSW, February 1998.

[10] L. Chen, D. Gollmann, and C.J. Mitchell. Key escrow in mutually mistrusting domains. In M. Lomas, editor, *Security Protocols – Proceedings, International Workshop, Cambridge, April 1996*, pages 139–153. Springer-Verlag (LNCS 1189), Berlin (1997).

[11] D. Denning and M. Smid. Key escrowing today. *IEEE Communications Magazine*, **32**:58–68, 1994.

[12] D.E. Denning. To tap or not to tap. *Communications of the ACM*, **36(3)**:25–44, March 1993.

[13] D.E. Denning. The U.S. key escrow encryption technology. *Computer Communications*, **17(7)**:111–118, July 1994.

[14] D.E. Denning. International key escrow encryption: Proposed objectives and options. In L.J. Hoffman, editor, *Building in Big Brother, The Cryptographic Policy Debate*, pages 208–225. Springer-Verlag, New York, 1995.

[15] D.E. Denning. *Information Warfare and Security.* Addison Wesley, 1998.

[16] D.E. Denning and W.E. Baugh. Key escrow encryption policies and technologies. *Information System Security*, **5(2)**:44–51, Summer 1996.

[17] D.E. Denning and D.K. Branstad. A taxonomy of key escrow encryption systems. *Communications of the ACM*, **39(3)**:34–40, March 1996.

[18] Y. Desmedt. Securing traceability of ciphertexts–Towards a secure software key escrow system. In L. Guillou and J. Quisquater, editors, *Advances in Cryptology–EUROCRYPT'95*, pages 147–157. Springer-Verlag (LNCS 921), 1995.

[19] T. Dierks and C. Allen. The TLS protocol, version 1.0, January 1999. RFC 2246.

[20] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, **22**:644–654, 1976.

[21] ETSI EG 201 781. *Intelligent Networks—Lawful Interception*, July 2000.

[22] ETSI ETR 331. *Security Techniques Advisory Group (STAG); Definition of user requirements for lawful interception of telecommunications; Requirements of the law enforcement agencies*, December 1996.

[23] ETSI TS 133 106. *Universal Mobile Telecommunications System (UMTS); 3G Security; Lawful Interception Requirements*, January 2000.

[24] P.A. Fouque, G. Poupard, and J. Stern. Recovering keys in open networks. In *Proceedings of IEEE Information Theory and Communications Workshop (ITW'99)*, Kruger National Park, South Africa, June 1999.

[25] Y. Frankel and M. Yung. Escrow encryption systems visited: attacks, analysis and designs. In D. Coppersmith, editor, *Advances in Cryptology–CRYPTO'95, California, USA, August 1995*, pages 222–235. Springer-Verlag (LNCS 963), Berlin (1995).

[26] E.H. Freeman. When technology and privacy collide. Encoded encryption and the Clipper chip. *Information System Management*, **12(2)**:43–46, Spring 1995.

[27] R. Gennaro, P. Krager, S. Matyas, M. Peyravian, A. Roginsky, D. Safford, M. Willett, and N. Zunic. Two–phase cryptographic key recovery system. *Computers & Security*, **16**:481–506, 1997.

[28] S. Gupta. A common key recovery block format: Promoting interoperability between dissimilar key recovery mechanisms. *Computers & Security*, **19(1)**:41–47, 2000.

[29] S. Gupta, S.M. Matyas Jr., and N. Zunic. Public key infrastructure: Analysis of existing and needed protocols and object formats for key recovery. *Computers & Security*, **19(1)**:56–68, 2000.

[30] U. Hansmann, M.S. Nicklous, T. Schack, and F. Seliger. *Smart Card Application Development Using Java*. Springer Verlag, 2000.

[31] J. He and E. Dawson. A new key escrow cryptosystem. In E.P. Dawson and J. Golic, editors, *Proceedings of Cryptography: Policy and Algorithms, International Conference Brisbane, Queensland, Australia, July 1995*, pages 105–114. Springer-Verlag (LNCS 1029), Berlin (1996).

[32] G. Horn, P. Howard, K.M. Martin, C.J. Mitchell, B. Preneel, and K. Rantos. Trialling secure billing with trusted third party support for UMTS applications. In *Proceedings of 3rd ACTS Mobile Communications Summit*, pages 574–579, Rhodes, Greece, June, 1998.

[33] G. Horn and B. Preneel. Authentication in future mobile systems. Technical Report KUL-ESAT-COSIC98-2, Katholieke Universiteit Leuven, 1998.

[34] G. Horn and B. Preneel. Authentication and payment in future mobile systems. In J-J. Quisquater, Y. Deswarte, C. Meadows, and D. Gollmann, editors, *Computer Security - ESORICS 98*, pages 539–548. Springer-Verlag (LNCS 1485), Berlin (1998).

[35] IBM SecureWay. Towards a framework based solution to cryptographic key recovery. http://www-4.ibm.com/software/security/keyworks/library/.

[36] International Organization for Standardization, Genève, Switzerland. *ISO/IEC 7816–4, Information technology—Identification cards—Integrated circuit(s) cards with contacts—Part 4: Interindustry commands for interchange*, 1995.

[37] International Organization for Standardization, Genève, Switzerland. *ISO/IEC 11770–2, Information technology—Security techniques—Key management—Part 2: Mechanisms using symmetric techniques*, 1996.

[38] International Organization for Standardization, Genève, Switzerland. *ISO/IEC 11770–3, Information technology—Security techniques—Key management—Part 3: Mechanisms using asymmetric techniques*, 1999.

[39] N. Jefferies, C. Mitchell, and M. Walker. A proposed architecture for trusted third parties. In E. Dawson and J. Golic, editors, *Cryptography: Policy and Algorithms — Proceedings: International Conference, Brisbane, Australia*, pages 98–104. Springer-Verlag (LNCS 1029), Berlin (1996).

[40] N. Jefferies, C. Mitchell, and M. Walker. Trusted third party based key management allowing warranted interception. In *Proceedings: Public Key Infrastructure Invitational Workshop*. MITRE, McLean, Virginia, USA, **NISTIR 5788**, September 1995.

[41] N. Jefferies, C. Mitchell, and M. Walker. Practical solutions to key escrow and regulatory aspects. In *Public Key Solutions '96*, Zurich, Switzerland, September/October 1996.

[42] J. Kennedy, S.M. Matyas Jr., and N. Zunic. Key recovery functional model. *Computers & Security*, **19(1)**:31–36, 2000.

[43] J. Kilian and T. Leighton. Fair cryptosystems, revisited. In D. Coppersmith, editor, *Advances in Cryptology–CRYPTO'95, California, USA, August 1995*, pages 208–221. Springer-Verlag (LNCS 963), Berlin (1995).

[44] S. Kim, I. Lee, M. Mambo, and S. Park. On the difficulty of key recovery systems. In M. Mambo and Y. Zheng, editors, *International Workshop on Information Security, Kuala Lumpur, Malaysia, November 1999*, pages 207–224. Springer-Verlag (LNCS 1729), Berlin (1999).

[45] L.R. Knudsen and K.M. Martin. In search of multiple domain key recovery. *Journal of Computer Security*, **6**:219–235, 1998.

[46] L.R. Knudsen and T.P. Pedersen. On the difficulty of software key escrow. In U. Maurer, editor, *Advances in Cryptology–EUROCRYPT'96*, pages 237–244. Springer-Verlag (LNCS 1070), Berlin (1996).

[47] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication, February 1997. RFC 2104.

[48] Y. Lee and C. Laih. On the key recovery of the key escrow system. In *Proceedings of 13th Annual Computer Security Applications Conference*, pages 216–220, San Diego, California, 1997.

[49] A.K. Lenstra, P. Winkler, and Y. Yacobi. A key escrow system with warrant bounds. In D. Coppersmith, editor, *Advances in Cryptology–CRYPTO'95*, pages 197–207. Springer-Verlag (LNCS 963), Berlin (1995).

[50] A. Maclean, S.M. Matyas Jr., and N. Zunic. Organization implementation guidelines for recovery of encrypted information. *Computers & Security*, **19(1)**:69–81, 2000.

[51] C. Madson and R. Glenn. The use of HMAC-MD5-96 within ESP and AH. RFC 2403.

[52] C. Madson and R. Glenn. The use of HMAC-SHA-1-96 within ESP and AH. RFC 2404.

[53] D.P. Maher. Crypto backup and key escrow. *Communications of the ACM*, **39(3)**:48–53, March 1996.

[54] C. Markantonakis and K. Rantos. On the life cycle of the certification authority key pair in EMV'96. In *Proceedings of Euromedia '99*, pages 125–130, Munich, Germany, April, 1999.

[55] T. Markham and C. Williams. Key recovery header for IPSEC. *Computers & Security*, **19(1)**:86–90, 2000.

[56] K.M. Martin, B. Preneel, C.J. Mitchell, H.J. Hitz, G. Horn, A. Poliakova, and P. Howard. Secure billing for mobile information services in UMTS. In S. Trigila, M. Campolargo, H. Vanderstraeten, and M. Mampaey, editors, *Proceedings of the 5th Internatiotal Conference in Services and Networks, IS&N'98*, pages 535–548. Springer-Verlag (LNCS 1430), Berlin (1998).

[57] S.M. Matyas Jr. and N. Zunic. Additional Key Recovery Functions. *Computers & Security*, **19(1)**:37–40, 2000.

[58] D. Maughan, M. Schertler, and J. Turner. Internet security association and key management protocol (ISAKMP). RFC 2408.

[59] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, 1997.

[60] S. Micali. Fair cryptosystems. In L.J. Hoffman, editor, *Building in Big Brother, The Cryptographic Policy Debate*, pages 149–173. Springer-Verlag, New York, 1995.

[61] W.J. Micali. Fair public key cryptosystems. In E.F. Brickell, editor, *Advances in Cryptology–CRYPTO'92*, pages 113–138. Springer-Verlag (LNCS 740), Berlin (1993).

[62] C.J. Mitchell and K. Rantos. A fair certification protocol. *ACM Computer Communication Review*, **29(3)**:47–49, July 1999.

[63] C.J. Mitchell, M. Ward, and P. Wilson. Key control in key agreement protocols. *Electronics Letters*, **34**:980–981, 1998.

[64] J. Nechvatal. A public key based key escrow system. *Journal of System Software*, **35(1)**:73–83, October 1996.

[65] J.G. Nieto, K. Viswanathan, C. Boyd, and E. Dawson. Key recovery system for the commerical environment. In E. Dawson, A. Clark, and C. Boyd, editors, *Information Security and Privacy – ACISP 2000*, pages 149–162. Springer-Verlag (LNCS 1841), Brisbane, Australia, 2000.

[66] National Institute of Standards and Technology. requirements for key recovery products, November 1998. Available at http://csrc.nist.gov/keyrecovery/.

[67] NSA report. threat and vulnerability model for key recovery, February 1998. http://www.fcw.com/pubs/fcw/1998/0413/web-nsareport-4-14-1998.html.

[68] OpenCard Framework — General Information Web Document, October 1998. http://www.opencard.org.

[69] OpenCard Framework 1.2 — Programmer's Guide, December 1999. http://www.opencard.org.

[70] National Institute of Standards and Technology. FIPS Publication 185: Escrowed Encryption Standard, February 1994.

[71] B. Pfitzmann and M. Waidner. How to break fraud-detectable key recovery. *ACM Operating Systems Review*, **32(1)**:23–28, 1998.

[72] K. Rantos and C.J. Mitchell. Key recovery scheme interoperability – a protocol for mechanism negotiation. Submitted.

[73] K. Rantos and C.J. Mitchell. Matching key recovery mechanisms to business requirements. Submitted.

[74] K. Rantos and C.J. Mitchell. Remarks on KRA's key recovery block format. *Electronics Letters*, **35**:632–634, 1999.

[75] K. Rantos and C.J. Mitchell. Key recovery for archived data using smart cards. In *Proceedings of the 5th Nordic Workshop on Secure IT Systems*, pages 75–85, Reykjavik, Iceland, October 2000.

[76] K. Rantos and C.J. Mitchell. Key recovery in ASPeCT authentication and initialisation of payment protocol. In *Proceedings of 4th ACTS Mobile Communications Summit*, pages 629–634, Sorento, Italy, June, 1999.

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

[78] B. Schneier. Security in the real world: How to evaluate security technology. *Computer Security Journal*, **15(4)**:1–14, 1999.

[79] A. Shamir. How to share a secret. *Communications of the ACM*, **22(11)**:612–613, March 1979.

[80] A. Shamir. Partial key escrow: A new approach to software key escrow. The Weizmann Institute, presentation at NIST Key Escrow Standards meeting, September 1995.

[81] T. Shoriak. SSL/TLS protocol enablement for key recovery. *Computers & Security*, **19(1)**:100–104, 2000.

[82] Skipjack. http://csrc.nist.gov/encryption/skipjack/skipjackkea.htm.

[83] M. Smith, P. van Oorschot, and M. Willett. Cryptographic information recovery using key recovery. *Computers & Security*, **19(1)**:21–27, 2000.

[84] M.R. Smith. *Commonsense Computer Security.* McGraw-Hill, 1994.

[85] Sun Microsystems. Java card 2.1 application programming interface, February 1999. http://www.java.sun.com/products/javacard/.

[86] Sun Microsystems. Java card 2.1 runtime environment specification, February 1999. http://www.java.sun.com/products/javacard/.

[87] Sun Microsystems. Java card 2.1 virtual machine specification, March 1999. http://www.java.sun.com/products/javacard/.

[88] Universal Mobile Telecommunications System (UMTS). http://www.umts-forum.org/.

[89] E.R. Verheul and H.C.A. van Tilborg. Binding Elgamal: A fraud-detectable alternative to key escrow proposals. In W. Fumy, editor, *Advances in Cryptology–EUROCRYPT'97*, pages 119–133. Springer-Verlag (LNCS 1233), Berlin (1997).

[90] S.T. Walker. Software key escrow: A better solution for law enforcement's needs? In L.J. Hoffman, editor, *Building in Big Brother, The Cryptographic Policy Debate*, pages 174–179. Springer-Verlag, New York, 1995.

[91] S.T. Walker, S.B. Lipner, C.M. Ellison, and D.M. Balenson. Commercial key recovery. *Communications of the ACM*, **39(3)**:41–47, March 1996.

[92] The White House. Directive on public key encryption management, 1993.

[93] M.J. Wiener. Efficient DES key search - an update. *RSA Laboratories' CryptoBytes*, **3(2)**:6–8, Autumn 1997.

[94] M. Willett. Features, attributes, characteristics, and traits (FACTs) of key recovery schemes/products. *Computers & Security*, **19(1)**:28–30, 2000.

[95] C. Williams and N. Zunic. Global interoperability for key recovery. *Computers & Security*, **19(1)**:48–55, 2000.

[96] N. Zunic. Organization considerations for retrieval of stored data via key recovery methods. *Computers & Security*, **19(1)**:82–85, 2000.