Technical Report

UCAM-CL-TR-780
ISSN 1476-2986

Number 780

UNIVERSITY OF
CAMBRIDGE

Computer Laboratory

Cryptography and evidence

Michael Roe

May 2010

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted
April 1997 by the author for the degree of Doctor of
Philosophy to the University of Cambridge, Clare College.

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Summary

The invention of public-key cryptography led to the notion that cryptographically protected messages could be used as evidence to convince an impartial adjudicator that a disputed event had in fact occurred. Information stored in a computer is easily modified, and so records can be falsified or retrospectively modified. Cryptographic protection prevents modification, and it is hoped that this will make cryptographically protected data acceptable as evidence. This usage of cryptography to render an event undeniable has become known as non-repudiation. This dissertation is an enquiry into the fundamental limitations of this application of cryptography, and the disadvantages of the techniques which are currently in use. In the course of this investigation I consider the converse problem, of ensuring that an instance of communication between computer systems leaves behind no unequivocal evidence of its having taken place. Features of communications protocols that were seen as defects from the standpoint of non-repudiation can be seen as benefits from the standpoint of this converse problem, which I call "plausible deniability".

Acknowledgements

I would like to thank Peter Kirstein and Ben Bacarisse for managing the research projects which caused me to become interested in this area; Steve Kent for many interesting discussions about the problems of key certification; Russ Housley for suggesting the term “plausible deniability”; Roger Needham for being my supervisor; and Bruce Christianson for his advice on how to write a dissertation.

To my grandfather,
George A. Lear

Contents

Chapter 1

Introduction

1.1 Narratives of conflict

The study of computer security, by its very nature, is a study of conflict. The notion of a computer security mechanism presupposes the possibility of a conflict of goals between people or organisations, and the possibility that part of this conflict will take place within a computer system.

Some of these conflicts can be described in terms of “insiders” and “outsiders”. From the perspective of the “insiders”, the “outsiders” have no right or legitimate reason to access the computer system, and the security mechanisms built by the insiders are directed towards keeping the “outsiders” outside.

When systems are described in this way, the author of the description is usually sympathetic towards the cause of the insiders, and intends that the readers will feel the same way. The computer security genre has conventions for providing the reader with hints as to where their sympathies should lie (rather like the good guys wearing white hats and the bad guys wearing black hats in Westerns). Security protocols are often described in terms of the imaginary protagonists “Alice” and “Bob”, together with as many of their friends, family, and foes as are needed to tell the story [30, chapter 2]. It is the convention that the reader should feel sympathy for Alice and Bob, while regarding the protagonists with names later in the alphabet with some suspicion.

A typical account of a security protocol can be summarised as follows: Alice and Bob live in a lawless and dangerous place, and are threatened by a succession of natural hazards and incursions by the outsiders. Alice and Bob employ a variety of ingenious technical means to overcome these threats, and live happily ever after.

This dissertation is concerned with conflicts which cannot easily be narrated in these terms. For example, there can be conflicts between different users of a computer system; between the provider and the users of a service; and between the authors of a piece of software and its users. These conflicts cannot be viewed as insider/outsider conflicts; both of the conflicting parties have some form of legitimate access to the system in question.

In much of the technical discussion which will follow, there will be no a priori assumptions about the identity of the guilty party. It might be Alice; it might be Bob; it might be both of them colluding together or someone else entirely. In this state of mind, even security protocols which could have been modelled in insider/outsider terms are seen in a new light.

Although we can free ourselves of any prejudice with respect to the imaginary characters “Alice” and “Bob”, the act of analysis and inquiry is still not neutral. In seeking to understand the situation, the reader and writer are implicitly taking sides in the conflict. Some participants stand to gain by the situation being better understood, while others stand to lose from this. However, when the true state of affairs is unclear, it can be unclear who is benefitting from the lack of clarity.

1.2 Non-repudiation

It is often the case that people who do not entirely trust each other wish to participate in a joint activity which they see as being mutually beneficial. In such a situation, each participant would like to have some form of protection against possible malicious actions by the other participants. Human society has developed many such protection mechanisms, for example, the law of contract, and the law of tort.

This dissertation is not about new or alternative forms of human social organisation or dispute resolution. Rather, it is about some specific new problems which have been caused by the use of computer networks as intermediaries in interactions between people.

Cryptography can be used to provide several different forms of protection in the electronic world. Together, these new forms of protection go some way towards making computer-mediated interactions as safe as non-computerised ones.

In this dissertation I will be examining a particular type of protection, namely the ability to form binding agreements between individuals and to have fair arbitration of disputes concerning those agreements. This form of protection is a small part of the total problem of making the electronic world a “safe place”, but the mechanisms and infrastructure developed to help resolve disputes also play a major role in solving many of the other protection problems.

It will be my contention that a critical problem with digital communications (or rather, with digital records of digital communications) is that it is easy to make good forgeries. In particular, it is easy to falsify after the event records of what took place and what was agreed. In the face of total uncertainty about “what happened”, fair arbitration becomes impossible, as the adjudicator cannot reach a decision on rational grounds. In turn, this makes forms of social interaction which depend on the possibility of arbitration, such as contracts, no longer viable.

To help resolve these problems of a digital world, the computer security community has developed a security service which is known as “non-repudiation”. In the words of the ISO Non-repudiation framework [13], the goal of this service is to:

> “provide irrefutable evidence concerning the occurrence or non-occurrence of a disputed event or action.”

In discussing this service, I will make frequent mention of the notion of an unbiased and open-minded observer. The intent of the non-repudiation service is that such an unbiased and open-minded observer should be convinced by the evidence that the service provides. Of course, in reality observers can be far from unbiased and open minded; but it is unreasonable to expect any *technological* mechanism to do anything about that. What we expect from the technology is this: putting ourselves temporarily in the place of this mythical unbiased observer, we would like to be able to decide (in specific instances) what happened. If the technology causes us to be left without hope of reaching a conclusion rationally, then there is a serious problem.

In this dissertation, I will examine in detail the evidence that is provided by the non-repudiation service. I will pay particularly close attention to the gap between the service we would like to have (even though this may be impossible to achieve) and the service that is actually provided by specific technical mechanisms. Once we have seen the gap between expectation and reality, will the evidence still be convincing?

1.3 Plausible deniability

It is legitimate to question to what extent the non-repudiation service is actually desirable. Who wants to know whether the event took place, and why do they want to know? Who has control over which events are part of the official version of history? For the benefit of those who conclude that non-repudiation is sometimes undesirable, this dissertation explores the potential of a new security service, which will be termed "plausible deniability".

1.4 Focus

This work has been directed towards a particular application area: the role of the non-repudiation service in commercial transactions carried out over European data networks. This choice of application area has influenced the scope of this dissertation in the following ways:

- This dissertation is only about new problems which have been caused by computers and data networks. It is not about other arenas in which conflict can take place.
- This dissertation is about the use of networks for commercial purposes, as opposed to military or recreational use. This setting determines what the conflicting parties stand to lose or gain, what they might be prepared to do to each other, and the type of methods that can be used to resolve conflict. Having said that, it is worth making two points. Firstly, the boundary between commercial and military conflict is sometimes crossed when the sum of money involved is large enough. Secondly, systems designed to meet a commercial purpose are sometimes influenced by military objectives of other parties (e.g. key recovery schemes where the government demands that Intelligence agencies be given back-door access to cryptographic keys used to protect other people's commercial traffic).
- To be successful, a new technical mechanism must fit in with the pre-existing legal and cultural conventions of the society which uses it. In this dissertation, I will be assuming a context of English-style common law.

  In particular, much of the reasoning surrounding the non-repudiation service implicitly assumes that two people can form an agreement in private which creates a binding contract; that any "reliable" record of the agreement will suffice; and that the terms of the agreement do not need prior approval by a government official.

The problem of non-repudiation can be approached from several directions. It can be approached as a legal problem (how do the existing laws of specific countries stand with respect to the admissibility of computer data as evidence?) or as an anthropological problem (what means have different human societies used to resolve disputes, and what can we learn from this?) I have approached the problem of non-repudiation from the perspective of a computer scientist with an interest in the theoretical bases of computation and communication. Non-repudiation involves both of these: it is about convincing someone (*by communication*) that an event (*of communication*) took place, and part of the reason that the listener becomes convinced lies in the theory of *computation*, specifically, the belief that some things are very much harder to compute than others.

The rest of this dissertation is arranged as follows:

- Chapter 2 describes the technical background to this work.
- Chapter 3 outlines some of the benefits and drawbacks of public-key cryptography, the technique which is most commonly used to build non-repudiation protocols.
- Chapter 4 discusses the basic principles of non-repudiation: what non-repudiation is, and what we expect a non-repudiation protocol to do for us.
- Chapter 5 examines some protocols which are intended to provide non-repudiation. There are technical reasons why these protocols do not entirely succeed in achieving this goal. Some of these problems are fixable, but an entirely risk-free protocol remains elusive.
- Chapter 6 examines the converse problem: if non-repudiation is deemed positively undesirable in a particular situation, how do we go about ensuring that unwanted evidence will not be available? As a demonstration of the concept, this chapter also describes some cryptographic protocols which are designed to achieve this.
- Chapter 7 describes aspects of non-repudiation which are internal to computer systems, in contrast to the external communications aspects which were described in chapter 5. While chapter 5 is mainly about the management of public keys, this chapter is mainly about the management of private keys.
- Chapter 8 presents some conclusions.

Chapter 2

Background

2.1 Notation

Digital signature

If CK_A is a public key used for confidentiality, $CK_A(m)$ will denote a message m encrypted using CK_A. An asymmetric key pair used for encipherment will be denoted by (CK_A^{-1}, CK_A), where CK_A^{-1} is the private component.

If IK_A^{-1} is a private key used for integrity, $IK_A^{-1}(m)$ will denote a digital signature for a message m computed using IK_A^{-1}. An asymmetric key pair used for digital signature will be denoted by (IK_A^{-1}, IK_A), where IK_A^{-1} is the private component.

That is, cryptographic keys are denoted by the mathematical function that is computed when the key is applied to data. The other half of an asymmetric key pair is the inverse function:

$$IK(IK^{-1}(x)) = x$$

$$CK^{-1}(CK(x)) = x$$

This notation is similar to that of Needham[23, 22], but differs in that it distinguishes encipherment from digital signature. In Needham [22, page 4] it is assumed that key pairs are always usable for both purposes, whereas in this dissertation there is no assumption that the operations of encryption (for confidentiality) or signature (for integrity) are in any way related.

When I use the notation $IK_A^{-1}(m)$, the reader should interpret it as denoting whatever procedures are appropriate for forming digital signatures with a particular cryptographic algorithm. Typically, this will be reducing m in length with a collision-free hash function, padding the result to a fixed length in some agreed way, performing a "low-level" signature operation on the padded hash, and then finally concatenating the original message with the signature. This has the consequence that anyone can recover m from $IK_A^{-1}(m)$, even if they don't know IK_A.

Similarly, the notation $CK_A(m)$ should be interpreted as denoting whatever procedures are deemed appropriate for encrypting a message under a public key. Typically, this will involve padding the message to a fixed length with random data [28, 1] and then applying a "low-level" public key encryption operation. The message m that is encrypted in this way will frequently contain a symmetric key which is to be used to encrypt other data. It should be taken as read that such symmetric keys are “well chosen”, that is sufficiently random and having whatever properties are deemed desirable for use with the symmetric key algorithm.

Symmetric encipherment

Symmetric keys will also be represented by functions. $CK_{AB}(m)$ will denote the message m enciphered using a symmetric key shared between A and B. Some symmetric keys are shared by all members of a group; CK_{AX} will denote a symmetric key used for enciphering messages between A and the members of a group.

The notation $m \oplus k$ will denote the bit-wise exclusive-or of a message m with key material k. When this notation is used, k will typically be a one-time pad.

Message exchanges

The messages exchanged during a run of a cryptographic protocol will be described using the “arrow” notation. A protocol step in which A sends the message m to B is written as follows:

$$A \rightarrow B : m$$

2.2 The integrity and confidentiality services

In this dissertation, I will frequently refer to two basic notions, namely *integrity* and *confidentiality.* The OSI Security Architecture [10] defines these terms as follows:

> **confidentiality:** The property that information is not made available or disclosed to unauthorised individuals, entities or processes.
>
> **data integrity:** The property that data has not been altered or destroyed in an unauthorised manner.

To paraphrase, *confidentiality* is concerned with keeping secrets secret, while *data integrity* is concerned with preventing the forgery, corruption, falsification or destruction of digital data.

It is worth noting that in the OSI definitions, confidentiality protects *information* (i.e. facts about the world) while integrity protects *data* (i.e. particular symbolic representations of those facts). This difference in the definitions is quite deliberate, and reflects a fundamental difference between the two properties. To keep some information secret, it is necessary to protect everything which contains an expression of that information, or from which that information can be derived. To provide data integrity, it is sufficient to obtain a copy of the data which is known to be good. The possibility that other representations might exist (even corrupt ones) does not harm integrity, but is disastrous for confidentiality.

2.2.1 Separation between integrity and confidentiality

From the point of view of their definitions, the notions of integrity and confidentiality are quite distinct. Someone who needs one of these services does not necessarily need the other. There is certainly no reason why we must necessarily use the same technological means to provide both these services.

However, confusion between these services can arise because the same technique (encryption) can be used to provide both services. For example, in systems derived from the Needham Schroeder protocol [24] (such as Kerberos [16]) the same encryption operation is applied to a sequence of several data items: some of these items are being encrypted to keep them secret, while others are being encrypted to protect them from modification.

In this work, I will try to maintain a clear distinction between the use of cryptography to provide integrity, and the use of cryptography to provide confidentiality. There are several reasons why it is important to maintain this distinction:

- The choice of cryptographic algorithm is influenced by the service it is to be used for. Some cryptographic algorithms are very good for confidentiality, but very poor for integrity (e.g. one-time pads). Similarly, some cryptographic algorithms are good for integrity but don't provide confidentiality at all (e.g. message authentication codes). In order to choose the right algorithm for the job, it is necessary to know why the algorithm is being used.

- Keys which are used for confidentiality often need to be managed in different ways from keys which are used for integrity. If a key used for confidentiality is revealed, this retrospectively destroys the confidentiality property for messages that were sent in the past (i.e. an attacker who has saved a copy of those messages will become able to read them). The same thing doesn't hold for integrity. Once the legitimate users of a key have decided to change over to using a new key, the old key is of no use to an attacker: the legitimate users will not be fooled by a forgery made using the old key, because they know that that key is no longer valid.

  As a result of this, the procedures used for the generation, storage, transmission and destruction of confidentiality keys may be very different from the procedures used for integrity keys, even if the same cryptographic algorithm is used for both services.

- The *authorisation* policies may be different for integrity and confidentiality. That is, the set of people who are permitted to read an item of data may be different from the set of people who are authorised to modify it.

  Clearly, the access control policy for cryptographic keys ought to be consistent with the access control policy for the data those keys protect. It makes no sense to decide that an item of data must be kept secret, and then to let everyone have access to keys that enable them to obtain that secret data. Similarly, it makes no sense to decide that an item of data needs integrity protection, and then to give everyone access to keys that enable them to modify that data.

  Should insiders such as systems administration staff or programmers have access to the keys used to protect other people's data? This is a policy question that needs to be decided for each application, and the answer to this question may depend on whether we are talking about keys used for integrity or keys used for confidentiality. If the answer is different in the two cases, this can lead us to use different key distribution methods for integrity keys versus confidentiality keys, in order to provide different levels of protection against insider attacks. For example, there may be a need to be able to recover a patient's medical records after the death or retirement of the doctor treating them, but there is no need to be able to retrospectively falsify those records.

2.2.2 Connections between integrity and confidentiality

Although the notions of confidentiality and integrity are quite distinct, the means for providing one service sometimes relies upon the other service.

Cryptographic integrity mechanisms rely on cryptographic keys, and in particular they rely on some of those keys being kept secret from unauthorised entities. A system which uses cryptography to provide integrity therefore needs some confidentiality as well, just to protect the keys.

Of course, cryptography isn't the only way to provide integrity. For example, physical measures that keep unauthorised persons physically isolated from a system can provide integrity, and they do so in a way that does not in any way depend upon confidentiality.

However, in this work I'm interested in providing security in public, international communications networks. In this situation, physical protection measures are infeasible; cryptography seems to be the only viable solution. This is in some respects unfortunate: it means that the systems I wish to construct must necessarily have a small component (key storage) for which confidentiality must be provided, even if confidentiality was otherwise unnecessary. However, the type of confidentiality needed to protect keys is less general than that needed to protect arbitrary user data, and hence may be easier to achieve, As we will see in chapter 7, it is sufficient to be able to maintain the confidentiality of keys in storage: systems can be built that never need to preserve the confidentiality of keys that are transmitted between systems. Furthermore, for the provision of integrity it is sufficient to have short-term secrets, that is secrets which become known after a while. Protocols can be constructed so that the integrity of data is maintained even after keys which previously protected that data have become publicly known. It is also clearly desirable to provide long-term non-repudiation using keys whose confidentiality is short-lived; as we will see in chapter 7, this can be done.

Conversely, cryptographic confidentiality mechanisms need integrity mechanisms to protect their keys. If an attacker can somehow change the keys which are used for encryption, by subverting the integrity property, then they can also break the confidentiality property, by substituting a key whose value they have chosen.

There are more subtle ways in which a failure of the integrity property can also destroy the confidentiality property, examples of which are given below. For this reason, one might take the view that any system which provides confidentiality should also provide integrity, because the user of the system probably needs integrity even if they don't realise that they need it. This line of argument is the reason why Internet Privacy Enhanced Mail does not provide a confidentiality-only mode, and also leads to an argument that the Diffie-Hellman type public-key systems are preferable to RSA type public key systems. I will elaborate on this later point in chapter 3.

These connections between the two services mean that we must take great care when designing systems which are to provide a very strong form of one service, but only a weak form of the other. An attacker may be able to exploit these dependencies between the services to first break the weak service, and then use this a means to carry out an attack on the "strong" service.

2.2.3 Some attacks

The difference between integrity and confidentiality is exemplified by the following attacks on some security mechanisms that become possible when those mechanisms are used inappropriately. (i.e. when a mechanism that provides one service is used but a mechanism that provides another service was really needed). These attacks are for the most part obvious and well-known. They are important because they are used as building blocks in some more complex constructions that will discussed later on.

MACs do not provide confidentiality

Message authentication codes (MACs) can be used to provide integrity. The MAC is a function of the data to be protected and the key. The typical usage of a MAC is when the data is sent unencrypted, followed by the MAC which protects it. Without knowing the key, an attacker cannot compute a combination of some data and a MAC which will appear valid to the recipient; in this way, MACs provide integrity. However, an attacker who is just interested in obtaining the data (i.e. violating confidentiality) can simply intercept the unencrypted data and completely ignore the MAC.

One-time pads do not provide integrity

One-time pads can be used to provide confidentiality. The ciphertext consists of the plaintext exclusive-or'd with the key (which in this case is known as the "pad", because historically it was printed on note pads).

Provided that the key material is truly random and uniformly distributed, and is only used once, then this system is unbreakable in the sense described by Shannon in the 1940's [31]. It is critically important that the pad is only used once, i.e. a key is never re-used to encrypt a second message. This weakness of the one-time pad system makes it impractical in many applications; however, this weakness is not the one that is relevant to the discussion of integrity versus confidentiality.

Even if all proper precautions are taken with a one-time pad (it is only used once, the key is kept physically protected where the attacker can't get at it etc.) it fails to provide integrity. If the attacker does not know what message was sent, they cannot determine this by examining the ciphertext. However if the attacker knows what message was sent and is merely interested in substituting a different message (i.e. violating integrity), then they can do this as follows:

$$M' \oplus K = (M \oplus K) \oplus (M' \oplus M)$$

If the attacker knows the message that was really sent (M), then they can obtain the enciphered message $M \oplus K$ by wiretapping and combine it with $M' \oplus M$ to produce a new message which will appear valid to the recipient. As one-time pads are only used once, the attacker must arrange that the legitimate message is lost in transit and only the substitute message ever reaches the recipient.

Re-use of keys with multiple mechanisms

If the same cryptographic key is used with several different cryptographic mechanisms, then this can sometimes make an attack possible, even if the mechanisms are secure when considered individually. This is because the attacker can use information learned by observing the key being used for one function to carry out an attack on the other function using the same key. This commonly occurs when the same key is used for both confidentiality and integrity. An example of this situation has been described by Stubblebine [33].

The existence of this type of attack provides an additional motivation for distinguishing integrity and confidentiality keys: we need to avoid the possibility of this type of attack, so we need to take care never to use the same key with both sorts of mechanism.

Weak integrity weakens strong confidentiality

Suppose that Alice sends a confidentiality protected message to Bob. If Carol can convince Bob that the message really came from Carol, she may be able to persuade Bob to reveal to her the content of the message. In this way, a weakness in the integrity mechanism can be turned into a failure of confidentiality.

The one-step authentication protocol defined in X.509 is vulnerable to this type of attack, because in that protocol encryption is performed before signature (rather than vice-versa):

$$A \rightarrow B : IK_A^{-1}(t_A, r_A, m_1, CK_B(m_2))$$

C can take a copy of this message and replay the encrypted and unencrypted portions with a new signature:

$$C \rightarrow B : IK_C^{-1}(t_C, r_C, m_1, CK_B(m_2))$$

This convinces B that m_1 and m_2 came from C, and B might be subsequently tricked into revealing information about the confidential message m_2. A more realistic version of the same attack occurs in store and forward messaging systems based on this protocol (e.g. X.400). If Bell-LaPadula style mandatory access control is in effect, and m_1 contains the classification level of the data in m_2, then C can trick B into believing that classified data is unclassified; B might then be tricked into releasing the data to persons who don't have a high enough security clearance (e.g. C).

A fix to this protocol is to include all relevant data (e.g. the security label of the data, or the name of B) in the enciphered message. X.400 implemented with the right selection of options incorporates this fix; X.400 with the wrong selection of options is still vulnerable to the attack.

2.3 Kerberos and the Needham-Schroeder protocol

In subsequent chapters I will illustrate several points by making a comparison with Kerberos [16] and the Needham-Schroeder protocol [24] on which Kerberos is based. However, the objectives of this work are significantly different from the objectives of Kerberos. Kerberos is concerned with *authentication* (that is, it provides communicating computer programs with information about who or what they are communicating with) and confidentiality. In contrast, I am primarily concerned with *non-repudiation*; enabling an independent third party to establish what happened after the event.

Non-repudiation often includes establishing the identity of the some of the entities involved: knowing *who* was involved is frequently a vital part of knowing *what* happened. In this respect, non-repudiation has some similarities with authentication. However, the methods used to provide these two services differ in their details. Establishing the identity of someone with whom you are currently communicating is a different problem from establishing the identity of someone who participated in an event which occurred in the past, and in which you were not directly involved.

The main technical reason why Kerberos cannot be used to provide non-repudiation lies in the way that it uses symmetric-key cryptography. When Kerberos is used to protect the communications between two entities, the two entities *share* a cryptographic key which is used both to compute the message authentication code on data before it is sent, and to verify the message authentication code on data after it is received. As both entities know this session key, they can use it to forge messages which appear to come from the other. If a dispute arises, the participants' own records of the message authentication codes isn't enough to tell which messages really were part of the exchange and which were falsified afterwards.

2.4 Digital signatures

The idea of using cryptography to facilitate the resolution of disputes first arose in the context of public-key cryptography [8]. Indeed, later publications (such as X.509 [12] and the OSI Security Architecture [10]) began to regard the ability to resolve disputes as being synonymous with the use of public-key cryptosystems.

Later on in this chapter, I will describe some systems which are currently in use and which make use of public-key cryptography. Before I describe these real systems, I will review how the original papers on public key imagined it being used, and why it was believed that public key cryptography provided non-repudiation. This review is to some extent an unfair caricature, as I will deliberately draw attention to problem areas that the authors of the early papers either ignored or considered to be someone else's problem.

The traditional account runs as follows:

Everyone generates their own private key using a computer which they themselves have checked to be functioning correctly, using software which they have written personally, and which therefore contains no programming errors or malicious code. This computer has access to a physical source of random numbers which absolutely cannot be predicted by anyone else (e.g. a Geiger counter next to a radioactive source). This computer is also physically secure (in a locked room, electromagnetically shielded to prevent the value of the key being revealed by electrical interference and so on).

Everyone takes their public key to a publisher, who prints a physical book, rather like a telephone directory, containing everyone's name and public key. Everyone checks their own entry in their copy of the book and their friend's copy, and raises a big fuss if there's an error. The printing process makes it extraordinarily expensive to produce one-off copies of the book with selected entries altered, so everyone is sure that every copy of the book says exactly the same thing.

To generate evidence of having entered into an agreement, a user performs a computation on their own computer using their private key and the text of the agreement. The software with with they do this is, of course, written by themselves and entirely free from errors.

A recipient of this "digital signature" can check it using their own software and a copy of the signer's public key from the phone book. Should a dispute arise later, the adjudicator can also check this digital signature using their own software and the public key from their own copy of the phone book. This will of course result in the same answer.

From this, the adjudicator becomes absolutely convinced that the signer must have intended to enter into the agreement, because that is the only conceivable way in which the adjudicator could have been presented with a binary value which has the correct algebraic properties.

As we shall see, real systems which use public key cryptography differ from this picture in almost every respect, except that they use public key cryptography. The belief that public key cryptography is synonymous with the ability to resolve disputes is based on the assumption that these differences don't matter.

2.5 PGP

2.5.1 The motivation for PGP

"Pretty Good Privacy" (PGP) is an e-mail encryption program written by Phil Zimmerman [40]. PGP is not intended to be used for non-repudiation, but it does use public-key cryptography for authentication. This makes it an interesting example for comparison when discussing the differences between authentication and non-repudiation. Before discussing the technical details of what PGP does, it is worth considering the implications of the word "privacy" in its name. The OSI Security Architecture [10] defines privacy as follows:

> "**privacy:** The right of individuals to control or influence what information related to them may be collected and stored and by whom and to whom that information may be disclosed."

Privacy is not the same as confidentiality (keeping secrets secret). There are many uses of confidentiality services that are not connected with protecting the privacy of individuals. For example, cryptographic confidentiality is often used to protect commercial secrets (plans for takeover bids, new products in development and so on); this is not personal privacy. Equally, personal privacy has many aspects beyond just encrypting e-mail. From a privacy standpoint, legislation such as the Data Protection Act (which regulates what personal information may be stored in databases) is probably more important than the use of cryptography.

The "privacy" in PGP can be regarded as a summary of Phil Zimmerman's motivation for creating PGP, rather than a description of what PGP actually does. It certainly does not, on its own, provide privacy in the sense that was described above. However, it can be argued that the e-mail encryption service it does provide is an important ingredient in providing privacy. If personal information (e.g. in personal e-mail between individuals) is to be transmitted over computer networks then in order to have privacy this information must be protected. The underlying premise of PGP is that in our society people need to communicate with each other over long distances, that this communication frequently involves personal information related to the communicating parties, and so cryptographic confidentiality is needed.

The privacy motivation for PGP helps explain why it does not set out to provide non-repudiation (although it does provide authentication and integrity). The essence of the non-repudiation service is that third parties might be made aware of the events the service protects, and indeed will be given relatively reliable evidence about those events. On the other hand, privacy is at least in part about preventing third parties from learning about events which don't concern them. If messages are intended to be kept private, there is no compelling reason for making them suitable for showing to someone else.

2.5.2 Reasons for the success of PGP

PGP is very widely used for personal e-mail on the Internet. To have achieved this level of success, PGP must have had some advantage which earlier e-mail encryption schemes (e.g. Privacy Enhanced Mail, which will be discussed later) lacked. The following features of PGP may account for its popularity: it is actually available (a real product and not just a theoretical idea); it's free; and it does not require any additional infrastructure in order to work.

The last point is particularly important. Other e-mail encryption schemes (such as PEM) need a "Public Key Infrastructure" in order to work. In order to use them, you need to make extensive use of supporting services which are supposed to be provided by other people or organisations. Of course, if no-one is actually offering to provide these supporting services, then these schemes can't get off the ground. PGP's great advantage is that all you need to use it is the software; you don't need to buy special services from anyone in order to get the software to work.

If PGP manages without a Public Key Infrastructure, why do the other schemes need one? The answer is that schemes such as PEM are actually solving a different problem from PGP, even though they might initially appear to be similar. As PGP shows, protecting personal e-mail between small groups of friends does not require an extensive infrastructure to make it work.

In place of a public key infrastructure, PGP supports what Zimmerman terms a "Web of Trust" model. Under the web of trust model, the primary means of obtaining cryptographic keys is by direct physical exchange between people. If Alice and Bob meet in person, and Alice gives Bob her business card with her PGP public key printed on it, then Bob knows for certain that he has the cryptographic key which Alice intended to give him. There is no possibility of confusion being caused by the existence of multiple people called "Alice"; Bob knows for certain that the key he has is suitable for communicating with the person he met. (There is a minor side issue that Alice might give Bob someone else's key as part of a complicated way of defrauding Bob, but I will ignore this for the moment, and return to it later in chapter 7).

The second means of obtaining keys with the web of trust model is indirect: if Alice and Bob have previously exchanged keys, then Alice can give Bob a copy of Carol's key over the protected channel creating using the physically exchanged keys. It is made quite explicit that this is only an acceptable way for Bob to obtain Carol's key if Bob "trusts" Alice in this respect: after all, Alice might lie or not be competent. In the web of trust model, whether Bob trusts Alice in this respect is left entirely to Bob's discretion. Neither Phil Zimmerman (the author of PGP) or the PGP program itself make any statement as to whether or not Alice should be trusted; after all, they have no personal knowledge of Alice and are in no position to make statements about her.

In addition to not needing any special new externally-provided services, the web of trust model has the added advantage that there is no need for all of its users to agree on who is trusted to provide keys. Bob's decision to trust Alice to provide a key for Carol is completely independent of anyone else's decision to trust Alice. This means that PGP can easily be used by lots of different groups of people who have different ideas about who should be trusted. In contrast PEM assumes the existence of organisations that every single user of PEM, everywhere in the world, agrees are trustworthy. The universally trusted root authority postulated by PEM is in practice almost impossible to set up. To be more exact, it is easy for someone to declare themselves to be the universally trusted authority, but it is much harder to get everyone else to accept their authority.

PGP avoids this problem by not having a root authority. This meant that people could actually use PGP while prospective Internet PEM users were still arguing about who was going to be the single universally trusted authority.

2.5.3 Limitations of PGP

Unattended operation

PGP succeeds by taking the most conceptually difficult part of its operation (deciding who should be trusted and who shouldn't), and making it the responsibility of the human being who uses PGP rather than the program itself. This makes life very much easier for the implementor of the PGP program; it is also in some sense the right thing to do, as the program itself lacks any knowledge which it could use to make a rational decision to trust someone or not. However, this approach also has a drawback: it places an additional burden on the user, and more significantly, it requires that a human being actually be present when the PGP program is run. This latter point is not a big problem when PGP is used to secure personal e-mail; the human being has to be there to understand the contents of the message, so they might as well help out with the security operations while they're at it. The requirement for a human to be present becomes more burdensome when it is desired to extend the application area of PGP to include things such as electronic commerce or remote access to databases. In these new application areas, it makes sense to have a computer running completely unattended and acting on messages received. It is unacceptably expensive to require a computer operator to be present just to support the security, if they wouldn't otherwise be needed. In addition, when PGP is used for the official business of an organisation (as opposed to personal use), there is the issue that the interests of the organisation and the interests of the person employed to run the program might not be entirely the same, and as a result the operator might deliberately make a bad decision.

Use within large user groups

While it works acceptably with small groups of users, PGP becomes harder to manage when the size of the communicating groups of users become too large. Specifically, how does a user decide who they should trust to provide them with another user's public key? This needs to be someone who is both trustworthy and in a position to know the required key. In a small user group, it is easy to identify someone who satisfies both criteria. In a large, geographically dispersed community this is much more difficult to do.

One possible rebuttal to this as a criticism of PGP is to ask why anyone would ever want to communicate securely with a complete stranger. The argument goes that for communication to be desired, there must exist some form of social relationship between the two people who wish to communicate, and that this social relationship usually provides a direct or indirect path by which cryptographic keys could be exchanged. The totally *ad hoc* approach of PGP only becomes unviable if the relationship between communicating parties is highly obscure and indirect.

Recovery from compromise of keys

With most systems based on public-key cryptography, the security of the system is completely dependent on the security of user's private keys. If an attacker somehow manages to gain access to one or more of these keys, then the security of the system can be impacted.

In an attempt to prevent this unfortunate eventuality, considerable effort is usually made to protect private keys and stop attackers obtaining them. Nevertheless, it is almost inevitable that some cryptographic keys will become compromised, either from carelessness of users, or exceptionally strenuous attempts to break the system on the part of an attacker. The question arises as to whether it is possible to do anything to ameliorate this unfortunate situation if it should arise.

PGP provides a mechanism for dealing with keys which are compromised in this way. The possessor of a key can use that key to sign a message which states that the key has been compromised, and should no longer be used.

Of course, if the key really has been compromised, both the legitimate owner of the key and the attacker both have a copy of the key, and so either can issue a revocation notice. Although in this situation we can't distinguish who issued the revocation, it doesn't matter: if either the legitimate owner of the key complains that it's been compromised, or an attacker demonstrates that they have compromised it, the prudent thing to do is to stop using that key.

No revocation mechanism can be entirely perfect, for reasons that will be examined later on in this dissertation. However, it is clear that it is PGP's approach to revocation can improved upon:

- The PGP revocation message will only be effective if potential users of the compromised key see it. PGP provides no mechanism to guarantee (or even make it likely) that revocation messages will be seen by the people who ought to see them.
- The legitimate user needs a copy of their key to revoke it. If the attacker destroys the legitimate user's copy of their own key, rather than just copying it, then the legitimate user can do nothing. Note that this difficult situation occurs if the physical media storing the key are lost or stolen, e.g. if a laptop computer storing the key is left on a bus.
- In the PGP scheme, revocation of a key requires the co-operation of the holder of the private key. If the only holder of the private key all along has been an attacker, then they may not co-operate. For example, suppose that an attacker generates their own key-pair, impersonates "Alice" and tricks someone ("Carol", say) into signing a statement that the public key is Alice's key. What can Alice or Carol do if they discover this deception? With the PGP scheme, there isn't much that they can do.

The last two of these problems can be fixed by requiring the user to sign a revocation certificate for their key in advance, and to deposit a copy of this with whoever certifies their key. In this way, signers of certificates can revoke keys which they have signed certificates for (by releasing their copy of the revocation certificate), and there will be a back-up copy of the revocation certificate even if the user loses everything. The disadvantage of such pre-signed revocation certificates is that they make it very hard to tell exactly when revocation occurred. For integrity and confidentiality, the time of revocation doesn't matter much. However, for non-repudiation it is critical. I will return to this issue in chapter 5.

2.6 X.509

The International Standard known as X.509 [12] defines a format for what it calls a *certificate*. The X.509 certificate is a digital representation of a declaration by one entity (Alice, say) that another entity (Bob, say) uses a particular value for his public key.

The purpose of certificates is to provide a means of obtaining someone's public key without having to meet them in person; instead, just ask an an intermediary with whom you can already communicate securely. PGP (which was described earlier) also uses certificates. The certificate format used by PGP differs in some minor details from that laid down by the X.509 standard. These differences in format are of little consequence; what really matters is how certificates are used, not how the information in them is laid out.

2.6.1 What X.509 doesn't say

X.509 is part of the "Open Systems Interconnection" (OSI) series of standards, which set out to prescribe how data is represented when it is exchanged between systems, without dictating what computer systems should be used for or how they should be run. In keeping with this general OSI philosophy, X.509 says relatively little about what you should do with certificates. It describes the format, but it considers such issues such as which certificates should be believed and which precautions it is sensible to take before issuing a certificate to be outside its scope.

These issues are, however, important. Before anyone can actually deploy a system that uses X.509, they have to make a decision on these points. PGP avoids some of the difficult problems by having the program ask the user what it should do. X.509 avoids even more, and leaves the difficult problems to the discretion of the programmer who will eventually write the program.

2.6.2 X.509 and non-repudiation

While the data structures and cryptographic processes described in X.509 are quite similar to those by PGP, X.509 takes a different approach to them in a number of respects.

In the previous section I explained why non-repudiation is not part of the central problem PGP is trying to solve. X.509, on the other hand, is concerned with non-repudiation. Or rather, the explanatory material in the appendices to X.509 states that "The digital signature mechanism supports the data integrity service and also supports the non-repudiation service". The main text of X.509 doesn't mention non-repudiation, and doesn't say anything about resolving disputes.

The reader of X.509 is thus left with a problem. From the explanatory appendices of X.509 it appears that X.509 is intended to provide non-repudiation, and enable disputes to be resolved. On the other hand, the main text doesn't explain how you use X.509 to do this.

2.6.3 Certification authorities

The second difference between X.509 and PGP is the notion of a *certification authority* (CA). In PGP, it is explicit that any user of PGP can tell any other user about someone's key, but they might not be believed. X.509 introduces certification authorities, which are people or organisations which are in the business of exchanging keys between users. This could be regarded as a natural development of PGP's key distribution process; initially, keys are exchanged on an *ad hoc* basis, and then someone makes a business out of providing this service on a large scale and on a regular basis.

However, by adopting a terminology which distinguishes certification authorities from normal users, X.509 raises the following two questions:

- Who is allowed to be a certification authority?
- How do you know that someone is a certification authority?

With PGP, the answer to these questions is clear: anyone can be a certification authority, and if you trust them to provide you with good keys, then they're a certification authority for you.

X.509, in its characteristic style, fails to answer these questions. However, as we we see later, some systems based on X.509 answer these questions in a way which is very different from PGP.

2.6.4 Certification paths

X.509 also introduced the notion of a *certification path.* With a certification path, a user discovers another user's key by examining a chain of statements made by several intermediaries. The user starts off by knowing the public key of intermediary C_1, and uses this to verify C_1's digital signature on a statement that C_2's public key is IK_2, and then uses IK_2 to verify C_2's digital signature on statement that C_3's public key is IK_3, and so on.

Later on, I will return to the question of whether this really works or not. It seems to be alright provided that all of the intermediaries are telling the truth; but on what basis does the user decide to believe them? How does the user know that one of these C_i isn't a pseudonym for an attacker?

The PGP answer to this question would be that a user determines whether to trust intermediaries based on his personal knowledge of them. But, if the user had direct personal knowledge of each of these intermediaries, why would she need to go through a long chain of other intermediaries to reach them? The whole point of X.509's long certification paths is that it enables the user to reach intermediaries with which she has no direct experience: but is reaching them any use if you have no means to assess their trustworthiness?

2.6.5 Revocation lists

X.509 defines a *revocation list* mechanism for dealing with compromised private keys. At regular intervals, certification authorities are expected to issue a digitally signed list of all the certificates that were issued by them, but which have been subsequently revoked.

Recall that with PGP revocation messages, there is a very real risk that a user might miss an important revocation message. X.509 goes a long way towards solving this problem by having a single revocation list for all the certificates issued by a CA. If a user has this list, they know that it is complete (or rather, that it was complete at the time of issue) and there is no possibility of there being other relevant revocation information that the user has accidentally missed. The one remaining problem is one of timeliness: how does a user know that the revocation list they have obtained is the most recent one? Perhaps there is another more recent list which revoked the certificate which the user is interested in?

X.509 revocation lists contain a date of issue, so the problem of timeliness can be partly solved if the CA issues revocation lists at regular intervals. If a CA issues a new revocation list daily, and all users of its revocation lists know this, then when a user has a revocation list with the current day's date in it, they know it is the most current one.

There are at least two remaining problems:

- Revocation information may not propagate immediately. If revocation lists are issued daily, then in the worst case there is a period of 24 hours between the key being reported as compromised and users realising that the previous revocation list (which doesn't include the compromised key) is not the most recent one. In some applications, this delay can be disastrous; the attacker may be able to do a large amount of unrecoverable damage in the 24 hours between stealing the key and it being revoked.

- If revocation lists are issued at regular intervals, then the CA has to regularly issue new revocation lists even if no new certificates have been revoked. This incurs communication costs to transmit the new lists to the users that need them, and it may also involve significant staff costs. The usual design for a certification authority is to have all communication between the machine which stores the CA's key and the outside network under direct human supervision. This reduces the risk of the key being compromised by an attacker breaking into the machine which holds the CA's key, but if the link needs to be used at regular intervals to generate revocation lists, then the CA operator also needs to be there at regular intervals to supervise it, and this is expensive in staff costs. The fail-safe nature of revocation lists means that if the CA operator isn't there to supervise the link (e.g. due to holidays or illness) then all that CA's certificates will stop being accepted, as there isn't a up to date revocation list to support them, Attempting to reduce the effects of problem 1 by issuing revocation lists more frequently makes problem 2 worse, and vice-versa. Changing the revocation list format to allow short “no change” certificates (which include a cryptographic hash of the full revocation list) would reduce the bandwidth needed for propagating updates, but still leaves us with the need for the operator to supervise the link at regular intervals.

Finally, note that there is a significant difference between X.509 and PGP over who can revoke a certificate. In PGP, it’s the subject of the certificate while in X.509 it’s the issuer. In the X.509 scheme, a certification authority can revoke a certificate without the consent of the certificate subject; in PGP, a user can revoke their key without the consent of those have certified that key.

2.6.6 X.509 versions

Although X.509 has come to be regarded as a general-purpose authentication protocol, it was originally designed to protect a specific application which had some unusual characteristics. X.509 is part 8 of the series of International Standards which describe a *Directory Service*: a global distributed database which holds information about both people and computers. The Directory Service is effectively an on-line omnibus edition of all the world’s telephone directories (both the “White Pages” for people and “Yellow Pages” for services), with lots of additional information thrown in for good measure.

X.509 was originally designed to protect access to this database. As a result, X.509 takes it for granted that this database actually exists and is considered desirable. X.509 uses this database in two ways. Firstly, if the database has been created, then in the course of its creation everyone in the world will have been allocated a unique identifier which refers to their entry in the global database. This identifier is used as the user’s name in X.509 certificates. Secondly, X.509 proposes the use of this database to store X.509 certificates; if the database exists, it’s a natural place to store them.

Since its inception, X.509 has been used as the basis for authentication in many other applications, e.g. Privacy Enhanced Mail, which will be described next. Some of the other applications would otherwise have had no need for a globally co-ordinated naming scheme or a world-wide database. The infrastructure presupposed by X.509 can be troublesome and costly in these applications. It is troublesome because the administrative act of creating a global database of individuals raises serious concerns of policy; for example, it may well fall foul of the privacy legislation in many European countries. It is costly, because from a technical perspective the global database is a very complex system which has to be built and maintained.

A second consequence of the Directory Service origin of X.509 was that X.509 designers did not consider to be within their remit the provision of features that weren’t needed by the Directory Service, even if those features were needed by other applications. For example, the Directory Service as defined in X.500 does not use any cryptographic confidentiality services, and hence version 1 of X.509 did not contain any features which are specific to confidentiality.

The situation changed slightly with revision 3 of X.509. This revision defines several extensions to the original protocol; some of these were of general utility, while others were oriented towards applications other the Directory Service. In particular, revision 3 of X.509 recognises that integrity keys and confidentiality keys may be different.

2.7 Internet Privacy Enhanced Mail

Internet Privacy Enhanced Mail [17, 15] was a draft standard for cryptographically-protected e-mail that was developed by the Internet Engineering Task Force. The development of PEM was abandoned by the IETF, and it was never formally approved as a standard. However, several vendors developed products based on the draft standard, and the PEM experience provided some useful lessons.

PEM is primarily concerned with providing security for people who are acting as agents of the organisation which employs them. This is illustrated by the following extract from RFC 1422:

> "Initially, we expect the majority of users will be registered via organizational affiliation, consistent with current practices for how most user mailboxes are provided." [15]

That is, the design of PEM is based on the belief that most e-mail is sent by people who are at work, using their employer's computers (or by students, using their university's computers), and hence that most cryptographically protected e-mail will be sent by people in those categories. The vast increase in the number of people accessing the Internet from home rather than at work occurred after the design of PEM.

In comparison, PGP was more oriented towards a home user (or at least, a user who is sending personal e-mail rather than doing their job of work). This difference between PEM and PGP is reflected in their naming structures. In PGP, names can contain anything but by convention are usually just the name of the person (e.g. "Michael Roe"). In PEM, names usually contain the name of an organisation and the organisation's country of registration in addition to the personal name of the individual. PEM treats the name of the organisation as being extremely important and intimately connected with the way security is provided; PGP considers it to be irrelevant.

In view of this, what PEM is providing is not really "privacy", even though the word "privacy" occurs in its name. Protecting the confidentiality of a company's internal workings as its employees exchange memos electronically is not the same as *personal* privacy. In addition, PEM is at least as much concerned with authentication and non-repudiation as it is with confidentiality: again, these are not privacy concerns.

2.7.1 Revocation lists

Although PEM adopted the certificate format from X.509, it defined a revocation list format which differed from that of X.509. The PEM revocation list format is more compact, and adds a *nextUpdate* field. The more compact format reflected a concern that revocation lists would become large, and that the cost of transmitting them would become considerable. The *nextUpdate* field indicates the date on which the CA intends to issue a new revocation list. The addition of this field makes it much easier to detect that an old revocation list is not the most up to date one: if the current date is later than the date given in the next update field, then there ought to be a more recent revocation list available. With the X.509 format, it is hard to be sure that a revocation list isn't up to date even if it is very old, because there is the possibility that it comes from a CA that only issues revocation lists infrequently.

Both of these changes were incorporated into the 1993 revision of X.509.

2.8 NIST/MITRE Public Key Initiative

The NIST Public Key Infrastructure study [2] investigated the practicalities of using X.509-style certificate based key management within the U.S. federal government. The main contributions of this study were that it proposed a certification structure which the authors of the report considered to be suitable for U.S. government use, and it made an attempt to estimate the monetary cost of using the public key technology on a government-wide basis.

The NIST report recognises that a very significant part of the total cost of the system is due to the certificate revocation mechanism used to guard against key compromise:

> "The PKI's yearly running expenses derive mainly from the expense of transmitting CRLs from the CAs. For example, the yearly cost is estimated at between $524M and $936M. All this except about $220M are CRL communications costs, which are charged at about 2 cents per kilobyte." [2, chapter 11]

My own experience with the "PASSWORD" European pilot project bears out this claim that CRLs represent a significant part of the resource cost of the public key infrastructure.

The CRL mechanism is frequently only considered as an afterthought in discussions of public key mechanisms. In view of its major contribution to the total cost, it deserves greater attention. I shall return to the CRL mechanism, and its implications both for cost and the effectiveness of the non-repudiation service in chapter 5.2.

2.9 The public key infrastructure

In the course of this chapter, I have introduced the elements of what is known as the "public key infrastructure". To summarize, these are:

- Certification Authorities
- Globally unique names
- A Directory Service for certificate retrieval
- An effective means for revoking keys (or certificates)
- An *agreement* among users about who is trusted for what

PGP manages to work without any of these things, but PGP is attempting to solve a slightly different problem from that addressed by some of the other protocols. In particular, PGP is not concerned with non-repudiation or with communications between people who do not know each other. Later chapters will explore the question of whether (in the context of a non-repudiation service) these components of the public key infrastructure become necessary, and whether there are other components that become necessary instead.

2.10 ISO non-repudiation framework

The ISO non-repudiation framework [13] has a special relationship to this work, as a large portion of the text in the ISO standard was written by me at the the same time as I was working on this dissertation. It is therefore not surprising that the technical approach taken in ISO 10181-4 has some elements in common with this dissertation.

The main advances made by the non-repudiation framework can be summarised as follows:

- ISO 10181-4 describes the non-repudiation service in terms of “evidence”, whereas earlier work (e.g. the OSI security architecture [10]) tended to use the term “proof”. In this dissertation, I have followed the evidence-based view of non-repudiation, and I elaborate on its consequences in chapter 4.
- ISO 10181-4 introduced the notion of an *evidence subject* as “an entity whose involvement in an event or action is established by evidence”. The point of this definition is that typical uses of the non-repudiation service are aimed at determining the involvement of a particular person (or legal person) in an event, and it is useful to be able to talk about this person without confusing them with the people who might have a dispute about the event.
- Earlier ISO standards such as ISO 7498-2 [10] and X.509 [12] imply that all non-repudiation protocols are based on public key cryptography, and that all digital signatures provide non-repudiation. ISO 10181-4 explicitly recognises that some symmetric key protocols also provide the ability to resolve some types of dispute. In this dissertation I also illustrate the converse: some public key protocols do not provide non-repudiation (e.g. some uses of PGP).

  From this I conclude that the non-repudiation property is a property of the system design as a whole, not of the particular cryptographic algorithms that are employed. It is worth remarking at this point that the symmetric key and public key non-repudiation protocols are not direct replacements for each other: each depends on different assumptions about its operating environment, and enables the resolution of a slightly different set of disputes.

ISO 10181-4 is heavily biased towards two types of non-repudiable event: the sending and receiving of messages by communicating OSI (N)-entities. In this dissertation, I take a view of the non-repudiation service which is more general than this, and arrive at a conception of the service which is not tied to the notions of sending and receiving.

Finally, the notion of plausible deniability developed in this dissertation has no counterpart in ISO 10181-4.

Chapter 3

Public key cryptography

Many of the mechanisms described in this thesis make use of public key cryptography. This chapter presents a brief overview of the relevant properties and methods of use of public key cryptography. In this chapter, I wish to draw special attention to two areas that are often neglected:

- These ways of using public key cryptography are based a large number of assumptions. Some of the assumptions aren't always true.
- These properties of public key cryptography aren't always desirable. While they are useful in some situations, they can be a threat in other circumstances,

3.1 Convert from integrity only to integrity plus confidentiality

Public key cryptography can be used to convert a communications channel which only has integrity protection into a communications channel which has both integrity and confidentiality protection.

Suppose that an integrity protected channel exists. By this, I mean that the following assumptions hold:

- There are two entities who wish to communicate with each other. To simplify the explanation, I will for the moment focus on the case where these two entities are people.
- There is a means of exchanging data between two computer programs, one running on behalf of one person, the other running on behalf of the other person.
- This means of communication guarantees that data exchanged between the two programs cannot be modified, deleted from, added to, re-ordered or otherwise changed by anyone except these two people (and programs acting on their behalf).

The last condition is often impossible to achieve, and the following weaker condition may be assumed instead:

- The means of communication either guarantees that the data has not been modified, or alerts the receiver of the data to the possibility that a modification might have occurred.

Furthermore, I shall assume (for the purposes of this section) that what has happened is that the channel has guaranteed the integrity of the data (rather than alerting the receiver to the possibility of error). That is, I am describing what happens when the protocol runs to completion, and not when it halts due to the detection of an error.

Finally, we sometimes also need to assume a fourth condition:

- The means of communication ensures that any data which is received (without an error condition being indicated) was intended to be received by the recipient (rather than some other entity).

Given the above assumptions, it is possible to state the first property of public cryptography as follows:

> Public key cryptography enables the two communicating entities to convert their integrity protected channel into a integrity and confidentiality protected channel. That is, they can in addition exchange enciphered data in such a way that no-one else is able to interpret it.

This is achieved by the following method:

- A creates a private key CK_A^{-1} and a public key CK_A.
- A gives the public key to B, using the integrity protected channel that is assumed to exist.
- B uses the public key to encipher messages sent to A.

  $B \rightarrow A : CK_A(m)$

The same procedure can be used with the roles of A and B exchanged to enable confidentiality protection in the reverse direction.

What this shows is that the invention of public key cryptography [6, 26] did not solve the problem of key distribution; it replaced one problem (creating a confidentiality protected channel) with another problem (creating an integrity protected channel). This new problem is also not trivial to solve.

3.2 Verifier needs no secrets

Public key cryptography can also be used to construct an integrity protected channel from another integrity protected channel (assuming that you have one to start with). The advantage of doing this is that the created channel may have a greater information carrying capacity, or may exist at a later point in time.

This is known as digital signature, and it works as follows:

- A creates a private key IK_A^{-1} and a public key IK_A.
- A gives the public key to B, using an integrity protected channel which is assumed to exist.
- Later, when A wishes to send a message m to B, A sends $IK_A^{-1}(m)$ to B over any communications channel, not necessarily an integrity protected one. As previously noted, this notation means concatenating m with a signed collision-free hash of m.

B can verify this using IK_A. Verification either succeeds or it doesn't, and therefore B either receives m (with a guarantee that it is unmodified) or an indication that modification may have occurred.

This is very nearly the same as the definition of a integrity protected channel that was given in the previous section. The difference becomes apparent if A ever wants to send B a second message (m_2). How can B tell if an attacker deletes m_1 but not m_2, or m_2 but not m_1? This can be fixed up by making a sequence number part of every message, or by having a challenge-response dialogue between A and B.

It can be seen that public key cryptography has been used to turn an unprotected channel into an integrity protected channel (given that another integrity protected channel previously existed). This can also be done with traditional symmetric-key cryptography. The significant advantage of this use of public key cryptography is that B does not need to keep any secrets. Even if the attacker knows everything that B knows (i.e. the name of A and B and the key IK_A), the attacker still cannot forge messages from A. However, if the attacker can change B's copy of IK_A, it is easy for the attacker to forge messages from A. Thus, B must somehow ensure that his copy of IK_A cannot be modified.

This is the second step in replacing the problem of achieving confidentiality with the problem of achieving integrity. In the first step, confidentiality protected channels were replaced by integrity protected channels. In this step, confidentiality protected keys are replaced by integrity protected keys.

3.3 One-to-many authentication

The property that the verifier needs no secrets can be used to achieve one to many authentication. A can safely give IK_A to as many other entities as she chooses (C, D etc.) without affecting the security of her integrity protected channel with B.

Suppose that A has an (unprotected) channel that can send the same message to many recipients. This might be a broadcast or multi-cast network connection, or it might just be a file that is stored and made available to many entities. A can send $IK_A^{-1}(m)$ over this channel, and achieve the same effect as sending m over separate integrity protected channels to B, C and D. The same piece of information ($IK_A^{-1}(m)$) convinces each of the recipients.

The significance of this is that with public key cryptography it is possible to provide integrity protection for a multi-cast channel in an efficient way. With symmetric key cryptography, a separate authenticator would be needed to convince each individual recipient, as follows:

$$m, IK_{AB}(m), IK_{AC}(m), IK_{AD}(m), \ldots$$

When the number of recipients is large, this sequence of authenticators will be much larger than the single digital signature that was needed in the public key case. It isn't possible to optimise the symmetric key case by giving each of the recipients the same symmetric key (K_{AX}, say) because then each of the recipients could pretend to be A to the other recipients. So there may be broadcast and multi-cast protocols which use public key cryptography just to keep the size of messages small, and which don't require any of the other features of public key. In such circumstances, some of the other properties of public key cryptography can turn out to be a disadvantage.

3.4 Forwardable authentication

In a similar manner, public key cryptography can be used to achieve *forwardable authentication.* If B has received $IK_A^{-1}(m)$ from A, he can pass this on to C and convince her the message really came from A. Again, this works because the verifier needs no secrets. Everything that B needed to verify the message can be safely shared with C, and so they can both verify the same message, using the same means.

This property can be very useful. For example, suppose that B is carrying out some work for A, and needs to convince C that A really asked for the work to be done. This situation occurs frequently where several computers are together jointly providing a service, and the user of that service doesn't know which of those computers will end up doing the real work.

The forwardable authentication property also has a disadvantage. C will be able to verify A's message even if A doesn't want her to be able to verify it. This can be harmful to both A and C.

Danger to the recipient

Forwardable authentication can be harmful to C, because C might be misled by an *out of context replay.* Suppose that A sends a message to B, knowing that it will be understood by B in a particular way based on the relationship between A and B. B can send this message on to C, who will realise (correctly) that it came from A, but may fail to detect that A intended it to be received by B, not C. C may interpret the message in a completely different way from that in which A intended B to interpret it, for example because C is providing a different type of service and expects to receive messages which have a different type of content.

There are several ways in which this problem might be countered. Firstly, putting the name of the intended recipient into the part of the message that is signed makes it clear who the intended recipient is. However, how does the unintended recipient (C) know that the name in a particular part of the message should be interpreted as the name of the intended recipient? B and C might have different ideas about where in the message to look for the name of the intended recipient (because they are using different protocols). The *intended* recipient knows how to interpret the message, because A and B must have had an agreement on what messages mean to be able to attempt to communicate. But C isn't a party to the agreement between A and B, and only receives out of context messages because they have been diverted by an attacker. So if C is very unlucky, the same message which B interprets as indicating B as the recipient might be interpreted by C as indicating C as the recipient, e.g. if the names of both B and C occur in the message, and B and C look in different places for the name of the intended recipient.

What we would like to be able to do is to construct the signed message so that it has the same unambiguous meaning in all possible contexts. But it is impossible to do this in the strong sense of *all* contexts, meaning every state of every entity in the world throughout history. Whatever the message looks like, it is at least possible for there to be an entity somewhere, at some time, which interprets it differently.

Luckily, it is not necessary for the message to be unambiguous in that wide sense. All that is needed is for the message to be unambiguous to all recipients who have acquired by "approved" means the public key needed to check the signature on the message.

Suppose that for every public key, there is an associated definition of the class of messages which are to be verified with that key, including a definition of what those messages mean. Every entity which uses the public key must somehow gain access to this agreement on meanings, and it must gain access to it in a secure manner. This agreement must also have the property that it ascribes the same meaning to the message regardless of who is trying to interpret it, and regardless of when it is received relative to other messages under the same key. This can be implemented by making each certificate which conveys a key indicate (by some means, which must also be unambiguous) the class of protocols for which that key may be used, and that class of protocols must have the property that all of their messages must be unambiguously identifiable.

Once this has been done, there is no possibility of out of context replays causing misunderstanding on the part of the recipient. However, this approach does nothing to prevent the risk to the *sender* caused by the possibility of replay. To deal with this, a different approach is needed.

Danger to the sender

Forwardable authentication can be harmful to A, because A might not want C to be sure what of what A said to B. As one example, suppose that the message m is a service which A provides to B, and for which B pays a fee in return. The forwardable authentication property means that B and C can collude so that C doesn't need to pay for the service, and yet C is protected against B giving her the wrong message. From A's point of view, it is undesirable that C can gain benefit from the service without paying A. This concern might motivate A to use an integrity mechanism which does not have the forwardable authentication property.

3.5 Non-repudiation

The forwardable authentication property of public-key cryptography works even if the eventual recipient (C) has doubts about the honesty of the intermediary (B). Only A knows the private key IK_A^{-1}, and only A could have created $IK_A^{-1}(m)$; B is unable to forge A's digital signature. If B is dishonest, B might have stolen a signed message that was really intended to be sent to someone else, or B might have tricked A into signing a misleading message. Regardless of how B came to be in possession of $IK_A^{-1}(m)$, C can recognise it as a message that could only have been created by A.

Furthermore, B can delay a little before sending the message on to C. C will recognise the message as authentic even if it arrives a little late. Very long delays, such as several years, are a different matter, because C's memory and beliefs might change over time so much that verification is no longer possible. For example, C might forget A's public key.

In addition, B can work out whether the signature will be acceptable to C without involving C. If B is aware of C's beliefs about keys (in particular, if B knows that C believes that IK_A is A's public key), then B can verify A's digital signature and be sure that C will also verify the signature in the same way.

These properties can be combined to provide a dispute-resolving property known as non-repudiation. B can keep the signature $IK_A^{-1}(m)$ as evidence that A signed the message. If no dispute arises, B never needs to perform the second step of passing the signed message on to C. However, if a dispute arises between A and B as to whether or not A signed the message m, B can convince C by performing the second step and forwarding $IK_A^{-1}(m)$.

Non-repudiation almost seems to be provided for free with public key cryptography. That is, a system that uses public key cryptography for integrity at first sight seems to provide everything that is needed for non-repudiation as well. It's not really quite as easy as that:

- With forwardable authentication, it is assumed that A is honest and following the protocol correctly. For non-repudiation, it is assumed that disputes between A and B can arise. This means that it no longer makes sense to assume that A is always the honest party. What can A do if she is malicious and sets out to deceive B or C? How can this be prevented?

- How does B manage to discover C's beliefs about A's key? (B needs to know this in order to check whether A's signature will be acceptable to C). In particular, how does B discover this with certainty when A is being actively malicious and has a motive for causing different parties to have different beliefs about her key?
- Forwardable authentication works if the signed message is forwarded within a "short" time. How short is short? To put it another way, for how long will B's evidence remain valid? How does B discover this?

3.6 Scalable key distribution

When we speak of the *scalability* of a distributed system, we mean the extent by which the amount of work each component of the system has to do grows as the system as a whole becomes larger. Clearly, there tend to be more things happening in a bigger system just because there are more components doing things. But does each component's job become harder just because it is a part of a larger whole? The answer to this question is almost invariably yes, it does to some extent become harder. A more revealing question is, will the work load of an individual component become intolerably large when the system grows to the maximum size we can reasonably expect it to grow?

In this context, a "component" is really a computer system, and its work load is its consumption of resources such as processor time, memory, disc space and communications capacity. As a rough rule of thumb, it is acceptable for each component's work load to grow logarithmically with the size of the system, but linear or worse growth is not acceptable. It may seem reasonable to pay twenty[1] times more to gain access to a system which allows you to communicate with any of 100 million other people rather than just 100 other people. It will probably not seem reasonable to pay a million times as much.

It is often said that public key cryptosystems scale better than symmetric key cryptosystems. However, we really cannot speak of the scalability of a cryptographic algorithm on its own. What really matters is the scalability of the entire system, of which the cryptography forms a small part. To assess scalability, we need to look at an entire system and examine how how the cryptography is used.

Certainly, Kerberos (the text-book example of a symmetric key system) suffers more from an increase in system size than does X.509 (the text-book example of a public key system). However, it is possible to build symmetric key systems that perform better than Kerberos. There is little incentive to add these features to Kerberos, because Kerberos has already established a market niche for itself in communities of a certain size (about the size of a university campus network).

The contention of this dissertation is that it is not the pure numerical size of the community that matters. What matters is that as we look at larger groups, the relationships between the groups' members undergo a qualitative, as well as quantitative, change. For example, they are less likely to be bound to a common employer by contract or subject to the internal disciplinary procedures of an organisation such as a university. This qualitative change is typified by the absence of face to face contact between the people involved, the absence of contractual relations between members,

[1] $20 \approx \log_2 \frac{10^8}{10^2}$

the absence of mutual trust and the absence of strong, enforcible procedural mechanisms to prevent people from misbehaving. A commercial Internet service provider cannot be run as a university campus that just happens to have an unusually large number of students. The change in the nature of the relationships means that a different kind of security needs to be provided, and to provide this different kind of security we need to make use of the special properties of public key cryptography which were described earlier in this chapter. The primary objective is not to save on disc space, processor time or bandwidth (although it would be nice to do that as well).

3.7 Promiscuous key management

With symmetric key systems, the process of exchanging a key (particularly a long term key that is to be used to exchange other keys) can be laborious and difficult. The key must be protected with both integrity and confidentiality; achieving this for long term keys often involves someone physically going to another location, taking the key along with them in some physically protected container. This sort of physical key exchange is very inconvenient, and is often given as one of the reasons why cryptography is rarely used in non-military applications. Public key cryptography offers the promise that it will eliminate some of this inconvenience. However, these inconvenient procedures do have a positive aspect: they reinforce an understanding of who the key is shared with, why it is being shared, and what the participants' mutual obligations are. Public key cryptography makes it easy to share a key without having any clear idea of who you are sharing it with, or why you are sharing it.

Most public-key based systems make it very easy for anyone to get a copy of your public key without telling you. (This is, after all, how public key cryptography is supposed to work). The negative side of this is that you need to exercise caution over what you sign: the entities who verify your signature may not be the ones you expect. This is closely related to the forwardable authentication problems described above, and is closely related to this dissertation's central theme of non-repudiation versus plausible deniability.

With some public key cryptosystems (most notably RSA), it is possible to receive a confidentiality-protected message that contains *no* information about who it came from; this just can't happen with symmetric key. This can be a very useful tool for building certain kinds of security protocols, but it is also a trap for the unwary. With symmetric key, a protocol designer can get away with being vague as to whether integrity is needed or not, because symmetric encryption can provide both (cf. the discussion of Kerberos in 2.1). With RSA, you don't get integrity unless you explicitly put it in.

Worse still, in systems that combine public and symmetric key, a lack of integrity can be passed on from the public key part to the symmetric key part. Consider the following exchange:

$$A \rightarrow B : CK_B(CK_{AB}), CK_{AB}(m)$$

A knows that only B (or rather, a holder of B's private key) will be able to recover m. B, on the other hand, has no idea where m came from.

As a first attempt at solving this problem, what we would like to do is present the application programmer with a protocol that always provides integrity as well as confidentiality (to avoid the problem of "secure" channels to unknown destinations), but which does not provide forwardable authentication (to avoid the risks associated with replays). Cryptosystems based on the discrete logarithm problem seem to have the right properties. I will elaborate on this point in chapter 6.

Chapter 4

The non-repudiation service

According to the ISO non-repudiation framework [13], the purpose of the non-repudiation service is to

> "provide irrefutable evidence concerning the occurrence or non-occurrence of a disputed event or action."

In this context, "irrefutable" should be understood as indicating that the evidence will remain convincing in the face of close examination and careful consideration of the arguments presented by both sides of the dispute. The evidence should convince a person who looks at it deeply, not just one who gives it a hasty glance. This is not to say that the conclusions drawn from the evidence must be absolutely infallible; we admit the possibility that conclusions borne out of long and careful consideration can still sometimes be wrong.

To provide this service, records of transactions are kept, and these records are protected so that they cannot be falsified by parties with a motive to falsify them, and they cannot be erased by parties with a motive to destroy them.

It is clear that one cannot make an item of software or electronics that will resolve disputes in general. Any real system that claims to provide non-repudiation must necessarily be designed for resolving specific types of dispute: this is reflected both in the types of information that the system records, and in the kind of protection against falsification and destruction which is provided.

In systems that do not use computers, paper records such as receipts, cheques and contracts provide something similar. The essential elements are that events are routinely recorded at the time (e.g. a contract records the terms of an agreement), and that the records of the event are made hard to falsify afterwards (e.g. by adding the hand-written signature of an appropriate person).

In a computer-based system, it is easy to record events. The hard part of the problem is ensuring that the records cannot be falsified, modified or erased by someone with malicious intent. There are many ways in which computerised records can be protected from falsification. One of the most common techniques (in the academic literature, if not in actual commercial use) is the technique of public key cryptography which was described in the previous chapter. When public key cryptography is used to provide integrity it is sometimes referred to as digital signature. It bears this name because it is intended to replace the protection which hand-written signatures gave to paper documents. This name is potentially misleading: digital signatures are not digitised images of hand-written signatures, and the type of protection they provide is subtly different. This chapter will explore some of the differences between digital signatures and the mechanisms that they replace.

In the last ten years, many academic papers have been published on the non-repudiation service. This chapter challenges some of the assumptions underlying much of the published literature.

4.1 Evidence is not the same as mathematical proof

The word "proof" has several different but related meanings in English. It is a technical term in mathematics, but it can also be used in a more general sense to mean an action that is intended to discover, test or measure something. A similar ambiguity exits in other European languages (e.g. "preuve" in French has a more general meaning than mathematical proof).

The word "proof" is frequently used when discussing the non-repudiation service, in phrases such as "non-repudiation with proof of delivery". As papers which discuss non-repudiation often contain a lot of mathematics, it is tempting to conclude that the word "proof" is being used in its technical, mathematical sense. This is not the case: whatever it is that a non-repudiation service provides, it is not a mathematical proof.

The difference between a mathematician's notion of proof and other notions of evidence is well illustrated by the following quotation from Wittgenstein [36]:

> "6.1262 Proof in logic is merely a mechanical expedient to facilitate the recognition of tautologies in complicated cases."

With the non-repudiation service, we are interested in establishing facts about the real world, such as whether or not a contract was agreed upon, or whether payment was received for goods which were supplied. ("Propositions with sense" in the terminology of Wittgenstein). It should be clear that no mathematical proof, no matter how long and complex it is, will ever establish the truth or falsity of these types of proposition.

Instead, what we have is a collection of observations and prior beliefs which when taken together tend to convince us that an event did or did not occur. Of course, these prior beliefs may be wrong, and our conclusions may be wrong as a result. We have lost absolute certainty, and with it we have lost any pretence of objectivity: exactly how much evidence is sufficient is in part a matter of taste or policy rather than logic.

Part of the usefulness of a non-repudiation service lies in the ability to convince many different people that an event happened. The user of the service would like to be able to convince everyone they need to convince, and it is not always possible to predict who this will be ahead of time. This aspect of the service would be placed in jeopardy if everyone had totally different, random and unpredictable criteria for accepting evidence. However, it often possible for a community to come to a common understanding of how much evidence is sufficient. Usually, adding more evidence in support of a conclusion doesn't make people less convinced. In this way, a target audience can all be convinced by presenting them with an aggregate of evidence which contains within it the evidence needed to convince each member of the audience.

When making a decision based on evidence, there is often an asymmetry between the seriousness of the consequences of a false negative (deciding that there was insufficient evidence when the disputed event did in fact occur) and of a false positive (concluding that an event occurred when in fact it didn't). This can be reflected in the decision as to how much evidence is considered sufficient. If a false positive has very serious consequences, the evidence can be required to be very strong.

4.2 The parties to the dispute are not necessarily participants in the disputed event

Many discussions of non-repudiation equate the parties to the dispute with the entities involved in the disputed event. Although this may be the commonest case, it is not the only possibility.

For example, suppose that two employees of different companies negotiate a contract on behalf of their respective employers. One or both of them changes jobs, and then their successors disagree over the terms of the contract. The disputing parties have an interest in the event, but were not themselves involved in it. (Indeed, this may even be why the dispute arose!)

The distinction between those involved in the disputed event and those involved in the dispute itself is important because it shows that we cannot assume that the disputing parties have direct, reliable knowledge of the disputed event.

4.3 It is not necessarily the case that one party is telling the truth and the other is lying

Discussions of non-repudiation often assume that there are two parties to the dispute, one of whom is telling the truth and the other is deliberately lying. However, given that the disputing parties do not necessarily have first-hand knowledge of the disputed event, it is possible that they are both acting in good faith, and the dispute has arisen due to a misunderstanding or a malicious action by someone else.

Given that this is possible, we would like the non-repudiation service to be able to detect it. For this to happen, its output needs to be more complex that just "party A is lying/party B is lying". Ideally, we would like a reconstruction of what really happened, even if this disagrees with the expectations of both the disputing parties.

Computer networks offer ample opportunities for parties to become genuinely mistaken. All information is carried by an intermediary (the communication network) which can lose, misroute or modify messages. This can happen even when no-one is malicious (hardware and software failure is sufficient), but the problem is compounded by the presence of an outside attacker.

Even in simple disputes, such as "A says she sent a message to B, but B says it didn't arrive", there are many possibilities as to what really happened.

4.4 There does not need to be a "judge"

Some accounts of the non-repudiation service state that the service always requires a judge or adjudicator with power over potentially disputing parties. The reasoning behind this is that it is needed to cope with the case where one party is intentionally lying. In this case, the disputing parties already know which of them is telling the truth, without needing to examine any cryptographic evidence. For the non-repudiation service to be of any real use in this situation, there must in addition be some means whereby matters can be put right.

However, a court with legal powers to compel the disputing parties is not the only means by which a situation can be put right, or injured parties compensated. The non-repudiation service is usable in a wider range of situations than just court proceedings. For example, one of the parties to a dispute might recover the money they lost in a disputed transaction by claiming against their own insurance policy, rather than by forcing the other party to pay. The non-repudiation service could be used to convince the insurance company that the first party's account of events was correct, and hence that the insurance claim should be paid out. Here, the criteria for evidence to be admissible can be set by the insurance company, based on their own assessment of what is an acceptable financial risk. This can be different from the levels of evidence that are defined by legislation as acceptable for use in criminal trials. Indeed, an insurance company and its customers could use the service in this way even if there is no specific legislation concerning digital signatures.

The existence of other environments in which the service may be used shows that several conditions which are often to be considered to fundamental to the service are not fundamental at all, but rather are specific to a particular application of the service. The disputing parties don't need to specify in advance how disputes will be resolved. Although one method of ensuring that the possibility of arbitration will exist is to describe the arbitration procedure in a contract agreed beforehand, this is not the only method. An independent third party is able to examine the evidence provided by the non-repudiation service even if they weren't nominated in advance as an arbitrator.

Futhermore, the parties don't even need to agree on who the arbitrator will be. In many circumstances such an agreement is needed to make the arbitrator's judgement effective, but not always. For example, if a merchant is defrauded by a customer who has obtained goods and not paid, then the merchant's insurance company can reimburse the merchant without the customer needing to agree. The customer doesn't ever need to know who the merchant's insurer is. The merchant could take the evidence to an alternative adjudicator (such as the police) without needing to have agreed this with the customer beforehand.

If no dispute actually arises, then no dispute resolution procedure is needed. This means that it is possible to use the non-repudiation service without ever deciding what the arbitration procedure is. If a dispute arises, then some method of arbitration must be found, but it is not *necessary* to agree on this unless and until an actual dispute arises.

However, there is a danger that a dispute will arise and it will be impossible to resolve it, because there is no-one who is both in a position to put things right and be convinced by the evidence. To prevent this state of affairs arising, it is desirable (but not necessary) to work out in advance what the arbitration procedure will be, and to check that the type of evidence collected by the non-repudiation mechanism in use will be considered acceptable by the chosen arbitrator.

4.5 Non-repudiation mechanisms can go wrong

The security of cryptographic mechanisms depends on the security of keys, and accordingly great care is taken to ensure that attackers do not obtain unauthorised access to private or symmetric keys. However, we must admit the possibility that such unauthorised access might happen, and that if they do the mechanism will give misleading results.

Some protocols (e.g. X.509) include a revocation mechanism to limit the damage caused by such a key compromise. In the context of the non-repudiation service, however, the addition of a revocation mechanism adds a new problem: the key owner might falsely claim that a key has been compromised, and this also causes the mechanism to to wrong. While good physical security measures should make actual key compromises relatively rare, there is no corresponding physical mechanism to make it hard to *falsely* claim that a key has been compromised.

This problem leaves us with some residual doubt; in some cases where it is claimed that a key has been compromised, we have no way of telling whether the key really was compromised or whether the claim is false. In chapter 6 I describe some improved mechanisms which enable us to resolve some of the doubtful cases; however, there is no known revocation mechanism which removes all doubtful cases.

The existence of this form of doubt is another reason why the evidence provided by the non-repudiation service cannot provide proof with certainty, even though it can provide very convincing evidence. We can be fairly sure that a highly tamper-resistant box hasn't been tampered with, allegations to the contrary notwithstanding. However, we cannot conclude this with absolute certainty.

4.6 Justification and obligation

Although for reasons described above we cannot always be sure of what happened with certainty, we can often establish something which is weaker but still useful. Two important cases of weaker guarantees are:

- *justification:* Having evidence that a participant's actions were justified, even though with the benefit of hindsight (e.g. knowing that the key was subsequently reported as compromised) it can be seen that it would have been better if they had acted differently.

  Questions of justification often arise when trying to determine if a party to the dispute was negligent. Demonstrating that a party's actions were justified amounts to showing that they had performed all the checks which the security policy required them to make, and hence that they were not negligent.

- *obligation:* Having evidence that establishes which party has an obligation to put matters right (even if we cannot establish with certainty who really caused the problem in the first place).

To be useful, a protocol that tries to establish one or both of these goals must be secure against the participants simply lying. That is, participants in a dispute should not be able to escape their obligations by lying, and similarly they should not be able to make an unjustified action appear to be justified.

In addition, a protocol that tries to establish obligations should ensure that whatever happens, if something goes wrong then at least one participant has an obligation to put matters right. (It is acceptable if a failure leaves two or more participants both being individually obliged to do something). This can be achieved by having a single message serve both to discharge one party's obligation and to commit another party. ("I accept responsibility for finishing this transaction, and hereby acknowledge that you have completed your part of it.") Even if the result of the non-repudiation mechanism is wrong as to whether this message was actually sent or not, it will at least be consistent. If the message is deemed to have been sent, then the acknowledging party has the obligation; if it is deemed not to have been sent, then the obligation remains with the first party; there is no intermediate position. Another way to express this is to say that in cases where there is residual doubt, the security policy defines who is obliged to do something, even if this might be unfair.

By speaking of justification and obligation rather than of what in fact happened, we can escape some of the problems that arise from the tenuous connection between the real world and electronic representations. However, a protocol would be considered unsatisfactory if it frequently gave rise to obligations and justifications that bore no relation to reality.

4.7 The OSI non-repudiation services

The OSI reference model [10] identifies two specific forms of the non-repudiation service, known as non-repudiation with proof of origin and non-repudiation with proof of delivery. The former "will protect against any attempt by the sender to falsely deny sending the data or its contents", while the latter "will protect against any subsequent attempt by the recipient to falsely deny receiving the data or its contents".

OSI is exclusively concerned with standardising data communications between systems, and considers the activities which take place within systems to be outside its remit. In effect, OSI is just about sending and receiving data. A natural conclusion to draw is that if one had as basic building blocks mechanisms which provided evidence concerning which messages were sent and which were received (i.e. non-repudiation with proof of origin and non-repudiation with proof of delivery), then these building blocks could be combined to provide evidence about any event which is within the scope of OSI.

I would like to call into question whether this way of breaking down the non-repudiation problem is the most effective way to proceed. I will challenge it on two grounds. Firstly, most non-repudiation protocols are built up from a single building block (digital signature), not two; "proof of origin" and "proof of delivery" are usually implemented the by same digital signature mechanism being invoked under different circumstances.

Secondly, the events for which I am interested in establishing evidence contain elements which go beyond the sending and receipt of messages, even though they often also involve OSI data communications. An example of such an event is "the user instructs their computer to order particular goods from a certain supplier". Contrast this with the OSI event, which is the transmission of a sequence of data items having a particular format. The usual way in which digital signature is used to provide a service which approximates to non-repudiation with proof of delivery is for the recipient to send a signed acknowledgement [11]. This mechanism does not live up to the definition of the service because there is a possibility that the recipient won't play the game. A recipient could receive the message and then not bother to send an acknowledgement. Although in reality the message has been received, there would be no cryptographic evidence of this. A considerable amount of effort has been expended on developing new mechanisms which provide the OSI service in a strict sense [39, 4]. I will take the opposite approach: what the existing mechanism actually does is fine; what is wrong is the OSI definition of what it is supposed to do.

I claim that what the non-repudiation service provides evidence about is its own invocation, not some related event such as the receipt of a message. On its own, this isn't very useful as it isn't connected to the events which are interesting. However, systems are built in such a way that invocation of the non-repudiation service (which is undeniable) should only occur when some other, more interesting, event occurs. Accidents may happen in which the non-repudiation service is invoked even though the intended triggering event hasn't occurred, but evidence that the non-repudiation service was invoked gives a strong indication that the intended triggering event happened.

With cryptographic non-repudiation mechanisms, the undeniability arises from the publication of a value which was once secret; this is undeniable because the public availability of the erstwhile secret can be proven by exhibiting it. Public key cryptography is good for this because it enables us to generate a multiplicity of separate secrets (signatures) from one master secret (the private key). Signatures are not usually regarded as secrets, because once they have been generated they are usually passed on to a verifier (effectively published). However, in this particular way of looking at cryptographic non-repudiation, the signatures which a user *could* have generated, but has not yet decided to generate, are considered secrets. Of course, there are infinitely many of these: but the private key provides a finite summary of them, as it enables any particular signature to be generated.

Once a signature has been published, the fact of its publication cannot be denied. However, it is still possible to argue about what caused it to be published, and what the fact of its publication implies for the disputing parties. These are issues which need to be addressed by the non-repudiation protocols.

Chapter 5

Certificates, certification paths and revocation

5.1 Certification paths

As was described in chapter 2, International Standard X.509 introduced the notion of a *certification path* as a means by which members of a very large distributed system can obtain each other's public keys. The certification path concept has been adopted by many subsequent specifications, including Internet Privacy Enhanced Mail and the VISA/Microsoft "Secure Electronic Transactions". Indeed, there is no widely-used alternative to certification paths for public key distribution in large communities. (PGP uses certificates but not certification paths. However, attempts to make PGP work in large user communities would probably take the form of adding non-hierarchical certification paths to PGP).

Public key mechanisms are critically dependent on users being able to obtain the right public key for other users. Hence it is vitally important to use a method for obtaining public keys that works. In this chapter, the certification path method is examined in detail.

5.1.1 Analysis of authentication properties

In this section, I use the formal logic developed by Burrows, Abadi and Needham [23] to analyse the certification path mechanism. I will start by analysing its properties as an *authentication* mechanism. Later on, I will show how these authentication properties are also relevant to the case of *non-repudiation*.

Before embarking on the details of the analysis, it is worth making a few remarks on what it is reasonable to expect this logic to do for us. One way of looking at BAN logic proofs is that they prove that a high-level description of a protocol can be expressed in terms of a combination of standard building blocks. These building blocks are represented by the axioms of the BAN logic, and in the opinion of the logic's authors they are good building blocks that have stood the test of practical experience. What proofs in the logic do not prove is that these building blocks are actually secure in reality, or that the actual implementation of a security protocol is fully in accordance with its high-level description.

Thus, we do not expect to be able to *prove* that a real implementation of the certification path mechanism actually works; this is too much to ask for. Instead, we expect to be able to gain some insight into the components of the certification path mechanism, and to discover the extent to which these components are ones which have a good track record in other systems.

I have previously published this analysis at the 1988 SIGOPS Workshop on Distributed Systems [27], and in a security study for the UK Joint Network team [35]. However, the consequences for non-repudiation that I shall draw from these analysis are presented here for the first time.

An entity B attempts to prove its identity to A by sending it the certification path $A \Rightarrow B$, which consists of a sequence of certificates:

$$IK_{X_0}^{-1}(X_1,\ IK_{X_1}, t_1, s_1) \ldots IK_{X_{n-1}}^{-1}(X_n,\ IK_{X_n}, t_n, s_n)$$

Where X_0 is A's *certification authority*, X_n is B, IK_{X_i} is (allegedly) X_i's public key, $IK_{X_i}^{-1}$ is the private key corresponding to IK_{X_i}, s_i is a certificate serial number, t_i is a certificate validity period, and round brackets denote digital signature.

A already knows the public key of its CA, IK_{X_0}. What we would like to be able to do is to show that if A knows the public key of X_i they can obtain the public key of X_{i+1} by examining the appropriate certificate, and hence prove by mathematical induction that A can obtain B's public key.

In the BAN logic, the following conditions are sufficient for this inductive proof to work:

1. To start the induction, A must already have a public key (IK_{X_0}) which she considers to be suitable for securing communications with X_0.

2. A has verified each digital signature $IK_{X_i}^{-1}(X_{i+1}, IK_{X_{i+1}}, t_{i+1}, s_{i+1})$ using the public key from the previous certificate, IK_{X_i}.

   This is easy for A to check.

3. A believes that the contents of each certificate are *timely*:

   $$A \models \# X_{i+1}, IK_{X_{i+1}}, t_{i+1}, s_{i+1}$$

4. A believes that X_i has *jurisdiction* over $IK_{X_{i+1}}$.

   $$A \models X_i \Rightarrow IK_{X_{i+1}} \rightarrow X_{i+1}$$

The first two of these conditions are given explicitly in X.509, and any implementation of X.509 would check that these conditions hold. But what about the other two conditions?

5.1.2 Timeliness

In the BAN logic, timeliness (represented by the # operator) includes at least two distinct notions: the notion that two copies of a message may not mean the same thing as one copy, and the notion that beliefs may change over time.

Protection against duplication

Sometimes, two copies of a message have a different effect from one, and it is important to prevent an attacker from taking someone else's message and replaying it twice. For example, it shouldn't be possible to pay the same cheque into a bank account twice by resubmitting copies of it.

Protection against duplication is sometimes necessary for messages that contain or refer to cryptographic keys. For example, as discussed in chapter 2, a one-time pad is very secure if used only once, but becomes very weak if it is used twice to encrypt different messages. One could imagine an attack where the attacker replays key management messages and tricks the victim into using the same one-time pad twice.

Unlike one-time pads, public keys can be used repeatedly with little degradation of security. Indeed, the reason why public key systems are attractive from a performance perspective is that the same certificate (and hence the same certified key) can be used many times for many different messages. If it was necessary to go to the CA to get a fresh certificate for each message, public key schemes would be far less efficient. By using this reasoning outside of the BAN logic, it can be argued that the duplication protection aspect of timeliness is not relevant to certificates.

Change in beliefs over time

The other aspect covered by timeliness in the BAN logic is that the participant's beliefs may change over time, and it is important to establish that a message reflects a participant's current beliefs, not just their beliefs at some point in the past.

This aspect of timeliness clearly is relevant to certification paths. A CA's belief in the validity of a certificate changes when either the corresponding private key is reported as having been compromised, or the subject's name changes (e.g. because the name reflects their organisational role, and their role changes).

So, what this part of the BAN analysis is telling us is that it is important to establish that the CA hasn't changed its mind since issuing the certificate. There are two fields in the certificate that we might use for this: the validity interval and the expiry date. The next section (on revocation) will examine in detail how these fields can be used to establish timeliness.

5.1.3 Jurisdiction

The BAN notion of *jurisdiction* formalises the idea that A must believe that X_i is an appropriate person to sign a certificate for X_{i+1}'s public key. That is, X_i is not just any person (e.g. an attacker) but is someone who A considers to be honest and competent for this function.

There is little problem with this for the first CA, X_0. A has made direct contact with this CA to get their public key, and can be presumed to have made a judgement at that time as to the CA's competence. (Although this does highlight that it is A's opinion of the CA that matters here, not anyone else's. I will return to this point later).

However, without additional assumptions, this fails for subsequent CA's in the path. There is no particular reason why X_i $(i > 0)$ should be someone who A believes to be honest and competent. This is not just a minor theoretical problem; it can give rise to a real attack.

Suppose that someone has implemented X.509 from just reading the International Standard, and has not implemented any jurisdiction-related checks because the standard doesn't mention such checks at all. An attacker who has a valid certification path for themselves can trick such an implementation by pretending to be a CA and creating an apparently valid certification path for B, in which B's public key is replaced with a value chosen by the attacker. The attacker (E, say) knows their own private key and so can use it to sign a message that looks like a certificate for B:

$$IK_E^{-1}\{B, IK'_B, s_B, t_B\}$$

where IK'_B is a false key for B chosen by E. E can take their own certification path ($A \Rightarrow E$) and concatenate it with the above false certificate to make a false certification path $A \Rightarrow B$.

I published this attack in 1988 [27]. Given that this attack is now well-known, new implementations clearly ought to do something to protect against it. There are several possible approaches:

- Only accept certificates from CAs for which the certificate user has direct knowledge. In effect, this means using certificates, but not certification paths. This is what PGP does.
- Include in the certificate an explicit indication of the certificate issuer's opinion of the certificate subject's trustworthiness. This is what X.509 version 3 does.
- Include in the certificate an implicit indication of the certificate issuer's opinion of the certificate subject's trustworthiness. This is what Internet Privacy Enhanced Mail did. PEM adopted a convention whereby the form of an entity's name indicated whether or not it was a certification authority, and if it was a CA, it also indicated which entities' keys it was permitted to certify. In the PEM model, a CA should only sign a certificate in which the subject name has the form of a CA's name if the entity being certified actually is a CA and has been assessed as being competent to perform that function. This prevents users from creating apparently valid certificates, as their own certificates indicate that they are not CAs.
- Provide an entirely separate mechanism that enables users to find out whether or not a particular entity has been assessed as competent to act as a CA. As far as I know, no system in widespread use does this.

Several of the designers of PEM strongly believed that certificates should be pure authentication certificates, and should say nothing about the trustworthiness (or otherwise) of certificate subjects. However, by adding a naming convention to prevent the aforementioned attack, PEM certificates became more than just authentication certificates. At the very least, a PEM certificate indicates whether the subject should (or should not) be trusted as a CA. It is arguable that PEM certificates also contain implicit statements about the relative trustworthiness of users too. For example, if an entity has a certificate naming them as the occupant of a particular role, then this could be taken as a statement by the CA that the certified entity has been judged fit to act in that named role.

5.1.4 Why non-repudiation is different

The preceding discussion was phrased in terms of authentication. The same considerations of timeliness and jurisdiction occur when certification paths are used for non-repudiation, but there are additional complications.

The timeliness issue is complicated because a dispute concerning an event can occur much later than the event itself, and the participant's beliefs may have changed in the interval. In particular, a certificate might be revoked after it has been stored as evidence concerning an event, but before it is needed to resolve a dispute about the event. These interactions between revocation and non-repudiation are explored in the next section.

The jurisdiction issue is also complicated, because while in the authentication case there was only one entity whose beliefs about jurisdiction were relevant, in the non-repudiation case there may be two or more: the entity who stores evidence in the hope that it might help resolve future disputes, and the entity or entities who examine this stored evidence in the course of resolving a dispute. Jurisdiction is to some extent a matter of opinion rather than objective fact, and hence differences of opinion over jurisdiction are likely. Reasoning about non-repudiation involves not just participants examining their own beliefs (as in the case of authentication), but also examining their beliefs about what other parties' beliefs might be.

5.1.5 Completing the paper trail

What happens if a certification authority falsifies evidence for an event that never took place? Conversely, what should we do if a certification authority is accused of having falsified evidence? One approach that might be taken is to say that the risk of falsified evidence is the price to be paid for using digital signatures, and all that can be done is to be careful about which CAs you trust.

However, it is possible to do better than this. Protection against CA misbehaviour is technically possible, and benefits all parties. It benefits the evidence subject, because it protects them against being falsely implicated in events which never took place. It protects the CA against false accusations of having acted improperly, because the CA can show that it was not in a position to have been able to falsify the evidence. Finally, it is an advantage to the evidence user, because it prevents the value of the evidence being undermined by accusations of CA misbehaviour.

We can protect against the CA falsifying evidence by doing the following two things. Firstly, the user should generate their own private key, rather than getting the CA to generate their key for them. If this is done, the CA can't misuse the user's private key because it never has access to it. Secondly, before issuing a certificate the CA should obtain some evidence that the user has requested them to issue a certificate. Such evidence will typically not be cryptographic, e.g. a written request, signed by the user and including a hash of the user's public key. If this is done, the CA can defend themselves against accusations of having certified the wrong key by producing this independent evidence.

This protection against CA misbehaviour has the additional effect that it makes the evidence independent of considerations of jurisdiction, apart from the very minimal and easy to agree policy that users have jurisdiction over their own keys. Once the CA has produced the written statement from the user including the user's public key, it no longer matters who the CA was or who trusts it. The signed data has been tied to the user in way that does not involve the CA.

Although this method prevents CAs falsifying convincing evidence, it does not prevent all forms of CA misbehaviour. A CA can still sign a certificate for the "wrong" public key, such as one for which the CA knows the corresponding private key. Such a deception will temporarily convince an evidence verifier. On-line, the digital signature will match the key in the certificate and nothing will seem amiss. However, once a dispute has arisen and additional off-line checks are performed, it will become apparent that the paperwork is not in order and the deception is definitely the fault of the CA rather than the user.

From this, we deduce that jurisdiction still has a role to play in non-repudiation. An entity holding a digital signature which they hope to later use as evidence is bearing the risk that the CA might have lied, and that should a dispute arise all they will have evidence of is the CAs misbehaviour, rather than the action which they thought took place. Users can still exercise discretion over which CA they are prepared to trust in this way.

5.2 Certificate revocation

5.2.1 The authentication perspective

The BAN logic analysis at the beginning of this chapter showed that the certification path mechanism lacks a means of ensuring *freshness.* To make this mechanism provide authentication in a way which meets the BAN criteria, it is necessary to add something which makes it resilient against changes over time (in particular, resilient to system users discovering that a private key has been compromised).

The 1988 version of X.509 filled this gap by adding certificate revocation lists. Each certification authority regularly dates and signs a list of the certificates which they have issued but no longer consider valid. A certificate which is not invalidated by being mentioned on this list is presumed to be still valid.

As their whole purpose is to provide the element of timeliness which certificates lack, it is essential that revocation lists are issued at frequent intervals and have short "lifetimes" (i.e. that a revocation list with a particular date of issue should only be acceptable for a short period after that date).

However, this regular updating of revocation lists imposes a significant cost: not only must these lists be transmitted across the network, but more significantly they require the existence of a highly available server which can provide these lists when asked. As was described in chapter 2 studies carried out by NIST and my own experiments have shown that these costs are a substantial part of the total cost of providing a certification authority service.

Cost isn't the only problem with the revocation list mechanism: in some circumstances, it doesn't even work. There are gaps in time between the key being compromised and the compromise being discovered; between the compromise being discovered and the CA issuing a new revocation list; and between the new revocation list being issued and users noticing that it is available. All of these gaps provide a window of opportunity in which an attacker can make use of a stolen key.

Clearly, we would like to improve upon this.

5.2.2 The non-repudiation perspective

The revocation list mechanism presents additional problems when it is used in support of a non-repudiation service. As an attacker has a window of opportunity between stealing a key and the key being revoked, there is the question of who bears the liability for malicious use of the key during this period (assuming that the attacker can't be located and brought to justice). This is largely an issue of policy that can be agreed beforehand between the legitimate users of the system, and will be driven by considerations of what the social and commercial relationships are between the participants.

It will often be the case that the participants' liability for misuse (or alleged misuse) of a private key depends on the relative times at which events took place. (In particular, when the key was reported as compromised versus when the alleged misuse allegedly took place). As the underlying assumption of non-repudiation is of an adversarial relationship between the participants, we must consider the possibility that participants might try to lie about the relative ordering of events in order to shift liability from themselves onto someone else. Hence if revocation lists are used, a successful non-repudiation mechanism needs to be able to convince a third party, after the event, of the relative ordering of events. This is not easy to do.

5.2.3 Some improved mechanisms

Notarization

ISO 7498-2 defines *notarization* as

> "The registration of data with a trusted third party that allows the later assurance of the accuracy of its characteristics such as content, origin, time and delivery." [10]

The provider of such a service is conventionally referred to as a *notary.* It is worth mentioning in passing that the service this entity is providing is somewhat different from that which is provided by a "notary public" under English law, and different again from the notion of a notary under French law.

The ISO 7498-2 notary is an attempt to enable the resolution of disputes about the relative timing of events. For example, the recipient of a digital signature can take the signature to a notary and obtain a counter-signature, stating that the notary has seen the signature at a particular time. This safeguards the original signature against subsequent compromise of the key that was used to generate it, by providing evidence that the signature was generated before the key was reported as compromised.

This mechanism doesn't entirely work. Firstly, the recipient of the signature is still at risk if the signature is notarised in the period after the key has been reported as compromised, but before the revocation list has been updated: neither the recipient nor the notary will be aware that the signature is bad (because the CA has not yet got around to distributing a new revocation list), but if a dispute arises it will be discovered that the signature wasn't valid because the corresponding key had been reported as compromised.

There are several responses to this problem. The risk to the recipient has certainly been reduced, even though it hasn't been eliminated, and in some circumstances a recipient might consider this to be good enough. Alternatively, the recipient can get the signature notarised and then wait until they receive a new revocation list from the CA whose start date is after the time of notarization. This, taken together with a policy agreement that a revocation list dated T must include all compromises reported before time T ensures that the recipient doesn't get caught in this problematic interval. The downside of this approach is that the recipient has to wait for the revocation list before acting on the signature; in many applications this wait is unacceptably long.

This mechanism has other problems too. It deals with the compromise of user keys, but what about the compromise of CA keys or even the notary's key? These keys may well be better protected than user keys, and so this mechanism has made failures less likely, but the problem still remains.

Finally, what if the notary lies? Or conversely, how does the notary defend its honesty if it is accused by one of the participants in the dispute? I will deal with this type of issue in the next section.

Separate key for CRLs

In the 1988 version of X.509, the same key is used to sign both certificates and revocation lists. This is certainly in keeping with the X.509 philosophy that each entity has a single key that they use for everything, but it has several practical drawbacks.

Certificate signing keys need to be very well protected, because the consequences of such a compromise are very severe. It is usually recommended that such keys be kept only in a device which is not directly connected to the network, in order to provide greater physical protection. The use of an indirect connection to the network means that there is a much greater delay between a signature being generated and it becoming available over the network. For certificates this is not a problem, but for revocation lists it is a disaster: after all, the whole purpose of revocation lists is to be timely.

Accordingly, we would like to adopt an architecture whereby separate keys are used for signing and for revocation. Revocation keys are kept on-line, which gives the necessary fast transfer from point of signing to point of distribution, but has the disadvantage that revocation keys are less well protected. This doesn't matter too much, because the compromise of a revocation key is much less serious than the compromise of a certificate signing key.

The separation of the two keys has a secondary benefit in that it enables the two functions to be carried out by separate people or organisations. A CA who realises too late that they have incorrectly issued a certificate might be tempted to cover up their error (and escape their consequent legal liability) by back-dating the revocation of the certificate. Such forms of cheating by CAs are made less likely by separating the functions, as the revocation authority has little incentive to implicate themselves to cover up someone else's mistake.

On-line revocation servers

Another approach to revocation is to do away with CRLs altogether, and instead use an on-line revocation server which can confirm that a particular certificate is still valid. The response from such a server should be signed to prevent forgery and to prevent its authenticity being subsequently denied. The response should contain the serial number of the certificate in question, the time, and whether or not the certificate was valid at that time. If the response also contains the hash of a signed message which is to be verified using the certificate, then the revocation server also provides a notary function, confirming that the signed message was in existence at a time when the certificate was still valid. Passing the signed message as a parameter to the revocation server also enables the provider of the revocation service to charge on a per-message basis, and to charge different types of messages at different rates. The disadvantage of this solution is that the verifier needs an on-line connection to the revocation server to confirm each certificate each time. A cached copy won't do. In some applications it is worth the additional communications cost to reduce the risk.

Arbitrated signature schemes

An alternative way to determine the relative times of signing and key revocation is to make the act of signing involve an interaction with an on-line server. ISO 7498-2 calls these types of mechanism *arbitrated signature mechanisms.* (Once again, this is a poor choice of term as the on-line server which participates in the generation of the signature is not necessarily the same as an arbitrator who is called in to resolve a dispute).

The simplest such mechanism is a variation upon the notarization mechanism described above. If the security policy requires the signer to take their signature to a notary to make it valid (rather than making notarization an optional action on the part of the recipient of the signature), then what we have is a form of arbitrated signature mechanism.

However, there are other ways in which the process of evidence generation could be split between the evidence subject and an on-line server. At one extreme, we could have the on-line server hold the subject's key, and generate signatures on behalf of the subject when requested to do so.

Indeed, this scheme is described in ISO 10181-4. Note that this extreme way of partitioning the functionality loses one of the main advantages of digital signatures: it gives the on-line server the ability to falsify evidence. If you don't need protection against forgery of evidence by servers, then the use of public key is superfluous. Indeed, ISO 10181-4 takes this to its logical conclusion and describes how to do non-repudiation with symmetric key techniques (provided you don't mind the server being able to falsify evidence).

However, in the context of this dissertation I am concerned with the threat of forgery of evidence by the server, so I would like to do better than these mechanisms from ISO 10181-4. It is possible to make use of threshold cryptography [5] to create a scheme in which two private keys are needed to create a valid signature: one of these can be held by the user and the other can be held by a revocation server. Both keys are needed, so the revocation server can't unilaterally falsify evidence. If the user detects that their key has been stolen, they inform the revocation server and from then on the revocation server refuses to participate in generating signatures, rendering the stolen key useless. For extra safety, this mechanism can be supplemented by the use of certificate revocation lists as well, to catch the case of the revocation server's portion of the key being compromised too.

The threshold scheme has the slight drawback that the revocation server finds out what the user is signing. The user might not want this, as the document she signs might be confidential. To plug this gap, threshold cryptography can be combined with blind signature techniques so that both keys are required, but the holder of the second key is never made aware of the contents of the documents that are being signed.

Tamper-resistant smart cards

Tamper-resistant hardware devices can often be used as local agents of a remote server. In this fashion, we can construct an alternative to the above protocol which uses tamper-resistant smart cards. In this new protocol, the user's private key is held within a tamper-resistant card. No-one, not even the user themselves, knows the value of this key. The card also shares a secret with a remote revocation server. The card will only generate signatures if it has recently engaged in a challenge-response protocol with the revocation server to establish that the key has not been revoked. If the card is reported stolen, it will stop working as the revocation server will no longer participate in the protocol, and the tamper-resistance of the card will make it extremely difficult for the stealer of the card to extract the private key and use it directly.

This version of the protocol provides the same functionality as blind threshold cryptography, without the expense of additional number-theoretic operations but with extra expense and risk of needing a tamper-resistant object.

Revoke authorisation, not certificates

The Privacy Enhanced Mail specification identified two main reasons for revoking a certificate: compromise of the key and change of the subject's name in response to a change in their access rights. The above discussion has focussed on the case of key compromise. Change of access rights is different. Here, we do not need to worry about the possibility that an attacker is misusing the key. The same person is accountable for the use of the key, whether it occurred before or after the change of rights, and resolution of a dispute will not hinge on timeliness in the same way.

Accordingly, it might be better to use an entirely different mechanism to cope with changes of access rights. For example, one might make the name in the certificate fixed for all time, and use some other protocol to find out what rights are associated with that name at a particular point in time.

As stated previously, some of the designers of PEM wanted certificates to be pure identity certificates, and not access control certificates. The discussion of names changing in the PEM documents shows that they failed to achieve this goal, and let names be representations of access rights. In view of the cost of revocation in the context of non-repudiation, it might have been better to have stuck to the original objective and made the names in the certificates pure identities, which don't change over time.

5.3 The moral of this chapter

The discussion in chapter 4 causes us to reduce our expectations of a non-repudiation protocol, because there are some goals that no communications protocol can achieve. The discussion in this chapter shows that the actual protocols on offer fail to meet even these reduced expectations, because they suffer from a variety of low-level technical problems. There are protocols which do rather better than X.509 at solving these technical problems, but they all fall short of perfection, even allowing for the fundamental limitations explained in chapter 4.

Chapter 6

The plausible deniability service

6.1 The service/threat duality

6.1.1 Distinction between a security problem and a reliability problem

The main identifying characteristic of a security problem is that it involves a conflict of interest between people who interact with a system (where this includes “outside” attackers, as well as “insiders” such as the users and designers of a system). In contrast, with reliability problems, it is assumed that there is no conflict of interest between participants, and that all participants agree as to which outcomes are desirable and which are not.

However, situations often contain concealed conflicts of interest which are not immediately obvious. As a result, a problem that initially appears to be one of reliability may also have a security aspect. It is important to recognise these situations, as the means used to provide security are very different from those used to provide reliability.

As an example, consider the problem of aircraft safety. When this is treated as a reliability problem, there is an implicit assumption that everyone agrees on what the desirable behaviour is (e.g. the aircraft not crashing); mechanisms such as replication of critical components and stringent testing beforehand are used to ensure that the desired behaviour is what actually happens (with high probability). However, this assumption is not always valid. Suppose there is a terrorist threat; then we must admit the possibility of someone desiring (and actively trying to achieve) a different outcome, such as the aircraft crashing. This changes a reliability problem into a security problem, and completely different types of protection mechanisms must be used (e.g. checking of baggage loaded onto the aircraft etc.).

Finally, observe that the owner of an aircraft is motivated by considerations such as reducing operating costs, and sometimes these considerations are in direct conflict with passenger safety. This conflict is a security problem, although it is a rather subtle one. There is another set of mechanisms (this time mainly legal and regulatory) to protect the passenger against the owners/operators of the aircraft.

Returning to the world of information processing systems, we see that this also contains similar concealed conflicts, and that it cannot be assumed that the interests of the provider of a service are co-incident with the interests of a user.

6.1.2 Whose goals should be supported?

When everyone's goals are the same, it is clear what the system designer should be trying to achieve. However, with security problems there is always a conflict of interest, and the system designer has to make a choice as to who is helped and who is hindered.

When cryptographic protocols are described, the situation is often described in terms of "Alice wishes to send a message to Bob, and Eve wishes to listen in", with the implicit understanding that the system has been designed to help Alice and Bob, while hindering Eve. However, was this the right thing to do? Should we instead have helped Eve and hindered Alice and Bob?

Without any information on who Alice, Bob and Eve actually are, and what their motives really are, it is impossible to decide which of them we ought to be helping.

An alternative argument is that as Alice and Bob are the ones sending the messages, they are clearly the owners of the system and hence their views will prevail regardless of the morality of the matter. However, this doesn't always follow. Even though Alice and Bob are sending the messages, it might be the case that they are using a system that was designed, constructed and paid for by Eve, for example if Eve is either a government agency or a network service provider. In short, abstract descriptions of protocols such as the BAN logic [23] don't tell you who is in the right and who has control.

This observation leads me to a conclusion which I shall term "service/threat duality". Every security service is also a threat (when viewed from the perspective of a different participant), and conversely, every threat can also be a security service.

Following this reasoning, security services ought to occur in pairs, where one service helps a particular participant and the other hinders them.

6.2 Plausible deniability

As was stated previously, the goal of the non-repudiation service is to provide "irrefutable evidence concerning the occurrence or non-occurrence of an event or action".

If we believe that there is a need for this as a security service, then by the duality argument of the previous section we must also concede that some participants desire the opposite effect: that there be no irrefutable evidence concerning a disputed event or action. I will call this complementary service "plausible deniability".

The word "plausible" is an essential part of this service. Anyone can always deny anything, even if there is an abundance of evidence which shows that the denial is false. A lie is only effective if it is believable, that is, if it is consistent with known facts. Thus, an invocation of the plausible deniability service involves the following elements:

- Event X actually happened.
- Party A (the service user) wishes to deny that event X happened.
- Party B (the adversary) wishes to demonstrate to a third party that X happened.
- Party A can produce a false account of what happened (Y) which is consistent with all the evidence that party B can present to the third party.
- Party A arranges that all evidence not consistent with Y is concealed or destroyed.

Non-repudiation and plausible deniability are mutually exclusive in that an entity can't both have and not have sufficient evidence to convince a particular party that a particular event happened. However, it is possible to imagine combinations of the two services in which some parties retain evidence of an event while others don't, or the evidence is sufficient to convince some parties but not others.

It must be clear to the entities involved which type of protocol they're running. If the verifier B participates in what it believes to be a non-repudiation protocol, but which later turns out to have been a plausible deniability protocol, then B was using a bad non-repudiation protocol as it allowed the event to be denied. On the other hand, it is often useful to conceal from *outsiders* which type of protocol is being used. All of the example protocols given in this chapter are distinguishable from non-repudiation protocols, both by the participants and by outsiders.

6.3 The plausible deniability paradox

There is a paradox underlying the plausible deniability service - if it is acknowledged that the service is being used, then this in itself may prevent the service from working. In many applications of this service, if the user admits to using the service, then this is equivalent to admitting that they are lying - which negates any benefit the service might have provided in supporting a false account.

With non-repudiation (the dual to plausible deniability), system design documentation can form an essential part of the provision of the service: in order to provide evidence that an event did in fact occur, it may be necessary to produce the system design documentation to show how the system has been carefully designed to provide an accurate and unfalsifiable record of what happened.

With plausible deniability, the opposite is true. To provide the service effectively, it may be necessary to destroy or falsify system design documentation, in order to prevent the adversary being able to prove that the service was used.

For this reason, the term "plausible deniability" is unlikely appear in requirements specifications for real systems: even if you realise that you need this service, you often can't admit (in public) to needing it.

6.4 Motivation

The obvious motivation for plausible deniability is when the two communicating parties are engaged in an activity which is somehow illicit, and they wish to avoid being caught. However there are other possible motives for using a plausible deniability service; this section outlines some of them.

6.4.1 Fair voting

Sometimes, there is a need to take a vote with all communication taking place electronically. There is no particularly compelling reason why the election of government officials should be conducted by electronic means. However, there are other situations where a group of individuals must reach a majority decision, and the nature of the decision means that the vote counting is most conveniently done by computer. For example, if the vote is about whether or not a particular change should be made to data held in a computer, then it is convenient to be able to verify the outcome of the ballot using the same computer system where the resolution which is adopted must be put into effect.

Typically, a computerised vote counting system is required to have the following properties:

Verifiability: It should be possible to verify that the vote counting has been carried out correctly, even with the assumption that the vote-counter has attempted to cheat.

Anonymity: It should not be possible for unauthorised persons to find out which way each voter voted. The idea behind this is that coercion of voters is less likely to happen if it is isn't possible to find out whether the coercion was effective or not. (The word "unauthorised" above conceals some serious issues of security policy - who, if anyone, is authorised to know which way people voted).

The above anonymity property can also be regarded as a plausible deniability property. As long as the voting was not unanimous, each voter should be able to plausibly deny having voted in a particular way. (If it is revealed that the vote was unanimous, the result gives away which way everyone voted, and no cryptographic protocol can do anything to prevent this). This form of plausible deniability can be achieved using blind signatures [3].

6.4.2 Personal privacy

When data about an identifiable individual is held on computer, there is often a legal requirement both that data subjects be able to access records about themselves (so that they can complain if the records are incorrect) and that these records should not be disclosed to third parties who are "not authorised".

As was the case with voting, this situation sometimes creates a significant risk of coercion. Third parties who are not entitled to examine an individual's records might coerce the data subject into obtaining their own records and revealing them. For example, this situation can occur with prospective employers demanding that job applicants provide a copy of their medical records.

An element of plausible deniability can sometimes help here. If a copy of a record can't be verified as authentic (except by people who are authorised to see it), this reduces the risk of coercion. The unauthorised person attempting to obtain a copy by coercion can never be sure that they have been given a true copy. To put it more forcefully: the data subject is given a copy of their record which only they can check is authentic, and are given permission to lie about it to any unauthorised person who asks to see it. Unauthorised recipients obtain a verifiable authentic copy by going through the proper procedures.

This form of plausible deniability can be provided without any use of cryptography. For example, the data subject can be permitted to see their records on screen at special locations but not allowed to take a printed copy away with them. This reduces coercion but may be undesirable for other reasons.

A second type of situation in which privacy concerns might lead to the use of a plausible deniability protocol is electronic commerce; a merchant needs to be able to authenticate her clients so that she knows which goods have been ordered, but for privacy reasons the customer might not want this information to be passed on to other people (e.g. direct mail advertisers). Use of a plausible deniability protocol doesn't prevent the information being passed on, but it reduces its value by removing the guarantees that it is accurate.

6.4.3 Protection of intellectual property

In a society where computers and networks are omnipresent, and copying data is extremely easy, a publisher might take the view that what they are selling is a guarantee of the provenance of information, rather than the information itself. That is, the publisher may decide that it is futile to attempt to prevent their product being copied, but customers will prefer to pay for a known authentic version as opposed to a possibly corrupt pirate copy.

This approach is particularly likely to succeed where the publisher has some potential legal liability for the accuracy of what they publish (e.g. computer software for safety-critical applications). Someone who obtains an unauthorised pirate copy has no legal recourse against the publisher if their copy turns out to be faulty.

To adopt this approach, the publisher needs to distribute the product to the customer in a way which enables the customer to verify its integrity, but does not enable a third party to verify the integrity of a copy. This can be achieved by using a plausible deniability protocol to deliver the product.

6.4.4 Limitation of liability

Organisations who wish to use public key cryptography for a particular application may be deterred by worries about their potential liability. Suppose that the application deals with information of relatively low value, and it is known and accepted that failure of the application might result in loss, corruption or disclosure of the data; however, the fear is that because digital signatures have legal force, a signature produced by the application might end up being interpreted (by someone else) in a way that was not intended, and which results in the organisation being held liable for an amount far greater than that of the data which was intended to be protected. (e.g. the system is only intended for log-on authentication, but it fails in such a way that it produces a signature on what appears to be a high-value contract with someone else).

The root of this problem is that the creator of a digital signature has no control over who that signature will be passed on to. This problem doesn't arise with symmetric key cryptosystems, because they don't have the forwardable authentication property. If it is desired to use public key cryptography for reasons other than forwardable authentication (e.g. so that key management can be achieved without needing encryption), then a plausible deniability protocol can be used to remove the unwanted forwarding property.

6.5 Some protocols

6.5.1 One-time pads

In addition to their original purpose of providing confidentiality, one-time pads can also be used to provide plausible deniability. An ostensible purpose of providing confidentiality can be used as a cover when the true goal is deniability.

The use of one-time pads to provide deniability works as follows: When A and B exchange key material (K), they also agree on a harmless cover message (M') for their subsequent communication. Later, when they need to send a deniable message (M), they encipher it with the pad as they would for confidentiality:

$A \rightarrow B : M \oplus K$

As soon as this message has been sent, A and B destroy their copies of the original key (K) and replace them with an alternative key K', where

$$K' = M' \oplus M \oplus K$$

This replacement should be done in a way that makes everything appear to be consistent with the value of the key having been K' all along (e.g. the modification time on the file containing K' is back-dated etc.) In addition, A and B should act in public as if they had just sent and received the cover message M', while taking care not do anything in public which demonstrates that they have really sent and received M.

If a third party later investigates the situation, everything is consistent with the message having been M'. Even if this third party has been conducting a wiretap and also has the power to compel A and B to reveal their keys, the third party is still none the wiser. If A and B reveal K', this, taken together with the intercepted messages, is consistent with the plaintext having been M'.

The confidentiality service provided by the one-time pad is "perfect", in the sense that even the possession of unlimited mathematical knowledge and unbounded computer power is no help to an attacker in breaking the system (although stealing the key K is another matter!). By the same token, the deniability provided by the one-time pad is "perfect": even the possession of unlimited mathematical knowledge and unbounded computer power does not enable a third party to tell whether the message sent was M or M'.

6.5.2 Diffie-Hellman

Public-key cryptosystems do not provide "perfect secrecy" in the Shannon sense described above. However, from a practical point of view they are much more convenient than one-time pads, and as in reality no-one has infinite mathematical knowledge and computational power their lack of "perfection" is of little practical consequence.

The natural question to ask is whether it is possible to provide the plausible deniability service using public key cryptography, in a way that gives practical convenience at the expense of introducing a theoretical vulnerability to attack by an infinitely powerful adversary.

This can be done using cryptosystems of the Diffie-Hellman type. As before, the cover story for the design of the system is the provision of confidentiality, while the real desire is for plausible deniability. In this case, the cover story is particularly convincing as various government agencies recommend the use of Diffie-Hellman cryptosystems rather than RSA. See for example CESG's recommendations as reported by Zergo Ltd. [38]. In addition, many standard cryptographic protocols such as the U.S. government's Message Security Protocol [21] are designed around cryptographic algorithms of the Diffie-Hellman type. The user of this form of plausible deniability can convincingly argue that they are only seeking confidentiality, and were merely following government recommendations of good practice when they adopted a system of the Diffie-Hellman type.

Consider the following protocol, which is based upon Alfred Menezes' MQV1$'$ [18, 9]. Let p be a prime number previously agreed between A and B, q a large prime factor of $p - 1$, and g an element of order q in the group of multiplications modulo p.

A and B each generate a random number (r_A and r_B respectively) in the range $0 \ldots q - 1$. At the end of a run of the protocol, they will destroy all records of these random numbers. The ostensible purpose of this is to do with confidentiality: it prevents an attacker from deciphering the transmitted ciphertext even if the attacker later breaks into A or B's computer and steals whatever information he finds there. The destruction of r_A and r_B has the side-effect of making it harder for a third party to determine what message was really sent, even if A and B are compelled to reveal all records that they have kept.

A and B exchange modular exponentials of their random numbers:

$$A \rightarrow B : g^{r_A}$$

$$B \rightarrow A : g^{r_B}$$

A and B then generate a pair of shared session keys as follows:

$$\begin{aligned} IK_{AB}, CK_{AB} &= g^{(r_A + X_A Y_A Z_A)(r_B + X_B Y_B Z_B)} \\ &= (Z_B Y_B^{Y_B Z_B})^{(r_A + X_A Y_A Z_A)} \\ &= (Z_A Y_A^{Y_A Z_A})^{(r_B + X_B Y_B Z_B)} \end{aligned}$$

Where X_A is A's private key, $Y_A = g^{X_A}$ is A's public key, $Z_A = g^{r_A}$ is the message that allegedly came from A, and so on. The session keys can then be used to provide confidentiality (by encipherment) and integrity (by message authentication codes) for data exchanged between A and B:

$$A \rightarrow B : CK_{AB}(m)$$

As R_A and R_B are random and uniformly distributed, and exponentiation is a one-one mapping into the multiplicative group, then either A or B can simulate a run of this protocol without involving the other. A invents a random number Z_B, pretends that it was received from B, and proceeds as before. The effect of this is that after a genuine run of the protocol, neither party has any evidence that a real run took place: everything they have could easily have been simulated.

6.5.3 Zero-knowledge protocols

In a zero-knowledge protocol, one entity (the prover) attempts to convince another entity (the verifier) that they know a secret value, without revealing any information at all about the secret. Proofs that such protocols reveal no information are usually based on the fact that the verifier on its own can invent a fake run of the protocol which is indistinguishable from a real one. In a real run of the protocol, the verifier chooses challenges at random, while in a fake run the verifier chooses them so as to make the run appear valid.

The usual motivation for using a zero-knowledge protocol is to protect against cryptanalysis of keys by ensuring that no information about those keys is revealed. However, zero-knowledge authentication protocols also have the plausible deniability property. As the verifier can fake an apparently valid run of the protocol, the verifier's record of a genuine run is of no value of evidence.

Chapter 7

The lifecycle of a private key

7.1 The enemy within

When discussing cryptographic protocols, computer programs are sometimes treated as being synonymous with the people that run them. See, for example, the way protocols are described in Burrows, Abadi and Needham [23]. This can be partly justified on the grounds that human beings by themselves cannot perform complex cryptographic operations or send data over a computer network; these actions are always carried out using computer programs. If a system is described in terms of what is transmitted over a network, human users are almost invisible, and only programs appear in the description.

Computer networks have the characteristic that people cannot interact with them without using computers; unaided human beings cannot perform complex cryptographic operations or transmit data over electronic networks. There are other situations where people use tools to enhance abilities such as strength or speed. However, as tools computer programs are both very complex and very easy to change. The combination of these two properties creates new risks and vulnerabilities which were not present in pre-computer systems.

In the context of the non-repudiation service, the risk created by the programmable nature of computers is the risk that the notion of intent will be undermined.

When not using computers, people usually have a reasonably clear idea of what the possible consequences of their actions will be. On this basis, people can be held responsible for the consequences of their actions. This still holds true even if mechanical tools are involved. However, computers start to undermine this. Bugs in computer software mean that the actual and the intended effect of a program can be radically different. In addition, the user of a program and its author are often different people. The program merges the intent of two people: the authors who wrote the program and the user who runs it in a particular environment with particular parameters.

If something undesirable happens involving a computer, how do we establish who is at fault? The earlier chapters of this dissertation showed how the non-repudiation service can help do this. However, the non-repudiation service only pins the problem down to a particular communication end-point, which is a combination of hardware, software, and a user providing input to the software. The non-repudiation service does nothing to help us disentangle the intent of the user from the intent of the programmers (and there may be more than one programmer!)

An approach that has been useful in other areas of computer security is to separate the parts of the computer program that are critical to security from those that are not critical. Once this has been done, the critical parts can be scrutinised closely, and hopefully this close examination engenders some confidence that the critical parts are sufficiently free from programming errors.

7.2 Generation and installation of key pairs

By the nature of public-key cryptography, the public and private parts of a key pair must be generated together, so that one is the inverse of the other. However, the two halves of the key are typically used by different entities. So the question arises as to who should generate the key pair, and how should key components be transported from where they are generated to where they are used.

As is described in X.509, there are basically three possibilities for the generation of users' private keys:

- Users generate their own keys.
- CAs generate users' private keys for them.
- Private keys are generated by some other entity, separate from both the user and the CA.

When discussing who should generate private keys, the following points should be kept in mind:

- It is vital that private keys are not leaked to attackers, and there may need to be protection against inside attackers as well as outside attackers.
- It is equally important to avoid being in a situation where the attacker generates your private key for you. In this case, not only does the attacker also know the key (because they generated it), but they can also arrange for it to have bad properties. A particularly dangerous example of this has been described by Moti Yung [37]: if the attacker provides the key generation software, they can arrange matters so that the generated private keys are known to the attacker, and so that no-one else can exploit the trapdoor, even if they have inspected the key generation software and discovered what the attacker has done.
- Some protocols are vulnerable to attacks in which the attacker takes someone else's public key and claims it as their own (even though they don't know the corresponding private key). To protect against this kind of attack, CAs would like to have some kind of assurance that the public key has not been taken from elsewhere before they issue a new certificate for it.
- Some cryptographic algorithms assume that keys have special properties, and behave in strange ways if the key does not have these properties (e.g. RSA where the public key has more than two factors). When these algorithms are used with some protocols, an insider attacker can gain unfair advantage by deliberately choosing their own key to be "bad'. Accordingly, a certification authority would like to be convinced that a key is "good" before issuing a certificate for it.
- When keys are transported by physical means, it is often easier to provide just integrity for the transported key than it is to provide both integrity and confidentiality. For example, a public key can be included in a printed book, or on business cards: this gives integrity (it's easy to establish that the copy you have was given you by a trusted source) but it's much harder to ensure that the key hasn't been read by anyone else. This is also partly a key destruction issue — if a key needs to be kept confidential, copies of it need to be disposed of carefully.

7.2.1 CA generates private key

The approach where the CA generates the key has several advantages. The CA can be sure that the key has not been taken from somewhere else, and has been constructed with the right properties, because the CA has made the key itself. In addition, this approach only needs one good source of random numbers for key generation (at the CA) rather than many such generators (one per user). Generating good random numbers is notoriously difficult, so this advantage is not to be discounted lightly. However, if we are interested in providing non-repudiation (rather than authentication or confidentiality), having the CA generate keys has a serious drawback: if a private key is leaked or misused (either by accident or malice) we have no way of determining whether this was the user's fault or the CA's fault.

Non-repudiation is mostly concerned with disputes between "Alice" and "Bob", both clients of the CA. However, if we acknowledge that Alice/Bob disputes may need arbitration to resolve, then we are also drawn to consider whether the Alice/CA and Bob/CA disputes might also require resolving. If the CA generates the keys, then we can resolve Alice/Bob disputes but not Alice/CA disputes. In some circumstances, this is sufficient, but in others it is not (it depends on the social and legal relationship between Alice and her CA).

7.2.2 User generates private key

The next possibility is to have each user generate their own key. This solves the problem of disputes about who leaked or misused a key, because each private key is only known by one entity. It also makes it easier to transport the key to its point of use: with this approach, we don't need to move the private key (which would require confidentiality protection), we only need to transport the public key, and for this integrity protection is sufficient.

A minor disadvantage is that we will have the added expense of a good source of random numbers for each user, but this is not an insurmountable problem (e.g. the key generation program can ask the user to roll some dice and type in the result). More seriously, we need some alternative way for the user to convince the CA that their key is properly formed and not taken from somewhere else.

The approach to this problem taken by Internet Privacy Enhanced Mail [14] is to have the user generate a "self-signed certificate" for themselves, and show it to the CA. Although in PEM self-signed certificates have the same format as ordinary certificates, they perform a very different function: they convince the CA that the owner of the private key (whoever they are) has consented to a certificate being issued for that key in a particular name. (Note that an attacker who wishes to impersonate Alice is only too happy to consent to having their key bound into a certificate for Alice; so a self-signed certificate on its own is no good for authentication). The CA combines this evidence that the owner of the private key wishes to be certified as Alice with separate evidence that Alice wishes to be certified as the owner of the public key (e.g. a written request from Alice), and issues a certificate.

The PEM approach almost solves the problem, and indeed is probably good enough for most purposes, but it misses two guarantees that the CA had in the case where it generates the keys. Firstly, it does not protect against the user deliberately choosing a bad key to attack certain protocols. It provides some minimal protection against this, because with some bad keys it is infeasible for Alice to compute a signature for the self-signed certificate. However, there are some types of key which are both "bad" and for which it is still possible for Alice to compute apparently valid signatures. If you care about this, it is possible for Alice to provide a zero-knowledge proof that her RSA public key has exactly two prime factors [34]. The second thing that the PEM protocol misses is that it does nothing to convince the CA that the key was truly generated at *random*. As far as I know, there are no known protocols to achieve this (although it is not necessarily impossible). Fortunately, there is no real need for this: lack of randomness in the key can be regarded as the responsibility of the user (and the supplier of the user's key generation software), and not the CA's problem; there are no known attacks a malicious user can carry out by deliberately choosing a key which is well-formed and unique to them but just not random enough.

7.3 Use of private keys

As was noted in the introduction to this chapter, there is an important distinction between the user and a program run by the user. To take account of this, we would like the implementation of the signature function to provide the following security properties:

7.3.1 Confidentiality of private keys

For public-key cryptography to work, private keys need to be kept secret from potential attackers. However, it is preferable if application programs do not have the burden of enforcing this. With many applications, we only need integrity of the application's data structures, and there is no strong requirement for confidentiality. If the value of a private key is mixed in with other application data structures, then suddenly there is a need for confidentiality of these data structures, and the environment in which the program executes may not be up to providing this. So what we would like is for the application to be able to invoke a "sign this data" function, where the actual computation of the digital signature is performed in an environment which is protected from the application. In this way, the application itself never needs to directly access the private key, and furthermore isn't able to directly access the key even if it wants to (e.g. because of lack of understanding on the part of the programmer who wrote it, errors in the program, or malicious code)

7.3.2 Protection against malicious programs

The previous paragraph dealt with programs that weren't actually malicious, they just didn't wish to be burdened with the problem of providing confidentiality. Even if we protect the confidentiality of private keys, a malicious program (i.e. one written by a programmer who is malicious) can still invoke the "sign this" function on data items which the person who ran the program doesn't want to sign.

Putting more of the security protocol inside the protected subsystem can provide better protection even against malicious programs:

- If the protected subsystem (rather than the application) provides a time-stamp which is included in the signed data, then we have protection against malicious applications deliberately back-dating or post-dating signatures (e.g. to defeat the revocation mechanism when the user discovers the application has been subverted and tries to revoke their key).
- If the protected subsystem includes the name of the application which invoked it in the data which was signed, then it is possible both to provide fine-grained access controls over which application is allowed to sign particular types of messages with a particular key, and

to identify which application was at fault if bad signatures are issued. (Note however that it will not necessarily be possible to prove to a third party which application was at fault, as there exists the possibility that the user might have modified their local system to put the wrong name in. There is also the issue that knowing the name of the program is not the same as knowing the content of the program).

- The protected subsystem can maintain a compact record of all the signatures which it has generated, such as a cryptographic hash of them. This can be used to provide a powerful end-to-end check that the system hasn't been subverted. For example, in systems where it is mandatory for signatures to be notarised, the notary's record of what the user has apparently signed can be compared against the user's record of what they believe they have signed.

This still doesn't provide protection against a malicious application signing the wrong thing. If we need to protect against this, then the trusted subsystem needs to have a user interface. It needs to be able to display the data to be signed to the user (in a form which is comprehensible) and to get a confirmation from them that they really do want to sign it. This requires what is known as a *trusted path* facility: the user needs to be sure that the display really was generated by the trusted subsystem (and not a malicious program), and the trusted subsystem needs to know that the reply really came from the user (and wasn't faked by a malicious program).

7.3.3 Independence from the implementation

It is often the case that implementations of the same protocol are available from multiple vendors. Furthermore, the same vendor will often bring out new and enhanced versions of their product. It is desirable to be able to switch to another vendor's product, or upgrade to a new version, relatively easily.

Unfortunately, the cryptography can make this difficult. If the user's private key is stored in the application's data structures, then typically the manner in which this is done is viewed by the vendor as a trade secret and they are not inclined to tell the customer how to extract it so that they can switch to an alternative product. In turn, this means that the customer can't switch vendors without generating a new key and getting it certified. At a minimum, this involves out-of-band communication with the CA, and it may involve additional complications if the user want evidence generated under the old key to still be considered valid. This is all very inconvenient.

On the other hand, if the private key is stored in a protected subsystem which is separate from the application, and which has a defined interface, then it is possible to switch to a new implementation without difficulty.

The slight downside of this is that if one of the implementations has a serious bug (or is malicious), it can be tricky for the user to work out which was at fault. The techniques described above of having the trusted subsystem include the name of the application in the signed data can be extended to include the version number and vendor as well to catch this problem.

7.4 Destruction of private keys

One of the advantages of public keys is that it is not necessary to destroy them carefully. No harm will result if an attacker obtains a public key by reading an old paper form that was submitted from a user to a CA, or by examining the contents of memory or discs after a program has been run.

However, we do need to be careful with private keys. Once a signature key has reached the end of its lifetime (i.e. all certificates for the key have expired, and no CA is willing to re-certify it with an extended lifetime) then no useful purpose is served by retaining it. (Private keys used for confidentiality are another matter; it is possible that old ciphertext will be found after the key has expired that the user would like to decrypt). Old private keys are only a source of danger: an attacker might manage to steal them and back-date a signature, or the user might back-date a signature to make it appear that an attacker has done this.

It is therefore a sensible policy to delete private keys once their certificate has expired. As the public key is still retained, old signed data can still be verified, so no evidence has been lost.

With key destruction, the end of the lifetime of the evidence (as opposed to the key, or the certificate) is vague. As time goes on, the signature becomes gradually less convincing: advances in cryptology make cryptanalysis more likely, and more and more of the physical paperwork that supports the keys will have been thrown away. However, if the algorithm chosen is secure enough and CAs keep their paperwork for long enough, the evidence can remain valid for a very long time.

There is, however, a more drastic alternative: private keys can be published after the corresponding certificate has expired. Once this has been done, all signatures produced under that key are no use as evidence unless they have been previously registered with a notary, as now anyone can use the published key to forge a signature. This approach puts a hard time limit on the lifetime of signatures as evidence: there is a fixed date before which all signatures must be either discarded as being no longer of relevance or converted into some new and more durable form, either by notarising them or getting the user to re-confirm them.

Key publication has the advantage that it forces people to use a mechanism to make evidence more durable, rather than hoping that it will last for ever. Its serious disadvantage is that it means that there must be an interface through which the private key is extracted and published, which raises the question of what happens if someone maliciously does this before the advertised expiry date.

7.5 Transport of private keys

From the preceding discussion it can be seen that transport of private keys from one system to another is not necessary: generation, use and destruction all occur within the same system. Not only that, but access control mechanisms put in place to prevent accidental key compromise by faulty applications will tend to prevent such transportation.

Upgrading from one implementation of an application to another (perhaps from a different vendor) is achieved not by moving the private key from one place to another but by giving the new version of the program access to the operating system facility which is invoked to use the key. The key stays where it is, under the protection of the operating system.

It might be argued that this view of key management is only applicable to one particular style of distributed system, characterised by large mainframe computers in bank vaults with attached cryptographic processors. Not all distributed systems are like that. In particular there are systems where users are mobile (access the system from different computers at different times), and where as many components of the systems as possible are made stateless in order to facilitate recovery after hardware failures. When building such a system, the temptation arises to transport private keys across a network. However, this can be avoided by either embedding the private key in a transportable processing unit (such as a smart card) or by giving workstations short-lived private keys to which authority is delegated only for the duration of a session (i.e. the generate-use-destroy cycle is completed in hours rather than months).

7.6 Key generation revisited

At the start of this chapter, I outlined how carelessness on the part of programmers was just as real a threat as the outside attacker. (Indeed many attacks are a combination of both — a programmer error exploited by someone else). In view of this we would like security software to be testable. If it can be tested, this increases the chance that the customer will notice if they have been sold faulty software, and it also increases the chance that the manufacturer will notice that their software is faulty before shipping it.

Digital signature is eminently testable, in the sense that it is possible to test that a bit pattern which is supposed to be a signature really does match the public key (and the data which is signed has the value it is supposed to have). Furthermore, because public keys are public, the testing tools do not need any privileged access to key material and so we have no worries that errors in the testing tools will compromise the system under test.

Other aspects of public key cryptosystems are not so easily tested. For example, it is harder to test that the signature generation module is not leaking the private key by some other communications channel. This can be stopped by using the operating system confidentiality mechanisms to ensure that the only place the signature routine can write to is its output, and then testing that its output has the correct value.

More fundamentally, some of the cryptographic operations involve the generation of random numbers, and it is very hard indeed to construct a black-box test to show that an allegedly random number is actually random.

As a case in point, some digital signature algorithms (such as DSA [20]) use random numbers in the course of generating a signature. It has been shown by Simmons [32] that this non-determinism can be exploited by a malicious implementation to leak additional information. A more realistic threat is that the random number generator will simply fail to be random enough, and this is enough for an attacker to recover the key [29]. These problems with signature can be fixed either by using a deterministic signature algorithm (such as RSA) or by defining the pseudo-random number generator to be used. If a pseudo-random number generator is used, it must be keyed in some way (or else the attacker could break the system by predicting it). The key for the pseudo-random number generator can be treated as part of the private key, and managed using the same techniques as are used for the private key. The only way in which such a non-standard "extended private key" could cause problems would be if it was necessary to transport the private key to another manufacturer's system which didn't extend the key in the same way. But, by the preceding discussion, transportation of private keys between systems doesn't happen, so this isn't an issue. The pseudo-random number generator can then be tested; however, this does require that the test software knows the pseudo-random number generation key, and so the test software needs to have privileged access to private key material.

A more fundamental cause of non-testable randomness is key generation. Here, using a keyed pseudo-random number generator doesn't solve the problem, because it leaves us with the problem of where the random-number generation key comes from (and what generates it, and how we know that it was generated at random). The best we can do here is to split the computation into two parts: one that contains all the complicated mathematics (such as primality testing) but which is entirely deterministic and hence testable, and one which is as simple as possible and generates the random numbers. If we want to verify this key generation box, it cannot be with a black box test: we have to resort to examining the source code and the physical construction of the device.

Chapter 8

Conclusions

Properties of Diffie-Hellman

The Diffie-Hellman cryptosystem and its descendants (e.g. MQV1$'$) enable us to do some surprising things. They are not just a poor substitute for RSA that use the discrete log problem instead of factoring; they have some special properties that RSA lacks. It has long been known that Diffie-Hellman provides a confidentiality property knows as "perfect forward secrecy" [7]. In chapter 3 I argued that some of these cryptosystems are less prone to being used in bad protocol designs than RSA, because they can only establish confidentiality-protected channels with entities who have been authenticated. (Although the original Diffie-Hellman cryptosystem can be used to set up a confidentiality-protected channel between two entities who remain anonymous to each other, MQV1$'$ and Nyberg-Rueppel [25] have the authentication built in). In chapter 6, I showed that some of these cryptosystems also have a special integrity-related property known as "plausible deniability".

Design of cryptographic modules

Chapter 7 outlined some of the design criteria for a cryptographic module that is to be used for non-repudiation. Although the main objective is the same as with cryptographic modules used for confidentiality (namely, to prevent malicious or erroneous applications disclosing keys), the end result is very different from a traditional cryptographic module design such as that of Meyer [19].

Some of these differences stem from the use of asymmetric (public key) cryptography rather than symmetric-key cryptography. Although at the mathematical level RSA signature and verification are almost identical (they are both modular exponentiation), the security properties we require of these two operations are so different that we may need to perform them on different physical hardware (signature on a dedicated cryptographic processor and verification on the main CPU), or at least in different address spaces.

Other differences in module design stem from differences between confidentiality and integrity-related services (including non-repudiation). With confidentiality, there may be a requirement to be able to decrypt old data and so old keys mustn't be lost, whereas with integrity and non-repudiation it is much more important to prevent disclosure of keys than it is to prevent loss.

Finally, we are much more concerned about malicious programs in the context of non-repudiation than we are in the context of integrity; this in turn means that separation between the application program and the cryptography needs to be more rigorous for non-repudiation.

Non-repudiation is not the same as integrity

At the beginning of this dissertation, I outlined how confidentiality differed from integrity. In the light of the subsequent discussion, it can be seen that although it is similar to integrity, non-repudiation is significantly different. What you do to convince someone else about the truth of an assertion is significantly different from what you do to convince yourself.

In typical uses of integrity, verification must be fast; the verifier needs to know that the copy they have of the information is correct so that they can get on with the job of using it for some purpose. In contrast, the dispute resolution phase of non-repudiation does not have to be rapid; this in turn means we can use techniques which are too slow for integrity (e.g. referring to physical paperwork which supports the electronic representation of the information).

Evidence and secrets

In chapter 2 I explained that keeping secrets and providing evidence are not the same problem, and that it would be desirable to be able to provide evidence by means that do not involve secrecy. Public key cryptography enables the provision of evidence with only a restricted form of secrecy: secrecy of a public key which is randomly generated and which never leaves the device in which it was generated. However, chapter 5 shows that secrecy is fundamental to public key cryptography. What is undeniable about a digital signature is that a value which was previously a secret (the signature of the data) has ceased to be secret. The problems with revocation described in chapter 5 are partly caused by this connection between secrecy and digital signature. A private key which is intended to remain secret forever might become inadvertently (or intentionally) revealed, and recovery from this situation requires revocation mechanisms and their attendant problems.

Collision-free hash functions were introduced in chapter 2 as a component part of the digital signature mechanism. However, they can also be used on their own to provide forms of evidence which do not involve secrecy at all. For example, a cryptographic hash value mentioned in a written contract links the inputs to the hash function (e.g. public keys) to the terms of the contract in a convenient way. The PGP public key fingerprints mentioned in chapter 2 are another instance of this type of use of collision-free hash functions. Hash functions can't be used as a complete replacement for public key cryptography, because they are not as convenient to use in some circumstances. For example, a user must commit to the data to be hashed before transmitting the hash output over a physically secure channel, but she can send her private key over that channel before deciding which data items she is going to sign.

Chapters 5 and 7 describe mechanisms such as notarization which are intended to safeguard digital signatures against the eventuality that the private key is later revealed. These mechanisms can be viewed as a means of converting one form of evidence into another. In particular, they convert forms of evidence that involve secrets (such as digital signatures) into forms of evidence which do not (such as notarised hash values).

The problem of interpretation

Non-repudiation mechanisms suffer from the problem of interpretation: the meaning which a recipient ascribes to a message may differ from that intended by its sender. This problem is not unique to computer communication, and occurs in many situations involving communication by means of signs, e.g. literary or theological texts. In the context of computer communication protocols, an attempt can be made to solve this problem by putting as much of the context as possible in the message. This attempt can never be entirely successful, because interpretation necessarily depends on some context which is not itself part of the message.

In view of this, we must amend our view of the intended audience of the evidence produced by non-repudiation protocols. This evidence is not for the benefit of arbitrary “outsiders”; it is intended for viewing by members of a closed community who subscribe to a shared set of beliefs concerning interpretation and the acceptability of various forms of evidence. In a similar vein, plausible deniability can be seen inter alia as an attempt to guard against misinterpretation by rendering evidence totally uninterpretable (indistinguishable from random) to those who are not members of a closed group.

X.509 is not a non-repudiation protocol

X.509 was designed as an authentication protocol for a specific application (the X.500 Directory). Its designers hoped that it would also be suitable for non-repudiation, on the assumption that all you need for non-repudiation is public-key cryptography [12, Annex B]. However, we have seen that there is more to non-repudiation than just public-key cryptography.

In particular, a protocol specifically designed to provide non-repudiation would probably differ from X.509 in at least the following respects:

- It would take more care about being able to convince a third party about the relative times of events, especially the act of creating a signature with a key relative to the revocation of that key.
- It would associate cryptographic keys with names that stayed the same; in particular, it would ensure that names didn’t change between the occurrence of an event and the resolution of a dispute concerning that event. A role name (indicating that the bearer of the name is currently authorised to perform some function) should be kept separate from the name that enables the role occupant to be later on held accountable for their actions.

Residual risk

Non-repudiation protocols tend to have some residual risk. That is, there are circumstances (hopefully rare) in which the protocol fails to live up to our expectations and one of the participants suffers as a result.

It is important to recognise that these risks exist and to understand who is bearing the liability for them; not least, so that they can make adequate financial provision for when things do go wrong.

By choosing different cryptographic protocols, we can change which party bears the risk and swap one form of risk for another (e.g. we can exchange reliance on a cryptographic algorithm for reliance on the tamper resistance of a physical object). The risk can be made smaller at the expense of more data communications or greater use of cryptography, but it is rarely eliminated entirely.

Be sceptical about cryptographic evidence

In protocols which do not attempt to provide non-repudiation (such as Kerberos or the EES) cryptographic information could have been created in several different ways, and there is no way for an independent observer to know which method actually created it. These protocols leave the independent observer unable to resolve disputes of the form “Alice says Bob created the message but Bob says Alice created the message". This, however, has long been understood by the cryptographic community.

Non-repudiation protocols aim to convince the independent observer of a particular account of what took place by presenting them with evidence that is consistent with only that account. However, even with non-repudiation protocols there are in fact alternative, conflicting accounts of what took place which are also consistent with the collected evidence. Even with a non-repudiation protocol, we need to look beyond the cryptography to understand what actually happened. Where non-repudiation protocols depend on "tamper proof" devices, we have to ask ourselves "Could this device have been tampered with?". Where non-repudiation protocols depend on the testimony of an allegedly independent witness to the events, we have to ask ourselves "Are they as independent as they claim? Could they be lying?" Many of the non-repudiation protocols do in fact depend on witness testimony, even though this may be obscured by technical details and terms such "Trusted Third Party". It is important to recognise witness testimony for what it is, and not to mistake it for absolute mathematical truth, even if the testimony has a lot of mathematics in it.

Bibliography

[1] D. Balenson. Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers. RFC 1423, February 1993.

[2] S. Berkovits, S. Chokhani, and J. Furlong. Public key infrastructure study. Technical report, The MITRE Corporation, November 1993.

[3] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of Crypto 82*, pages 199–203. Plenum Press, 1983.

[4] T. Coffey and P. Saidha. Non-repudiation with mandatory proof of receipt. *Computer Communications Review*, 26(1):6 – 17, January 1996.

[5] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In G. Brassard, editor, *Advances in Cryptology — CRYPTO '89*, number 435 in Lecture Notes in Computer Science, pages 307 – 315. Springer-Verlag, 1992.

[6] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[7] W. Diffie, P. C. van Oorshot, and M. J. Wiener. Authentication and authenticated key exchanges. In *Designs, Codes and Cryptography 2*, pages 107–125. Kluwer Academic Publishers, 1992.

[8] M. E. Hellman. An overview of public-key cryptography. *IEEE Communications Society Magazine*, pages 24–32, November 1978.

[9] IEEE P1363 working draft. Technical report, IEEE, February 1997.

[10] International Standards Organization. Information Processing Systems — Open Systems Interconnection — Basic Reference Model — Part 2 : Security Architecture. International Standard ISO 7498–2, 1988.

[11] International Standards Organization. Message transfer system : Abstract service definition and procedures. International Standard ISO/IEC 10021–4, 1988.

[12] International Standards Organization. Information Technology — Open Systems Interconnection — The Directory : Authentication Framework. International Standard ISO/IEC 9594–8, 1993.

[13] International Standards Organization. Information Technology — Open Systems Interconnection — Security Frameworks for Open Systems — Part 4: Non-repudiation. International Standard ISO/IEC 10181-4, 1996.

[14] B. Kaliski. Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services. RFC 1424, February 1993.

[15] S. Kent. Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management. RFC 1422, February 1993.

[16] J. Kohl and B. Neuman. The Kerberos Network Authentication Service (V5). Internet RFC 1510, September 1993.

[17] J. Linn. Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures. RFC 1422, February 1993.

[18] A. J. Menezes. The fast MQV protocol. Technical report, September 1996.

[19] C. H. Meyer and S. M. Matyas. *Cryptography: a new dimension in computer data security.* John Wiley and Sons, 1982.

[20] Digital signature standard. FIPS 186, National Institute of Standards and Technology, May 1994.

[21] National Security Agency. *Secure Data Network System : Message Security Protocol (MSP)*, January 1996.

[22] R. M. Needham and M. Abadi. Prudent engineering practice for cryptographic protocols. Research Report 125, DEC Systems Research Center, June 1994.

[23] R. M. Needham, M. Burrows, and M. Abadi. A logic of authentication. Research Report 39, DEC Systems Research Center, February 1989.

[24] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12), December 1978.

[25] K. Nyberg and R. Rueppel. A new signature scheme based on the DSA giving message recovery. In *First ACM conference on computer and communications security*, pages 58 – 61. Association for Computing Machinery, 1993.

[26] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[27] M. Roe. Jurisdiction of certification authorities. In *1988 SIGOPS European Workshop*, 1988.

[28] RSA Encryption Standard. PKCS 1, RSA Laboratories, November 1993.

[29] J. I. Schiller. Issues surrounding the use of cryptographic algorithms and smart card applications. In *Proceedings of the Privacy and Security Research Group Workshop on Network and Distributed Systems Security 1993*, pages 115 – 118. National Technical Information Service, 1993.

[30] B. Schneier. *Applied Cryptography.* John Wiley and Sons, second edition, 1996.

[31] C. E. Shannon. A mathematical theory of cryptography. Technical Report 45-110-92, Bell Laboratories, 1945.

[32] G. Simmons. Subliminal communication is easy using the DSA. In Tor Helleseth, editor, *Advances in Cryptology — Eurocrypt '93*, number 765 in Lecture Notes in Computer Science, pages 218 – 232. Springer-Verlag, 1994.

[33] S. G. Stubblebine and V. D. Gligor. On message integrity in cryptographic protocols. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, May 1992.

[34] J. van de Graaf and René Peralta. A simple and secure way to show the validity of your public key. In Carl Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, number 293 in Lecture Notes in Computer Science, pages 128 – 134. Springer-Verlag, 1987.

[35] S. Wilbur and M. Roe. A study of security in directory and electronic mail services. Technical report, Joint Network Team, October 1990.

[36] L. Wittgenstein. *Tractatus Logico-Philosphicus.* Humanities Press International Inc., 1961. Translated by D. F. Pears and B. F. McGuinness.

[37] Adam Young and Moti Yung. The dark side of "black box" cryptography or: Should we trust capstone? In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, number 1109 in Lecture Notes in Computer Science, pages 89 – 103. Springer-Verlag, 1996.

[38] The use of encryption and related services within the NHSnet. Reference number E5254, Zergo Ltd./NHS Executive Information Management Group, 1995.

[39] J. Zhou and D. Gollman. A fair non-repudiation protocol. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 55 – 61. IEEE Computer Society Press, 1996.

[40] P. R. Zimmerman. *The Official PGP User's Guide.* MIT Press, 1995.