CHAPTER 6

Digital Signatures

C. J. MITCHELL
Royal Holloway and Bedford New College
Department of Computer Science
Egham, TW20 OEX, England

F. PIPER AND P. WILD
Royal Holloway and Bedford New College
Department of Mathematics
Egham, TW20 OEX, England

1 INTRODUCTION

Since the original description of the concept of a digital signature in Diffie and Hellman's 1976 paper [Diffie 76], there has been a great deal of public interest shown in applying this idea to practical security problems. The publication of the Rivest–Shamir–Adleman (RSA) algorithm in 1978 [Rivest 78] showed that practical schemes for realizing the concept do exist. As a result many experts have been expecting digital signatures, and public key cryptography in general, to have a major impact on the design of secure systems.

However, it is only now, some 15 years later, that this revolution can truly be said to be taking place; even as recently as 5 years ago, digital signature techniques still had not had much impact on the marketplace for security products. Why should this be happening now, and why did it not take place earlier? One possible reason is that only in the last 5 years or so has general commercial awareness of the need for security grown to the point at which serious money is being spent. The traditional marketplaces for security products, namely the military and national governments, are not the places in which digital signature services are likely to be very important. One the other hand, resolution of disputes and protection against repudiation of messages are of fundamental importance in the commercial arena, and are often much more important than issues such as confidentiality of data.

The purpose of this chapter is to provide a user guide for digital signatures. In doing so it covers four major topics: (i) what a digital signature is (and how one is used in practice), (ii) techniques for generating digital signatures, (iii) techniques for cryptographic hash functions, and (iv) applications for digital signatures.

The second section, Fundamental Concepts, discusses the nature of digital signature techniques, and what distinguishes them from other methods for providing data authentication and integrity protection services. General approaches for constructing digital signature algorithms are described, prefatory to the material covered in Section 3. A discussion of the legal implications of digital signatures is included, together with a general discussion of commitments and of disputes and their resolution. Finally, the use of hash functions is motivated by considering practical aspects of the use of digital signature algorithms.

The third and fourth sections, Techniques for Digital Signatures and Techniques for Hashing, respectively, concentrate on specific examples of digital signature and hashing algorithms. The aim is to cover some of the most important algorithms, while not attempting to be encyclopedic. Full descriptions of each algorithm are given, with some discussion of possible attacks against these types of technique.

The chapter concludes in the fifth section, Applications for Digital Signatures, with a description of some key areas for the application of digital signatures, including a discussion of key certification.

2 FUNDAMENTAL CONCEPTS

2.1 Digital Signatures

There are many situations where the only security requirement is that the receiver should be confident that the contents of a message have not been altered since it left the sender and that the identity of the sender is not misrepresented. In other words, the receiver needs assurances concerning the authenticity of the message and of its origin.

For the moment we will ignore the problem of identifying the sender and concentrate on the problem of ensuring the authenticity of the message, that is, the message integrity. If data are transmitted over a noisy channel then, provided that the noise level is not too great, there are a number of well-known techniques for ensuring that the receiver can, at worst, detect that the message has been distorted. In fact, in many situations it is even possible to ensure that the distortion is "removed" so that the correct message is received. These techniques involve the use of a suitable error detecting (or correcting) code. Unfortunately, as we will illustrate with a trivial example, they do not offer any protection against deliberate alterations.

The simplest type of error detecting code is probably the parity check code. If the message is a string of bits then, if a parity check code is used, the transmitted signal contains an additional bit which is chosen to ensure that the total number of 1's in the transmitted signal is even. If a single bit, or any odd number of bits, is changed during the transmission then the received message will contain an odd number of 1's and the receiver will know that the received message is different to the message that was sent. Thus this code is guaranteed to detect a single error.

The reason why such a code offers no protection against deliberate alteration should be clear; anyone wishing to change some bits would merely adjust the parity bit (if necessary) so that the number of 1's in the altered message was still even. The alteration would then be undetectable and the receiver would accept the message as authentic. Although this particular example is ridiculously simple, it illustrates the fact that if the technique that is used to provide the authenticity is publicly known and

implementable by everyone then anyone who knows the technique can alter the content of a message and then use the technique to "adjust" the altered message so that it appears authentic. Thus, if an authentication technique is to protect against deliberate alteration it must be such that the receiver is confident that only the sender could have applied it to the message. This suggests the use of some form of (secret) identifying information by the sender.

One possible solution involves the use of a conventional symmetric key cryptographic algorithm. In this scenario the sender uses the algorithm with a secret key, known only to the sender and receiver, to produce a "cryptographic checksum" from the message. This checksum, whose value should depend on each bit of the message, is then appended to the message prior to transmission. Thus the process is somewhat analogous to the parity check code described above. There is one crucial difference, however. A cryptographic checksum for a given message can only be computed by someone who knows the secret key and thus, since anyone who alters the message will be unable to make the appropriate change to the cryptographic checksum, the participants can be confident that, so long as they manage to keep the key value secret, no third party can alter their messages without detection. (This technique is frequently used by financial institutions and one very common application is the protection of transmissions between an automatic teller machine (ATM) and the institution host computer. A particular technique for computing the cryptographic check, which is often called a message authentication code (MAC), is described in American National Standards Institute (ANSI) X9.9; see sections 2.8 and 4, below.)

This solution to the message authentication problem is perfectly acceptable provided that the participants have faith in their ability to keep the key value secret and provided there is never a dispute between the sender and receiver. Furthermore, it may also be sufficient to convince the receiver of the sender's identity. There is nothing, however, to prevent the receiver from changing the received message and then altering the cryptographic checksum so that it is authentic.

Similarly, there is no way to prevent the sender from "proving" that he sent a message that was different to the one received. Thus there is nothing in this solution that prevents either party from trying to cheat the other and, in the case of a dispute, no evidence to enable a third party to settle the disagreement. The reason for this is that the cryptographic checksum depends on secret information that is shared by both parties and, therefore, either party can "imitate" the other.

We must stress here that if the two parties trust each other then the above solution is probably sufficient. If they do not, however, as will certainly be the case if they do not even know each other, then the ability of the receiver to impersonate the sender must be removed. Instead we need the sender to be able to "sign" the message in such a way that if anyone changes the message during transmission then the "signature" will reveal the fact that an alteration has occurred. Furthermore, if the signature is such that it cannot be "forged" then this will also authenticate the sender. We have deliberately used the term "signature" here because of the obvious analogy with written signatures.

In paper-based transactions the validity of a document is authenticated by a written signature. This signature then serves as evidence of the signer's agreement to the authenticity of the information on the document and, furthermore, can be presented in court if the signer ever tries to deny agreeing to the statements on that document. The emergence of computer-based message systems has led to the need to find a digital equivalent of the written signature.

The concept of a "digital signature" was first discussed by Diffie and Hellman in their classic paper "New Directions in Cryptography" [Diffie 76] and has since been the subject of numerous research papers, conference lectures, etc. [Akl 83]. In general terms, a digital signature should comprise some data that the receiver can keep as evidence that a particular message was sent and that the signer was the originator. It should thus prevent two types of fraud; the forging of a signature by the receiver (or any third party) and the repudiation of the transmission of a message by the sender. Moreover, it must be such that the receiver can save it as evidence which can be presented to a referee who can then check the validity of the signature and settle any dispute.

2.2 Properties of Signatures

If digital signatures are to be as widely used as written signatures and to be accepted for such important tasks as signing legal contracts, then they must have those properties of the written signature that make it such a workable and reliable form of authentication. But what are those basic properties? The crucial properties of a written signature are that it is easy to produce, easy to recognize but difficult to forge. It is this latter property that means that it cannot be repudiated and, as a consequence, written signatures are accepted as providing lasting evidence of precisely what has been authenticated and who authenticated it.

Thus, in an electronic message system we require that each user be able to produce with ease a digital signature whose authenticity is easily checked by a recipient. Moreover, such a digital signature must have the property that it could not have been produced by anyone else and can be used by a referee to resolve disputes.

A digital signature is a digital signal, usually represented as a string of bits. It may be appended to a message or the message may form an integral part of it. Thus, by its very nature, it differs from a written signature. Producing a written signature is a physical process, considered to be intrinsic to the person who is signing, and it is this uniqueness that identifies the signer and means that a signer cannot later repudiate his signature. In contrast, a digital signature is produced by machine. Instead of the signer performing a physical process, the signer merely provides some input to the process and it is this input that determines what particular pattern of bits makes up the digital signature. The aim is that no other person should be able to produce the same pattern of bits which, of course, means that they must not know the signer's input. Thus it is the use of the signer's input that identifies him. The signer's input may be some (secret) information that only he knows or a physical characteristic such as a fingerprint. Since it is the use of the signer's input that identifies him, if the input is secret information then the signature only identifies the signer as someone who knows the secret information. It is clearly possible that someone else may be able to obtain that information and if they do then they will be able to impersonate the signer. If, on the other hand, the signature is dependent on a physical characteristic then a forger must successfully copy that physical characteristic. However, it must not be forgotten that if, during the signing process, that physical characteristic is converted into a sequence of bits then the signer may be able to obtain that bit string without actually needing to copy the physical characteristic.

Another fundamental property of a person's written signature is that it is the same on all documents. Although it may be a difficult task, a forger may, therefore, be able to learn from studying examples of the signature and so duplicate it (without detection).

The security of a written signature lies in the difficulty of producing undetectable forgeries. In the case of a written signature, which is physically attached to a paper document, it is the ability to detect whether or not the document has been doctored that guarantees that the document is one that was signed. On the other hand, since there is no physical way of determining how a digital signal was produced or what input was given and since a digital signal is easily replicated, a digital signature must be different for each message. It is the particular pattern of bits of each individual digital signature that guarantees what message was signed and therefore, to prevent the substitution of an (altered) message to correspond to a signature, the signature must be a function which is dependent on all of the message. A forger, having seen many examples of a person's digital signature, should be no better informed as to how to produce a new (forged) digital signature of another message.

2.3 Signing and Verifying Transformations

Since a digital signature is a message-dependent bit-pattern, the signing process must transform the message into the signed message (or signature). Furthermore, since the signature can only be computed by the sender, this transformation must use some information that is unique to the sender. As we saw in the last section, it is then the use of this secret information that is accepted as proof of the identity of the signer, that is, as establishing the origin of the message. We must now express those concepts more formally.

We will let $\mathcal{M}$ denote the set of all messages which the sender may sign and $\mathcal{S}$ be the set of all possible signatures. The signing process is now nothing more than a transformation from $\mathcal{M}$ to $\mathcal{S}$. If we let S_X be the (secret) signing transformation used by signer X then, for any $M \in \mathcal{M}$, $S_X(M)$ is the signature on message M by user X. This signature $S = S_X(M)$ can then be concatenated to the message M to give (M, S), which we call the signed version of M. If, as is frequently the case, M can be recovered from S then there may be no need to transmit M. In this situation we abuse our notation slightly and regard S as both the signature and the signed version of the message.

Before accepting a message as authentic the recipient must verify the signature. However, since he does not know S_X, the receiver cannot compute $S_X(M)$ from M. Instead, during the verification process he must use public information to check that the user's secret transformation was used. Thus, since the verification process must also authenticate the message, we may view this process as the application of a (public) transformation V_X from $\mathcal{M} \times \mathcal{S}$ to the set {True, False}. (Thus V_X is the transformation that verifies signatures by X.)

Clearly the first requirement of V_X is that it should identify those pairs for which the signature does not "belong" to the message. More formally we require:

1. $V_X(M, S)$ = True if, and only if, $S = S_X(M)$.

However, it is also necessary for the sender to have confidence that, whenever $V_X(M, S)$ = True, the message M originated from X and that the signature is not a forgery. Thus we also require:

2. It is not computationally feasible for anyone, except user X using S_X, to construct $M \in \mathcal{M}$ and $S \in \mathcal{S}$ such that $V_X(M, S)$ = True.

Concern is often expressed about the special case of (2) when someone might be able to change the message without invalidating the signature. It is clearly desirable

to require that it is impossible to find two distinct messages M, M' and a single signature S such that $V_X(M, S) = V_X(M', S) =$ True. This is equivalent to requiring that $S_X(M) \neq S_X(M')$ for all messages $M \neq M'$ and is automatically satisfied if a message is recoverable from the signature. Unfortunately, as we shall see when we discuss specific examples, in many practical situations $|\mathcal{M}| \gg |\mathcal{S}|$ so that this stronger requirement is unachievable. However, condition (2) implies the weaker property that, given signature S on message M, it is computationally infeasible to find another message M' such that $V_X(M', S) =$ True.

Another clearly desirable property is that the probability of guessing (or randomly generating) $S_X(M)$ for a given M should be minimal. This imposes size restrictions on $|\mathcal{S}|$.

Yet another important observation about V_X is that, although V_X can only use public information, it is crucial that knowledge of this public verifying information should not enable anyone to deduce the secret signing transformation S_X or, in any other way, allow false signatures to be generated.

We end this section by noting that if the message were recoverable from the signature then we could have given an alternative definition of the verifying process as one determined by a (public) transformation V_X from $\mathcal{S}$ to $\mathcal{M}$ such that $V_X(S) = M$ belongs to $\mathcal{M}$ if and only if S $= S_X(M)$.

2.4 Initial Agreement and Legality of Signatures

It is important to realize that conditions (1) and (2) are both necessary. The need for condition (1) is obvious. It is condition (2), however, that guarantees that forgery is difficult. Since condition (2) is so crucial each user must agree to the verifying transformation that is to be used to validate his signatures. Furthermore, the receiver requires a guarantee that the verifying transformation he uses to validate a signature is one to which the signer has agreed. Thus, before any signed messages are transmitted, there must be an agreement about what verifying transformation is valid. This agreement may necessarily need to be the subject of a paper document with handwritten signatures.

The sender and recipient must also agree on how disputes are to be resolved. Usually a dispute will be resolved by a third party, the referee. When a dispute arises it usually means that a receiver, user B, has a signature S that he claims to be the message M signed by the sender, user A, while A claims he did not sign M. The essence of a digital signature is that a referee can determine whether B is putting forward an S that he knows A did not generate or whether A is repudiating a signature he has generated. We have already noted that if messages are sent authenticated by a conventional (shared key) cryptosystem, a third party is not able to discover these ruses. (This is, basically, because the verifying transformation V_X involves the use of S_X. Indeed, to determine whether $V_X(M, S)$ is True or False the receiver computes $S_X(M)$ and compares its value with S.)

When B claims that S is the signature of M by A, his claim is presumably based on the fact that $V_A(M, S) =$ True where V_A is the verifying transformation for signatures by user A. Therefore B presents to the referee the evidence S, M, and V_A. The referee verifies for himself whether or not $V_A(M, S) =$ True and thereby determines whether B is making a false claim or A is attempting to repudiate his signature. In such a dispute the referee must also rule on whether V_A is the verifying transformation for

A's signature, agreed to by A and B. This process is also likely to include verifying that M is a valid message, that is, it belongs to the agreed message set $\mathcal{M}$.

Although in some circumstances the referee may be trusted by A and B to resolve disputes fairly in his own way, there are many situations where A and B may have a signed written agreement with the referee on how disputes between them will be resolved. In any case the protocol for resolving disputes must be defined and agreed to by A and B in advance of any messages being signed. Different protocols determine different signature schemes even though the signing and verifying transformations may be the same. This problem is discussed further in Section 2.7.

Whereas both the laws and legal precedent regarding signed paper documents are well established, the situation regarding digital signatures is not so clear. It seems that, at present, if digital signatures are to have any standing in law, then an initial written agreement between the parties concerned, that is, the sender, receiver, and referee, is required. This agreement should define the procedures for obtaining a digital signature, how it would be recorded, what each party's commitment would be and how disputes would be resolved. If one party to this agreement felt that another party had not honored his commitment then he could test this in court.

This may seem unsatisfactory and, as digital signatures become more widely accepted, it is possible that legislation dealing with digital signatures may be enacted. Experience with the use of digital signatures will undoubtedly help determine what form these laws may take. There will, of course, be many problems that need addressing. These include: the means of authenticating the verifying information, the obligation of a user to ensure the security of his signing transformation, the means of resolving disputes, penalties for the misuse of digital signatures, the use of stolen keys, signing false statements or misdating signed messages, and the action required in case of compromised keys. It is likely to be a few years before adequate legislation is introduced.

2.5 Methods for Digital Signatures

Whether or not users can reach an agreement to accept digital signatures will depend largely on their belief that transformations S_X and V_X satisfying conditions (1) and (2) above (see Section 2.3) can be constructed. When the concept of digital signature was first discussed by Diffie and Hellman, [Diffie 76], they suggested a solution to the problem by use of public key cryptosystems. Since then there have been many proposed implementations of digital signatures. Some suggested schemes make use of conventional (or symmetric) key cryptosystems, some of public (or asymmetric) key cryptosystems, and others rely on transformations that are independent of cryptosystems.

Two categories of digital signatures may be identified: true signatures and arbitrated signatures. In a true signature scheme, signed messages are sent directly by the signer to the receiver who verifies their validity. Unless there is a dispute there is no need for the signature to be referred to a third party. The situation for an arbitrated signature scheme is totally different and signed messages can be sent only via a trusted third party called the arbitrator. The recipient is unable to verify the sender's signature directly, but is assured of its validity through the mediation of the arbitrator. The arbitrator is trusted by both parties to play his role correctly and this gives the sender assurance that his signature will not be forged and the receiver assurance that the signatures he receives are valid. The arbitrator also plays the role of referee. He is trusted to resolve disputes (fairly) should they arise.

2.5.1 Arbitrated signatures. Two users, X and Y, who trust each other can use a conventional cryptosystem to exchange authenticated messages. By some secure means they share a secret key k, known only to them, which determines both an enciphering transformation E_k and a deciphering transformation D_k. They trust each other not to reveal this key to any other user. For X to send an authenticated message M to Y, the cryptogram $C = E_k(M)$ is formed and transmitted. User Y decrypts C to obtain $M = D_k(C)$ and checks that M is valid, that is, that M belongs to the agreed message set. If M belongs to the agreed set then Y can be confident that C came from X provided that the probability that someone, who doesn't know k, can find a cryptogram that decrypts to a valid message is small. If this probability is too high then the problem can be overcome by transmitting M concatenated with C, that is, (M, C).

We have already discussed this method of authentication in detail and seen that it does not provide a signature as a third party cannot determine whether X or Y produced $C = E_k(M)$ and cannot resolve a dispute between X and Y about whether X sent message M to Y or not. However, if X and Y both trust a third party A, then this form of authentication may be used to allow X, with the cooperation of A, to send a signed message to Y. In this scheme each user X shares with A, called the arbitrator, a secret key k_X which determines enciphering and deciphering transformations E_{k_x} and D_{k_x}. (This key k_X is known only to X and A.)

One such scheme involves user X sending messages to user Y via A. User X signs message M by encrypting it with key k_X. Thus $S_X(M) = E_{k_x}(M)$. User X sends $E_{k_x}(M)$ to A. The arbitrator A has key k_X and so can decrypt this to obtain M. Provided M is a valid message for user X, A now appends X and M to the signature $E_{k_x}(M)$, encrypts the result with key k_y, and sends $E_{k_y}(X, M, E_{k_x}(M))$ to user Y. User Y decrypts this to obtain X, M, and $E_{k_x}(M)$. Provided the identity X and message M are acceptable to him, Y accepts $E_{k_x}(M)$ as the signature of M by X. Thus $V_X(M, S)$ = True if and only if Y receives $E_{k_y}(X, M, S)$ from A (which occurs if and only if $S = E_{k_x}(M)$). In case of dispute, Y, who claims that he has the signature $S = S_X(M)$, sends $E_{k_y}(X, M, S)$ to A who uses k_Y to obtain X, M and S and then uses k_X to check that $E_{k_x}(M) = S$. Thus in this scheme Y cannot actually check X's signature directly. It is there solely to settle disputes. Thus in this arbitrated digital signature scheme, both users X and Y must trust the arbitrator A. Signatures can be sent only via A and only A can verify signatures and resolve disputes. User X must trust A not to reveal key k_X and not to generate false signatures $E_{k_x}(M)$. Similarly, user Y must trust A to send $E_{k_y}(X, M, S)$ only if $E_{k_x}(M) = S$. Furthermore, both users must trust A to resolve disputes fairly. If the arbitrator acts as expected then X has an assurance that no one can forge his signature and Y has the assurance that X cannot disavow his signature.

In a slight variation of this method, user X sends $E_{k_x}(M)$ directly to Y who then sends $E_{k_y}(X, M, E_{k_x}(M))$ to A to have the signature validated.

There is another arbitrated signature method in which user X and the arbitrator A produce the signature between them. In this scheme the arbitrator A also has another key k_A known only to him. To sign message M, user X sends $E_{k_x}(M)$ to A. Now A uses k_X to obtain M, adjoins the identity X, and enciphers the result with key k_A. This he sends back to X and it is this that forms the signature of M by X, that is, $S_X(M) = E_{k_A}(X, M)$. Now X can send the message M and signature $S = E_{k_A}(X, M)$ to Y. To validate S, user Y sends M and S to A who uses k_A to obtain $D_{k_A}(S)$ which he enciphers with key k_Y. Thus A returns $E_{k_y}(D_{k_A}(S))$ to Y. Now Y uses k_y to obtain $D_{k_A}(S)$ and accepts S as a valid signature only if $D_{k_A}(S) = (X, M)$. Thus $V_X(M, S)$ = True if and only if $D_{k_y}(E_{k_y}(D_{k_A}(S))) = D_{k_A}(S) = (X, M)$, that is, if and only if $E_{k_A}(X, M) = S$. In case of

dispute user Y sends $E_{k_y}(X, M, S)$ to A, who rules the signature valid if and only if $S = E_{k_A}(X, M)$.

In this scheme the assistance of A is required both to generate and to validate signatures and, as before, only A can resolve disputes. Thus both users X and Y must trust A, and if A is trustworthy they have an assurance that signatures cannot be forged nor disavowed.

2.5.2 True signatures. A true signature scheme is one in which the signer sends the signature directly to the receiver who is able to verify the signature without recourse to a third party. In this case the transformation S_X acts as a broadcast cipher and all users can use V_X to verify the origin of a message. However, only one user, namely X, can generate S from M. Diffie and Hellman [Diffie 76] describe this property as one-way authentication.

The verifying transformation V_X allows the receiver to recognize an S as the signature $S = S_X(M)$ on message M by user X, but knowledge of V_X must not allow him to form the signature of a message. In arbitrated signature schemes the receiver needs the cooperation of the arbitrator to perform V_X and it is the trustworthiness of the arbitrator that ensures that user X, and only X, is able to generate signatures. True signature schemes rely on properties of the signing and verifying transformations to provide this assurance.

Clearly this reliance necessitates the use of a special type of function for the signing and verifying transformations. These functions, whose importance was recognized by Diffie and Hellman, are called one-way functions. Their fundamental property is that they are "easy" to implement but "hard" to invert. More precisely, a *one-way function* is a function f such that, given any x in its domain, $y = f(x)$ is easy to evaluate, but, for almost all y (in its range), it is difficult to find an x such that $y = f(x)$. One particularly important family of such functions are the *trapdoor one-way functions*, that is, one-way functions that allow someone in possession of specific secret information (the trapdoor) to compute an inverse.

In fact, in any true signature scheme the verifying transformation V_X determines a function with the properties of a trapdoor one-way function. To see this we define a function $f: \mathcal{M} \times \mathcal{S} \rightarrow \mathcal{M} \cup \{\infty\}$ by $f(M, S) = M$ if $V_X(M, S) =$ True and $f(M, S) = \infty$ otherwise. Then, clearly, f is easy to evaluate since, otherwise, checking signatures would be difficult. However, for any given $M \in \mathcal{M}$ it is difficult to find $(M, S) \in \mathcal{M} \times \mathcal{S}$ such that $f(M, S) = M$ without knowledge of S_X. (This is condition (2) in our requirements for the signing and verifying transformations.) This knowledge of S_X is the trapdoor that enables the signer to compute the inverse.

Several methods for obtaining a digital signature scheme based on one-way functions and their trapdoors have been published. When Diffie and Hellman introduced the concept of a public key cryptosystem based on a trapdoor one-way function, they showed how it may be used to obtain a digital signature scheme. Other asymmetric transformations (with trapdoors), however, and conventional (symmetric key) cryptosystems have also been used to provide digital signature schemes.

2.6 Some General Settings for Signature Schemes

In this section we list a number of general settings for true signature schemes.

According to ISO the term "digital signature" is used "to indicate a particular authentication technique used to establish the origin of a message in order to settle

disputes of what message (if any) was sent.'' Unfortunately there is not always universal agreement on precisely what is acceptable as indisputable evidence for settling such disputes and thus there is often debate as to whether or not a specific proposal constitutes a true digital signature scheme, according to the ISO definition. We do not attempt to act as judge as to whether or not a given scheme warrants the term ''digital signature'' but merely list all those proposals that we know are being seriously considered for implementation.

2.6.1 Public key cryptography. A public key cryptosystem is a multiaccess cipher with the fundamental property that the encryption and decryption transformations are separated. Thus whereas the encryption transformation is public and anyone can encipher a message, decryption is only possible to the holder of the (secret) decryption key.

In a public key cryptosystem each user X has two keys: a public key p_X which determines an enciphering transformation and a private (secret) key S_X which determines a deciphering transformation. A message M may be sent to X in secret by enciphering it as the cryptogram $C = E_{p_x}(M)$. User X can decipher the cryptogram using his deciphering transformation D_{s_x} which must satisfy $D_{s_x}(E_{p_x}(M)) = M$ for all messages M. Both transformations E_{p_x} and D_{s_x} must be easy to perform and it must be computationally infeasible for anyone, from the knowledge of E_{p_x}, to derive D_{s_x}. Thus, with p_X made public, anyone can send to X a secret message that only X can decipher (using D_{s_x}).

The public key cryptosystems considered by Diffie and Hellman that can be used to provide digital signatures are those for which the transformations E_{p_x} and D_{s_x} both act on the same set. Then E_{p_x} and D_{s_x} are inverses of one another and $E_{p_x}(D_{s_x}(M)) = M$ for all messages M. When this is the case, user X may sign a message by forming the signature $S = D_{s_x}(M)$. The sender and receiver agree on a subset $\mathcal{M}$ of all possible messages to be the set of valid messages. The signing transformation S_X is then the secret deciphering transformation D_{s_x} restricted to this set. The verifying transformation V_X is defined by $V_X(M, S)$ = True if and only if $E_{p_x}(S) = M$ and $M \in \mathcal{M}$. Once again we stress that the set $\mathcal{M}$ of valid messages must be chosen so that, unless D_{s_x} is known, it is computationally infeasible to construct M and S such that $V_X(M, S)$ = True. So, for example, for randomly chosen S, the probability that $E_{p_x}(S) \in \mathcal{M}$ must be negligible.

Public key cryptosystems for which the range of the enciphering transformation E_{p_x} is not the set of messages have also been proposed. These systems are not immediately suitable for signature schemes as in such a system $D_{s_x}(M)$ may not be defined for all messages M or $E_{p_x}(D_{s_x}(M))$ may not necessarily be equal to M. However, Merkle and Hellman [Merkle 78] and Schöbi and Massey [Schöbi 83] have shown that, by suitable modifications, a signature scheme may be obtained from certain of these public key cryptosystems provided $E_{p_x}(D_{s_x}(M)) = M$ for a suitable proportion of the set of messages M. In these signature schemes the signature of a message M is obtained by applying D_{s_x} to a message M' obtained from M (in a defined way) such that $E_{p_x}(D_{s_x}(M')) = M'$. Thus $S_X(M) = D_{s_x}(M')$ and $V_X(M, S)$ = True if and only if $E_{p_x}(S) = M'$.

2.6.2 Other trapdoor signature schemes. A public key cryptosystem has two transformations E_{p_x} and D_{s_x} satisfying $D_{s_x}(E_{p_x}(M)) = M$ for all messages M. Furthermore it is not possible to determine the secret deciphering transformation D_{s_x} from knowledge of the public enciphering transformation E_{p_x}. Suppose instead that we have

two transformations E_{p_x}, D_{s_x} satisfying $D_{s_x}(E_{p_x}(M)) = M$ for all messages M with the extra property that it is not possible to determine E_{p_x} from knowledge of D_{s_x}. Then a signature scheme may be obtained by making the key S_X public while keeping the key p_X secret. Thus the signature by X on message M is $S_X(M) = E_{p_x}(M)$ and the signature S is verified by checking that $D_{s_x}(S) = M$ is a valid message.

Thus, given transformations f, g satisfying $f(g(M)) = M$ for all messages $M \in \mathcal{M}$, we obtain a public key cryptosystem if g is a one-way function for which f provides the trapdoor information (and then $E_{p_x} = f$ and $D_{s_x} = g$) and we obtain a signature scheme if f is a one-way function for which g provides the trapdoor information (and then $S_X = f$ and $V_X(M, S) =$ True if and only if $g(S) = M$ is a valid message). A public key cryptosystem may or may not be suitable for a signature scheme and similarly a signature scheme may exist independently of any public key cryptosystem.

2.6.3 Symmetric key cryptography. In a conventional (symmetric key) cryptosystem both the encryption transformation E_k and the decryption transformation D_k are determined by the same key k. Then $D_k(E_k(M)) = M$ for all messages. However, the transformation E_k is not a one-way function and if made public would make known the deciphering transformation D_k. For such a cryptosystem to be secure against a known plaintext attack it must be difficult to determine the key k given pairs of corresponding plaintext M and ciphertext $C = E_k(M)$. If this is so, then we may use the enciphering transformation to define a one-way function acting on the set of keys k. Fix a message X and define f_X by $f_X(k) = E_k(X)$ for all keys k. Provided the cryptosystem is secure against a known plaintext attack, f_X is a one-way function. This one-way function can be used to obtain a signature scheme provided it is possible to find a suitable trapdoor as the signing transformation S_X.

For most conventional cryptosystems the one-way function f_X may not have a trapdoor. However when a one-way function f is used to obtain a signature scheme, the trapdoor information is (merely) the means by which the signer finds a solution for S to an equation $f(S) = h(M)$ for some M. If two tables of pairs are constructed, one containing the pairs (M, S) and the other $(M, f(S))$, this solution can be found by comparing entries in the two tables with the same first entry M. Thus the table of pairs $(M, f(S))$ can be made public and used to evaluate $h(M)(= f(S))$ and the table of pairs (M, S) kept secret and used to produce the signature $S = S_X(M)$ by X on message M.

This is the approach taken in the Diffie–Lamport signature scheme [Diffie 76]. In its simplest form this scheme has a set of two messages $\{M_1, M_2\}$, a secret table containing pairs $(M_1, k_1),(M_2, k_2)$, and a public table containing pairs $(M_1, K_1),(M_2, K_2)$ and public one-way functions f_1, f_2 such that $f_1(k_1) = K_1$ and $f_2(k_2) = K_2$ (e.g., $f_i(k) = E_k(M_i)$.

This use of tables shows one way in which a conventional cryptosystem, secure against a known plaintext attack, may be used to provide a signature scheme. The method has the drawback that the number of possible messages must be small enough for the tables to be manageable. If the number of messages that can be signed is too small then this restriction can be overcome by representing long messages as a sequence of messages from the small set and signing each member of the sequence. To prevent deletion or replay of signatures of messages that have appeared in a sequence before, however, the tables of pairs (i.e., the one-way functions f_1, f_2) must be continually changing. In any scheme where the number of messages is small it is always feasible to keep a log of previous signatures and then replay them (in a different order) as a forged

message at a later time. Signatures obtained using this type of tables are therefore called one-time signatures, since the tables must be continually being used and discarded. They cannot be used again.

To sign a message of n bits these one-time signatures require of the order of $2n$ bits of verification data to be made available in the tables and so, if a great many messages are to be signed, there may be difficulties in storing, handling, and authenticating this verification data. Merkle [Merkle90] has suggested ways in which the verification data may be generated (and authenticated) in a tree structure as the messages are signed. The storage requirement is reduced to the storage of the root of the tree. The essential idea is to use a succession of one-way functions (and corresponding tables) to successively authenticate the (next) tables and finally sign the message. The length of the resulting signature increases proportionally to the log of the number of messages signed (see Section 3.4).

2.6.4 Signatures by tamper-resistant modules. It is the fact that there is a (natural) separation between encryption and decryption in a public key cryptosystem that makes such systems candidates for digital signature schemes. Since each of the two functions has its own key it is possible to be able to verify a signature (using encryption) without being able to generate a signature (which requires decryption). For conventional systems there is not the same (natural) separation of keys. However, if the encryption and decryption transformations were successfully separated then it would be possible to provide signatures using such a system. One way of achieving this separation is by physical means using tamper-resistant modules (TRMs).

Suppose we have a system where each user has a TRM that contains a key k_X for a conventional cryptosystem and that k_X determines an encryption transformation E_{k_x} and decryption transformation D_{k_x}. Suppose the key k_X could be communicated securely to all other TRMs and the modules were constructed so that, while only X's TRM could perform E_{k_x}, all other TRMs could only perform D_{k_x}. Then, provided that the tamper resistance of each module could be trusted, only X's TRM could generate a signature $S_X(M) = E_{k_x}(M)$ but any other TRM could verify a signature by computing $D_{k_x}(S)$ and checking whether or not $D_{k_x}(S) = M$.

It should be noted that, although this is the first example where it is explicitly stated that the security relies on tamper resistance, most digital signature schemes rely on some form of physical security. If, for example, we consider a scheme based on public key cryptography then the security relies on the sender X's ability to keep his secret key S_X secret. Great care is taken to ensure that S_X cannot be computed from knowledge of the algorithm and X's public key p_X. However, this is not sufficient and X must also take steps to ensure that S_X is not exposed by physical means. Typical ways of achieving this are to use tamper-resistant devices and/or to store all devices that hold s_X in physically secure locations.

2.6.5 Composition of trapdoor permutations. In this case we assume that we have two trapdoor one-way permutations, f_0 and f_1, on the set of all users, and that messages are represented by binary strings. If $M = M_1M_2 \ldots M_n$ is a sequence of 0's and 1's then the signature of X on M is

$$S_X(M) = f_{M_n}^{-1}(f_{M_{(n-1)}}^{-1}(\ldots (f_{M_1}^{-1}(X)) \ldots))$$

where f_0^{-1} and f_1^{-1} are the (trapdoor) inverses of f_0 and f_1, respectively, and are only known by user X. The rule for checking the signature should be clear, that is, $V_X(M, S)$ = True if and only if $X = f_{M_1}(f_{M_2}(\ldots (f_{M_n}(S) \ldots))$.

Note that while computing $V_X(M, S)$ the receiver produces signatures for the messages $Mi = M_1, M_2 \ldots M_i$ for $i = 1, \ldots, n - 1$. Thus, this scheme can only be used if, for any $i < n$, the sequence Mi is not a valid message.

2.6.6 Schemes with implicit verification functions. So far we have considered signature schemes in which $V_X(M, S)$ = True if and only if $f(S) = g(M)$ where f is a one-way function and g is easily computed. More generally it may be preferable to consider verifying transformations defined by $V_X(M, S)$ = True if and only if $h(M, S) = 0$ for some implicit function h of M and S. Gibson [Gibson 88b] discusses an example modeled on the Fiat–Shamir scheme [Fiat 87] where an implicit function arises as a result of the introduction of a random element and the use of a family of trapdoor one-way functions.

For this signature scheme we use a publicly known one-way function h and a family of trapdoor one-way functions indexed by elements $e \in E$. User X has private functions g_e which provide the trapdoor inverses to the functions f_e. He signs message M in the following way.

X chooses a random element r and calculates $e = h(M, r)$ and $y = g_e(r)$. His signature $S_X(M)$ is then given by $S_X(M) = (y, e)$ which, clearly, depends on r. The receiver may verify the signature $S = (y, e)$ by checking that $h(M, f_e(y) = e$.

2.6.7 Scheme with probabilistic verification. In arbitrated signatures the signer shares his secret signing information with the arbitrator. This allows the arbitrator to verify a signature and pass on this assurance to the recipient. Rabin [Rabin 78] has described a signature scheme in which, after signing a message, the signer reveals only part of his secret signing information to the recipient. This allows him to have confidence in the signature. However, only in the case of dispute does the signer reveal all his secret signing information to an arbitrator who can then verify the signature. The receiver's verification is a probabilistic verification in the sense that there is a (small) probability that a signature verified by the receiver will not be validated by the arbitrator.

Any signature scheme of this type must be a one-time signature scheme in the sense that the signer's secret information can be used to sign only one message. Thus the scheme must have a register of validating information used in the verifying process to validate the signer's revealed secret signing information.

In Rabin's scheme a user wishing to sign a message M uses an even number, say $2t$, of one-way functions $f_1, \ldots, f_{2t}$. (The f_i could, for example, arise from a conventional cipher system as enciphering transformations E_{k_i} for some choice of keys $k_1, \ldots, k_{2t}$.) The functions $f_1, \ldots, f_{2t}$ are the signer's secret signing information and, in order to enable verification of their use in the signing process, pairs (x_i, y_i) where $y_i = f_i(x_i)$ are published in a register whose authenticity is accepted by all parties. The signature on message M is then the tuple $(f_1(M), \ldots, f_{2t}(M))$.

To validate this signature the recipient chooses a subset I of t of the subscripts $1, \ldots, 2t$ and asks the signer to reveal f_i for $i \in I$. The recipient verifies that the revealed f_i are part of the signer's secret signing information by consulting the register of pairs (x_i, y_i) and then partially checks the signature $(S_1, \ldots, S_{2t})$ on message M by

checking that $S_i = f_i(M)$ for $i \in I$. The recipient accepts the signature $(S_1, \ldots, S_{2t})$ if and only if all these checks are satisfied.

In case of dispute the recipient presents the message M and signature $(S_1, \ldots, S_{2t})$ and the signer reveals all of $f_1, \ldots, f_{2t}$ to an arbitrator. The arbitrator verifies that $f_1, \ldots, f_{2t}$ are the secret signing functions by checking that $f_i(x_i) = y_i$ for $i = 1, \ldots, 2t$ and validates $(S_1, \ldots, S_{2t})$ as the signature on M provided $S_i = f_i(M)$ for at least $t + 1$ functions f_i.

The problem of determining the precise number of pairs (f_i, S_i) to be checked by the arbitrator is interesting. If, for instance, we insisted that he checked all $2t$ of them then the sender might provide a "signature" with $2t - 1$ correct S_i and one incorrect one. This signature would be guaranteed to be declared invalid by the arbitrator but the chances of the receiver choosing t correct values, and thereby accepting the signature, would be high. If, on the other hand, the arbitrator checks only $t + 1$ then the signer might be able to create a "signature" with exactly t correct values which would be accepted by the receiver but, of course, rejected by the judge. However, the chance that the receiver would request these t functions, and so accept this signature (which an arbitrator would not validate) is only one in $\binom{2t}{t}$. For example, when $t = 18$, this results in only one chance in $\binom{36}{18} \sim 10^{10}$.

It is important to reemphasize that this signature scheme is a one-time signature scheme. The pairs (x_i, y_i) can only be used to sign one message because any repeated pair for which f_i was revealed could be used to forge a signature.

Note also that, since in case of dispute the signer shares all of his secret signing information with the arbitrator, it is possible for the arbitrator, after resolving a dispute, to forge a signature. The signer must therefore trust the arbitrator not to use this information or give it to anyone else.

2.7 Commitments and Disputes

We have already stressed the care needed to set up a signature scheme in which all the participants can have confidence. The basis of this confidence lies in the two transformations S_X and V_X. If they satisfy properties (1) and (2), above (see Section 2.3) then user X may feel happy about being committed to any message M for which a receiver holds the signature S such that $V_X(M, S) =$ True, knowing that the only (feasible) way for the receiver to obtain S is from X himself. User X, of course, must also be satisfied that no one else can discover the transformation S_X. It follows that S_X must be held securely. Provided user X is assured that no other person has access to the transformation S_X he can feel sure that no one can produce (forge) his signature. He may then be willing to commit himself to a signature S on a message such that $V_X(M, S) =$ True.

If a receiver has a (signed, written) agreement with X or some other trusted assurance that X is committed to signatures S verified by V_X then he may be happy to accept these signatures. Of course, the receiver may also require an assurance that the signer will be held to that commitment.

Unfortunately, no matter how much care is taken in setting up the scheme, it is still possible that a sender and receiver may end up in dispute. A receiver may (falsely) claim that S is the signature on message M by user X when no such message was signed by X. Similarly, a sender may (falsely) deny signing message M when in fact he did. In such cases the sender and/or receiver might refer their dispute to a referee for resolution. As we stressed in Section 2.4, it is crucial that both sender and receiver know [and

agree (in writing)] the means which the referee will use to resolve a dispute. Furthermore, they must both trust the referee or have some commitment from the referee to abide by the agreed method of resolution.

2.7.1 Resolution of disputes. When the sender and receiver seek resolution of a dispute, the receiver sends to the referee X, M, and S, claiming that S is the signature on message M by user X. The referee requires some way of determining whether S is a valid signature on M by user X. One method of resolution might be for the arbiter to use the verifying transformation V_X agreed to by user X. Thus the referee rules in favor of the receiver if $V_X(M, S) = \text{True}$.

The receiver will be happy with this method of resolution provided he is assured that he has access to the same verifying transformation V_X as the referee uses. The sender will be happy with this method of resolution provided that he is confident that the referee uses the agreed verifying transformation V_X, that V_X is a (trapdoor) one-way function, and that S_X has not been compromised (i.e., that he (user X) is the only person, except possibly the referee, with access to S_X). Both sender and receiver must then rely on the referee not to make a false ruling but to rule according to the outcome of $V_X(M, S)$.

Thus we have identified four necessary properties of a signature scheme for the resolution of disputes:

1. V_X and S_X have properties (1) and (2) (see Section 2.3).
2. V_X is authentic.
3. The referee is trustworthy.
4. S_X is secure.

Where a proof that a signature scheme satisfies (1) is not available, the validity of (1) is usually claimed by the scheme's successful resistance to cryptanalysis. Of course, while there is an incentive to do so, new cryptanalytic attacks will be sought which may provide the means of breaking a scheme. So this may be a volatile property and a scheme's security status may change with time.

Establishing (2) may be difficult and, indeed, the means of authenticating V_X may depend on the nature of V_X.

In an arbitrated signature scheme in which the receiver (and sender) must trust the arbitrator to verify a signature, it seems reasonable to assume that property (2) holds if property (3) holds. Of course, it would be the duty of a responsible trusted arbitrator to ensure that the verifying transformation V_X he uses is the one agreed to by user X (and has not been tampered with by anyone else).

In a true signature scheme, where more than one person requires access to (the same) V_X, there must be some formally agreed-on method of authenticating verification functions. Prior to setting up the scheme, user X must have agreed to the verifying transformation V_X and, in the final analysis, this may require some signed written document that details how to derive V_X. Furthermore, to make it widely available, it may be necessary to hold details of V_X in some trusted register. It may be possible for the referee to get a copy of V_X signed by the registry. This signed copy of V_X is often called a certificate and the registry is trusted to issue only valid certificates, that is, certificates containing only true copies of V_X. Of course, this creates a further requirement that the registry uses a secure signature scheme and keeps its signing

transformation secret. However, the verifying transformation of the registry may be authenticated by wide publication (e.g., in the daily press) so that everyone can be reasonably sure of its validity.

Property (3) is essential—without a referee trusted by both sender and receiver there seems to be no possibility that a dispute can be resolved by a third party.

The method of resolution of a dispute described above makes it essential (for the sender) that S_X is kept secure. If the referee upholds any signature S that is verified by V_X then, in addition to being sure that it is infeasible for anyone to find S such that $V_X(M, S)$ = True for some message M without using S_X, the user must be confident that no one but himself has access to S_X.

We have already discussed the use of one-way functions to prevent anyone from calculating S_X. However, this is not sufficient and property (4), the security of S_X, will almost certainly also require physical security. One option is for user X to be the only person with physical access to the machine that performs S_X. Another is that some physical characteristic of user X that cannot be duplicated (e.g., a fingerprint) may be a necessary input before S_X can be performed. Alternatively, the security of S_X may be obtained by making something (a secret) that user X knows, but no one else does, a necessary input before S_X can be performed.

In the case of physical security user X would need to be confident that it was physically impossible for anyone else to perform S_X either due to lack of access to the machine or due to lack of the appropriate physical characteristic that was needed to activate the machine to perform S_X. In this latter case the machine would have to be tamper-resistant so that it is impossible to discover how to alter the machine or build another machine to perform S_X without the input of user X.

Similarly, when the input is some secret information, user X would want an assurance that no one else has access to the secret input that he (alone) knows. Secret information, however, can be lost or stolen and so, as the holder of this information, it may be the duty of user X to ensure its secrecy. As an "insurance policy" against compromise of this secret information, X might insist on being able to ensure that no more signatures be verified with V_X, that is, he might insist that V_X be withdrawn from the authentication registry. Unfortunately this type of requirement causes new problems. One obvious problem arises from the fact that a signature S may need to be verified long after the message was signed. Thus signatures signed before S_X was compromised may still need to be verified long after that compromise. This would not be possible if V_X were withdrawn.

Another problem, perhaps of even greater importance, arises from the fact that if V_X is withdrawn then no earlier signature of X can be verified. This is a fact that X may exploit for fraudulent purposes. Indeed, by falsely declaring the loss of his secret information or by revealing this information, user X could revoke messages he has signed. Clearly, signatures that could be repudiated in this way by the sender would not be acceptable to a recipient.

It seems reasonable to try to insist that if a signature could be identified as having been signed before S_X was compromised then V_X could still be used to verify it. However, there are practical difficulties with this notion. For instance, it is usually difficult to determine exactly when S_X may have been compromised. Even if we ignore this problem, V_X must now be authenticated as valid at a certain date and time. Furthermore a means, independent of S_X, must be used to authenticate the date and time of each signature to verify whether or not it was signed before or after the compromise of S_X.

The "obvious solution" of merely adding the date and time to a message and signing the resulting message using S_X is not sufficient since anyone who obtains S_X can predate a message and then sign it. Thus the receiver must obtain a time-stamp on the signature $S = S_X(M)$ from a third party. (This third party may be a registered timekeeper who adds the date and time to a signature S presented to him and signs the result to produce a signature certificate.) In addition, the receiver must also obtain from the registry of verifying transformations an authenticated copy of V_X which has been time-stamped by the timekeeper. The receiver would then check each message and would not accept a signature time-stamped with a date and time that did not precede that stamped on V_X. Of course, in this scenario the certificates issued by the registry and timekeeper need to be verified, but this can be achieved by using their respective verifying transformations which, in turn, can be authenticated by being published widely (e.g., in the daily press).

As soon as the "machinery" discussed above is established then we can determine our procedures for reporting compromises of secret information. If user X discovers (or suspects) that his signing transformation has been compromised, he notifies the registry. If this notification is signed using S_X, the registry knows that either the notification has come from user X or that, in any case, S_X has been compromised. Thus it can safely accept it as genuine. The registry then issues X with a certificate, time-stamped by the timekeeper, containing the statement that the compromise of S_X had been reported. User X could then repudiate any signatures that were verified by a verifying transformation which was time-stamped after that time.

In the case of dispute a referee would hold user X responsible for a signature only if (i) the receiver had certificates that verified that the signature was time-stamped at a date and time that preceded that at which the verifying transformation was time-stamped and validated; and (ii) the signer could not produce a certificate that verified that his signing transformation had been reported compromised at a date and time prior to the date and time stamped on the signature.

If (i) does not hold the referee dismisses the receiver's claim. If (i) holds but (ii) does not then the referee rules that the registry has issued a certificate recording the compromise of S_X with a date and time that precedes the date and time of a certificate that validates the verifying transformation V_X and holds the registry responsible.

Thus this protocol allows the signer to repudiate signatures when his secret information has been compromised by shifting the responsibility to the registry which issues the certificates. Now, however, the signer is not able to use this for fraudulent means, as the receiver is protected by having certificates with the signature and verifying transformation time-stamped and presumably the registry will protect itself by not issuing certificates for verifying transformations that are time-stamped with a date and time that does not precede the date and time stamped on a certificate that records the compromise of S_X.

The registry must also protect itself by safeguarding its signing transformation and keeping it secret so that no one can issue a forged certificate. The responsibility of user X to keep S_X secret has been shifted to the responsibility of the registry to keep its secret signing information secret. But, in view of their relative roles, this is probably more realistic. Note, however, that, to allow user X to repudiate signatures when S_X has been compromised, it has now become imperative to the registry that its signing transformation is kept secret. The reason is, quite simply, that the registry is likely to be held responsible for any signatures verified using forged certificates.

A disadvantage of this form of signature scheme is the fact that the recipient must communicate with the registry and timekeeper at the time of receiving the signature. However, unlike an arbitrated signature scheme, the recipient does not need to trust the intermediaries (the registry and timekeeper) as he can verify for himself whether a signature he obtains via the certificates will be upheld by the referee (assuming that he can verify the signatures of the registry and timekeeper). On the other hand, the signer does need to trust the registry and timekeeper not to issue certificates that are time-stamped with a date and time that precede the date and time at which the signer reported the compromise of his signing transformation. The signer must also trust the registry and timekeeper to keep their respective signing transformation secure since if they are also compromised then forged signatures and certificates may be generated. It is sufficient that the signer trust the timekeeper, as the registry's signing transformation alone is not enough to generate the required certificates. Conversely, it is necessary that the signer trust the timekeeper, since, with the aid of a copy of V_X time-stamped and authenticated before S_X has been reported compromised, the timekeeper could predate forged signatures to produce certificates which would be upheld by the referee.

Denning [Denning 83] describes a method of protection against the predating of forged signatures. This method involves the timekeeper keeping a log of all time-stamped certificates in order of issue on a write-once device. Only certificates appearing in the log would be valid. When user X reports the compromise of his signing transformation this is also recorded in the log. Everyone would have open read access to the log to ensure this. As the device cannot be overwritten or erased, the predating of signature certificates before the date of notification of the compromise would be detected. Of course, some means of maintaining the security and authenticity of the log is required. It seems that this would be best achieved by some suitable physical means.

2.7.2 Witnessed digital signatures. The use of witnesses can also be used to relieve the sender of some of the burden of keeping his signing transformation secure. When user X signs message M, a witness, user W, physically confirms that user X agrees to message M and signs the message M' = "User X agrees to M." The pair $S = S_X(M)$, $S' = S_W(M')$ are sent to the receiver and form the signature. The signature is verified using V_X and V_W (which, of course, must be authenticated). Thus the recipient (and the referee in case of dispute) accepts the pair S, S' as the signature on M by user X if and only if $V_X(M, S)$ = True and $V_W(M', S')$ = True.

The sender and receiver must agree in advance on which user may be an acceptable witness, and the referee must be able to verify this agreement. Now, if the signing transformation of user X is compromised, forged signatures still cannot be generated unless the signing transformation of a witness is also compromised. Of course the signer must trust the witness not to witness any signature that the signer has not produced.

2.8 Practical Use of Signature Schemes

In the definition of signature given above, it is assumed that the message to be signed is an element of a message space that is acted on by a signature function S_X. In practice, signature functions will take as input strings of bits of fixed length, and messages of any greater length will need to be processed in some way prior to signature. One possibility would be to divide the message into a sequence of "blocks" of appropriate size for the signature function, and then sign each piece individually, that is, if the message M is made up of the sequence

$$M_1, M_2, \ldots, M_t$$

where each M_i has the required length, then user X would sign the message by computing:

$$S_X(M_1), S_X(M_2), \ldots, S_X(M_t).$$

Unfortunately, this method has a number of disadvantages. The first, and probably most important, is that there is no linkage between different parts of the message, and so the recipient of a signed message will not know if the message components (M_i) have been reordered, replicated, or partially deleted during transmission. This can be remedied by the inclusion of redundancy in the message blocks, but this has the disadvantageous effect of increasing the number of signatures to be computed.

Second, even if the message only consists of one block, there is a problem caused by the public nature of the signature checking process. Suppose Y, a would-be forger of X's signature, knows X's public verification function V_X. Then Y chooses a value S at random and computes

$$M = V_X(S)$$

Then S will be a valid signature on M, although Y has no control on the contents of M. This problem, which we first encountered in Section 2.6.1, can be removed by the addition of redundancy to the message to be signed. However, this would reduce the efficiency of the system still further.

A third important defect is that the above process requires the signature algorithm to be applied t times. Many proposed signature functions are relatively difficult to compute (i.e., they run slowly) and therefore this could be very inefficient.

These deficiencies are sufficient to deter the use of this type of signature and, as a result, Rabin [Rabin 78], [Rabin 79], Merkle [Merkle 79], and Davies and Price [Davies 80] introduced the concept of a "one-way hash function." A one-way hash function is a public function h (which should be simple and fast to compute) that satisfies three main properties:

- $H1$. It must be capable of converting a message M of arbitrary length into a fixed-length "digest," $h(M)$, which has the length required to be input to the secret signature transformation.
- $H2$. It must be *one-way*, that is, given an arbitrary value y in the domain of h, it must be computationally infeasible to find a message M such that $h(M) = y$.
- $H3$. It must be *collision-free*, that is, it must be computationally infeasible to construct two messages M, M' with the property that $h(M) = h(M')$. (As we discuss below, $H3$ implies that h should normally be chosen to have a domain of size at least 2^{100}.)

Given an appropriate hash function h then, in order to compute a signature on a message M, user X first computes the digest (or "hashed version") $h(M)$ and then signs this digest using his secret transformation S_X. Thus the signed digest for the message M is:

$$S_X(h(M))$$

X then transmits M concatenated with the signed digest $S_X(h(M))$. Note that a further motive for using one-way hash functions is provided by Denning [Denning 84]. Briefly, if a signature scheme is used without first applying a hash function, and if a cryptanalyst can persuade a user to sign certain selected messages, then in some cases signatures can be forged on other messages.

The verification of this signature requires both prior knowledge of the public hash function h and X's public transformation V_X. Using this knowledge $h(M)$ and $V_X(h(M), S)$ can be computed. If $V_X(h(M), S)$ = True then the message is accepted as valid.

In the discussion above we introduce three properties ($H1$, $H2$, $H3$) that a hash function should have; we now show why these are necessary. The need for $H1$ should be clear. $H2$ is present to prevent the interceptor of a message and its signature (M and $S_X(h(M))$ say) replacing the valid message M with a fraudulent one M' with the property that $h(M) = h(M')$. The need for $H3$ is less obvious, as is the fact that $H3$ requires the domain of h to be so large. Before discussing this point further we observe that, in practice, $H3$ implies $H2$.

To demonstrate the need for $H3$ we will assume that we have a hash function h that does not satisfy $H3$. If this is so then it may be possible for a malicious party, say Z, to construct two messages M, M' with the following properties:

1. $h(M) = h(M')$
2. User X would be happy to sign M.
3. User X would not be happy to sign M' but Z would like to obtain X's signature on this message.

Z could then persuade X to sign M and send this signed message to some third party, that is, X sends M accompanied by $S_X(h(M))$. It is then only necessary for Z to attach this signature to M' to achieve his fraudulent aim.

Having seen why collision freedom is so important, we now indicate why a hash function with a domain of size significantly less than 2^{100} cannot be collision-free. In general, if the hash function has a domain of size 2^n, then it is only necessary to generate approximately $2^{n/2}$ variants of each message to have a reasonable chance of finding two variants with the same digest. (This is the so-called "birthday attack" and is discussed in detail in Section 4.1.2.) Thus, to make such an approach completely impractical requires n to be large and $n = 100$ has been suggested as a practical minimum, giving a suitable margin for error.

An alternative approach, allowing the use of 64-bit hash functions, has been proposed by Davies and Price [Davies 84]. They suggest that the originator of a message should always modify it in some way prior to signature, typically by appending a random value to the message. Unfortunately, as pointed out by Akl [Akl 84] and Mitchell et al. [Mitchell 89], this procedure does not prevent potential frauds by the originator, and it would seem prudent to avoid use of any hash functions producing digests of less than 100 bits when producing a digital signature. However, we must point out that hash functions producing 64-bit digests are still of value for other types of authentication function.

Having given three properties that a hash function must satisfy, (i.e., $H1$–$H3$, above), we observe that satisfying all three properties does not mean that a hash func-

tion can be used in a secure way with all signature algorithms. In certain circumstances apparently strong signature and hashing algorithms, when used together, reveal weaknesses. This is precisely what occurs when the standard RSA signature technique is combined with the X.509 hash function, as has recently been shown by Coppersmith [Coppersmith 1989]. (This topic is discussed further in Section 4.2.2.)

In all our discussions so far we have assumed that the message M is sent in "clear" form along with the signature $S_X(h(M))$. However, there may be situations where secrecy is also required. Of course, if M is confidential then it can be encrypted before transmission. But it is important to note that this encryption should always take place after the signature process, that is, signatures should always be computed on "clear" data. This is not only established good practice (see, e.g., Davies and Price [Davies 84],) but also of great significance in protecting against various subtle forms of attack. This is exemplified by weaknesses recently revealed in certain of the security facilities in the 1988 CCITT X.400 and X.500 recommendations, in which construction of "tokens" can involve signing encrypted data; these standards and the attacks on them are discussed further in Section 5.

A second question that naturally arises, when considering encryption of signed messages, is whether or not the signature itself needs to be encrypted. In general this depends on the entropy of the message space. If there are a very large number of possible messages, all with small associated probabilities of occurrence, then the signature does not need to be encrypted. However, if the number of possible messages is small, or a certain small number of messages are significantly more probable than the others, and the signature is unencrypted, then the value of an intercepted signed message could be revealed by an interceptor hashing each possible (or each likely) message in turn until one is found which matches the signature. The only alternative to encrypting the signature in this case is for all messages to be expanded by the addition of random padding prior to signature and encryption. This latter process may be mandated where standards dictate unencrypted signatures (e.g., the International Telegraph and Telephone Consultative Committee (CCITT) X.500 series recommendations—see Section 5).

Apart from their use with digital signatures, cryptographic hash functions can be used to provide authentication and integrity protection for sets of data in other ways. For example, data files on a computer can be protected against change by computing a digest and then storing this digest in a secure place. In this case, property $H3$ would not normally be necessary, and so a 64-bit digest would suffice.

Alternatively, a message digest could be appended to a message prior to encryption to provide origin verification and integrity protection for the message; again, in this case a 64-bit digest would normally suffice. Generalizing this application to the case where a message is to be sent to more than one recipient, much cryptoprocessing can be saved by computing a single message digest and then encrypting it differently for each intended recipient (see, e.g., the Internet electronic mail scheme [RFC1113], [RFC1114]). In this latter application $H3$ is necessary to prevent certain kinds of attack (see, e.g., [Mitchell 90]).

We conclude this general discussion of hash functions by considering *keyed* hash functions, also called *parameterized* hash functions by Merkle [Merkle 89a], [Merkle 89b]. In all the above discussion we have considered hash functions that are completely public, that is, no secret knowledge is required to compute a message digest. However, in some applications it may be desirable to use a hash function that requires use of a secret key to compute the digest for a message. Of course, this means

that the digital signature can only be checked by someone knowing the appropriate secret key, but it will also help prevent attacks by malicious third parties without knowledge of the secret key.

For example, we noted above that if the number of possible messages is small and message confidentiality is required, then an unencrypted signature could reveal the message; use of a keyed hashing function would prevent this. In addition, where straightforward origin authentication and integrity protection is required, a keyed message digest can often be used on its own without signature to provide the desired service. This is the basis of the standardized message authentication codes (MACs) based on the Data Encryption Standard (DES) block cipher [ANSI X9.9], [ANSI 9.19]; these functions were mentioned in Section 2.1 and are discussed further in Section 4.

3 TECHNIQUES FOR DIGITAL SIGNATURES

We have already seen the importance of one-way functions for digital signature schemes and how trapdoors are exploited to achieve the objective that, although it is computationally infeasible (without knowledge of S_X) to find a pair M and S with $V_X(M, S)$ = True, it must be easy to check whether $V_X(M, S)$ is True or False.

In Section 3.1 we discuss some of the one-way functions that have assumed central roles in modern cryptography. Then in Section 3.2 we discuss how they may be used for digital signature schemes.

3.1 One-Way Functions

3.1.1 Multiplication of two large primes. If p and q are large primes then computing their product $n = pq$ is easy. However, unless at least one of the primes has special properties that facilitate factorization, it is extremely difficult to determine p and q from n. Within the last decade considerable resources have been directed toward the factorization problem and considerable progress has been made. A number of general-purpose factoring algorithms have been discovered. The most practical one seems to be the multiple polynomial quadratic sieve (MPQS) [Silverman 87] which has an expected running time: $L(n) = \exp((1 + 0(1))(\ln(n)\ln(\ln(n)))^{1/2})$, independent of the size of the factors of n. This algorithm is well suited to parallel implementation and has been used to factor 102- and 106-decimal digit numbers in about 1 and 4 months, respectively.

In addition to the general-purpose factoring algorithms, there are a number of other algorithms that are much faster but which only "work" if either n itself or (at least one of) the factors of n have special properties. The most recent of these algorithms, the number field sieve [Lenstra 89], is particularly interesting and warrants discussion.

The number field sieve (NFS) algorithm factors Cunningham numbers, that is, numbers of the form $r^e \pm s$ with r and s small, in heuristic expected running time:

$$T(n, c) = \exp(c + 0(1)\ln(n)^{1/3}\ln(\ln(n))^{2/3}) \text{ with } c \cong 1.526.$$

This is asymptotically faster than any other known algorithm applicable to numbers of this form and it has been used to factor several numbers with more than 100

digits in a matter of weeks*. Work is in progress on generalizations of the number field sieve algorithm, applicable to general integers. It is suspected that these algorithms will have expected running time $T(n, c)$ with a larger value of c. It remains to be seen whether c will be small enough to give a general-purpose factoring algorithm that is significantly faster than MPQS for numbers of the size likely to be used in digital signature schemes.

3.1.2 Exponentiation modulo $n = pq$. If $n = pq$ is the product of two large primes and e is chosen so that $(e, (p-1)(q-1)) = 1$, then the modular exponentiation function E, defined by $E(m) = m^e \pmod n$, is a trapdoor one-way function. If we are given a value c such that $c = m^e \pmod n$ for some unknown m then the only known practical method of determining m is to use the exponent d with $ed = 1 \pmod{(p-1)(q-1)}$ and calculate $c^d \pmod n$. (The fact that $m = c^d \pmod n$ is, of course, the basis of the RSA public key system, see [Beker 82]). However, the determination of d from n and e requires knowledge of the factors p and q and it is, therefore, this knowledge that provides the trapdoor. It is also true that knowledge of one pair of values for e and d is sufficient to determine p and q, that is, to factor n. Thus inverting E without the trapdoor appears to be as difficult as factoring n.

3.1.3 Exponentiation in a finite field. For any prime power q the multiplicative group of the finite field $GF(q)$ of order q is cyclic. If q is a large prime power and a is a primitive element of $GF(q)$ (i.e., a generator of the cyclic multiplicative group) then for any nonzero element y in $GF(q)$, $y = a^x$ with $0 \leq x \leq q-1$.

The integer x is called the discrete logarithm of y to the base a in $GF(q)$. When q is a prime $GF(q)$ is isomorphic to the integers modulo q and thus we may write $y = a^x \pmod q$. In practice it appears that the only fields that are used for implementations are $GF(p)$, where p is a large prime, or $GF(2^n)$ for large n. For a large prime p the function defined on the integers modulo p by $x \rightarrow a^x \pmod p$ is a one-way function. The computation of $a^x \pmod p$ requires at most $2 \log_2 x$ multiplications while the best general algorithms known for extracting logarithms modulo p require a precomputation of the order of $\exp((\ln p)^{1/2}(\ln\ln p)^{1/2})$ operations.

This expression is similar to the time complexity function $L(n)$ for the MPQS factoring algorithm. However, just as with factorization, we must point out that, although it represents the best results known for arbitrary primes p, it is possible to do considerably better if the primes have special properties. For instance, Pohlig and Hellman [Pohlig 78] have described an efficient algorithm for computing discrete logarithms mod p when $p-1$ has only small prime factors. Thus if we are to regard $x \rightarrow a^x \pmod p$ as a one-way function then the prime must be chosen very carefully.

Arithmetic in $GF(2^n)$ for large n can be performed faster than arithmetic over large primes, so there is some reason to prefer the one-way function $x \rightarrow a^x$ in $GF(2^n)$. However, fast algorithms for evaluating discrete logarithms in $GF(2^n)$ have been developed [Coppersmith 84] so care must be taken to ensure that the exponent n is large

---

*In mid-June 1990, the ninth Fermat number ($F_9 = 2^{2^9} + 1$)—having 155 digits—was factored by using the NFS [Lenstra90,90a,91]. The calculations were done over a four-month period on approximately 700 workstations scattered worldwide with the final computation being done on a supercomputer.

enough. (Coppersmith's algorithm has an expected asymptotic complexity function of $\exp(n^{1/3}\ln(n)^{2/3})$.

3.1.4 Squaring modulo *n*. If $n = pq$ where p and q are two large primes then the function $x \rightarrow x^2 (\text{mod } n)$ is a trapdoor one-way function, where the trapdoor information is knowledge of the prime factors.

For any modulus m, an element y is called a quadratic residue modulo m if there exists an x such that $y = x^2 (\text{mod } m)$. Any value x such that $y = x^2 (\text{mod } m)$ is then called a square root of y modulo m.

If the modulus m is a prime then Adleman, Manders, and Miller [Adleman 77] have described a probabilistic polynomial time algorithm for finding square roots modulo m. Furthermore, if m is composite with known factors then this algorithm can be "combined" with the Chinese remainder theorem to extract square roots modulo m. Thus, extracting square roots modulo m is easy provided the factors of m are known. However, Rabin has shown that any algorithm that extracts square roots modulo m can be used to factor m [Kranakis 86]. Thus if $n = pq$, where p and q are unknown, it is only as feasible to invert the one-way function $x \rightarrow x^2 (\text{mod } n)$ as it is to factor n.

3.1.5 The knapsack function. Let $A = \{a_1, \ldots, a_n\}$ be a set of n distinct integers. If $\mathbf{a} = (a_1, \ldots, a_n)$ and $\mathbf{x} = (x_1, \ldots, x_n)$ is a binary n-tuple we write

$$\mathbf{a} \cdot \mathbf{x} = \sum_{i=1}^{n} a_i x_i$$

Then we may define the function $\mathbf{x} \rightarrow \mathbf{a} \cdot \mathbf{x}$ from the set of binary n-tuples to the set of integers. Clearly this function is easy to compute. However, for general $\mathbf{a}$, it is difficult to determine $\mathbf{x}$ from $\mathbf{a}$ and $\mathbf{a} \cdot \mathbf{x}$. (In fact, this is an NP-complete problem.) Thus, for many $\mathbf{a}$, the function $\mathbf{x} \rightarrow \mathbf{a} \cdot \mathbf{x}$ is one-way. Of course if $\mathbf{a}$ is chosen "badly" then this function is easy to invert and it was this observation that formed the basis of the Merkle–Hellman public key system [Merkle 78].

3.1.6 DES. Let $E_k(m)$ denote DES encipherment of message m under key k. Apart from a few weak keys, the only known way to determine the key k from a pair c, m with $c = E_k(m)$ is by an exhaustive key search. Hence, for any given plaintext block x, the mapping $k \rightarrow E_k(x)$ is a one-way function from the set of binary 56-tuples to the set of binary 64-tuples.

3.2 Implementations of Digital Signature Schemes

There have been many proposals for implementing digital signature schemes. In this section we look at some of them and discuss the various ways in which the appropriate one-way function is used in the design.

3.2.1 Digital signatures using RSA. The RSA public key scheme uses the one-way function $E(m)$ defined by $E(m) = m^e (\text{mod } n)$, where n is the product of two (suitably chosen) large primes p and q and $(e, (p-1)(q-1)) = 1$. Its security relies on the infeasibility of determining the deciphering exponent d from the public information e and n and so, since it is only knowledge of d that distinguishes the genuine receiver from everyone else, it seems reasonable to regard d as identifying the genuine receiver of the public key (e, n).

For RSA, the enciphering and deciphering transformations act on the same domain (integers modulo n) and are inverses of each other. Thus if $s = m^d \pmod n$ then $s^e = m^{de} = m \pmod n$. Furthermore, given $m = s^e \pmod n$, n and e then it is difficult to compute s (unless d is known). Thus as user X holds the secret key d, he may sign a given message block m by forming the signature $s = m^d \pmod n$. Anyone who knows the public key e can compute $s^e \pmod n$ and thus decide whether $V_X(m, s) =$ True or False.

By using this technique user X may sign any message from the set of messages $\mathcal{M}$ of integers between 0 and $n - 1$. However, if we are to have a secure signature scheme then we must restrict our message set. In particular, we need $\mathcal{M}$ to have the following two properties:

1. There must be only a small probability that an integer s chosen at random from 0 to $n - 1$ satisfies $s^e \pmod n \in \mathcal{M}$.

2. If $m_1, m_2 \in \mathcal{M}$ then $m_1 m_2 \notin \mathcal{M}$.

The reason for property (1) has already been discussed in Section 2. The reason for property (2) is that if X signs two messages m_1 and m_2, with signatures s_1 and s_2, respectively, then anyone can forge his signature on the message $m_1 m_2$ by computing $s_1 s_2$. (*Note:* $(s_1 s_2)^e = s_1{}^e s_2{}^e = m_1 m_2 \pmod n$.)

The usual solution to properties (1) and (2) is to require that messages be hashed before they are signed. In this case the hash function must satisfy the following property:

3. It is infeasible to find $m_1, m_2 \in \mathcal{M}$ such that $h(m_1 m_2) = h(m_1)h(m_2)$.

If property (3) is not satisfied then $(m_1 m_2, s_1 s_2)$ can be constructed from (m_1, s_1) and (m_2, s_2).

3.2.2 El Gamal's signature scheme. El Gamal [El Gamal 85] has proposed a public key cryptosystem and a digital signature scheme based on discrete logarithms. These schemes have the advantage that the public enciphering key and the public information used in the verifying transformation are the same. However, the two transformations are distinct. Furthermore, it appears that the public key cryptosystem cannot be adapted (as RSA was) to provide digital signatures and the digital signature scheme cannot be adapted to provide encryption. El Gamal's digital signature scheme makes use of the difficulty of computing discrete logarithms over $GF(p)$ where p is a large prime.

In this scheme, a large prime number p is chosen such that $p - 1$ has a large prime factor. The secret information that user X has as input to the signing transformation is an integer x and the public information is a primitive element a of $GF(p)$ together with the integer y such that $y = a^x \pmod p$. Thus, recovering the secret information from the public information requires the computation of a discrete logarithm modulo p. A message block m is now an integer between 0 and $p - 1$ and to sign m user X does the following:

1. Randomly chooses an integer k, $1 < k < p - 1$, such that $(k, p - 1) = 1$ and k has not been used to sign a previous message
2. Computes $r = a^k \pmod p$
3. Finds s such that $m = xr + ks \pmod{(p - 1)}$.

The calculation in (3) is straightforward because, since $(k, p-1) = 1$, Euclid's algorithm can be used to find k^{-1} such that $kk^{-1} = 1 (\text{mod } (p-1))$ and then $s = k^{-1}(m - xr) (\text{mod } (p-1))$.

Once these calculations have been performed, X's signature on m is the pair (r, s).

Since $m = xr + ks \ (\text{mod } (p-1))$, $a^m = a^{xr+ks} = a^{xr}a^{ks} = y^r r^s \ (\text{mod } p)$. Thus the recipient can verify that $(m, r, s) = S_X(m)$ by checking that $a^m = y^r r^s \ (\text{mod } p)$.

Clearly, for any fixed $r = a^k \ (\text{mod } p)$, finding s such that $a^m = y^r r^s \ (\text{mod } p)$ is equivalent to computing a discrete logarithm modulo p. However, it is not clear that the more general problem of simultaneously finding a pair r, s with $a^m = y^r r^s \ (\text{mod } p)$ is also equivalent to finding discrete logarithms modulo p. Nevertheless, it is believed to be difficult and, as yet, no one has found an efficient algorithm to solve this more general problem.

In this scheme user X should only be prepared to sign all messages in the message set $\mathcal{M}$ provided that it is computationally infeasible for anyone else to find $m \in \mathcal{M}$ and integers r, s satisfying $a^m = y^r r^s \ (\text{mod } p)$. As for the systems based on RSA, this places restrictions on the choice of $\mathcal{M}$. For instance, if b and c are arbitrary integers with $(c, p-1) = 1$ then we can construct a triple (m, r, s) satisfying $a^m = y^r r^s \ (\text{mod } p)$ as follows: $r = a^b y^c \ (\text{mod } p)$, $s = -r/c \ (\text{mod } (p-1))$ and $m = rb/c \ (\text{mod } (p-1))$. Thus a necessary condition is that

1. If b, c are integers with $(c, p-1) = 1$ and if $r = a^b y^c \ (\text{mod } p)$ then it should be unlikely that $-rb/c \ (\text{mod } (p-1))$ is an element in $\mathcal{M}$.

Along similar lines we have that if (r, s) is a valid signature for $m \in \mathcal{M}$ and if z, b, c are arbitrary integers with $(zr - cs, p-1) = 1$ then the following triple (m', r', s') satisfies $a^{m'} = y^{r'} r'^{s'} (\text{mod } (p-1))$:

$$r' = r^z a^b y^c \ (\text{mod } p), s' = (sr^1/(zr - cs)) (\text{mod } (p-1)), \ m' = \frac{r'(zm - bs)}{zr - cs} \ (\text{mod } (p-1))$$

Thus we also require:

2. The probability that $m' \in \mathcal{M}$ for an arbitrarily chosen r, z, b, c should be negligible.

Again, the usual solution to (1) and (2) is to require that the messages be hashed before they are signed.

The El Gamal scheme can be generalized to use arithmetic in the field $GF(q)$. For practical implementations q is usually either a prime or a power of 2.

3.2.3 The Fiat–Shamir signature scheme. The Fiat–Shamir scheme [Fiat 87] exploits the difficulty of extracting square roots modulo n where n is the product of two large primes p and q. User X holds k integers $s_1, \ldots, s_k$ in the range 0–n such that, for each i, $(s_i, n) = 1$ and they are the secret information that he uses in the signing transformation. The values $v_1 = s_1^{-2} \ (\text{mod } n), \ldots, v_k = s_k^{-2} \ (\text{mod } n)$ are made public and are used in the verifying transformation. For this scheme to be secure, it is crucial that $s_1, \ldots, s_k$ cannot be computed from $v_1, \ldots, v_k$. Thus, since being able to find square roots implies being able to factor n, this means that n must be large enough that it is difficult to factor.

To sign a message M, user X

1. chooses random $r_1, \ldots, r_t$ between 0 and $n-1$ and computes $x_i = r_i^2 (\text{mod } n)$, $i = 1, \ldots t$.
2. computes $f(M, x_1, \ldots, x_t)$ where f is a public hashing function and uses the first kt bits as entries e_{ij} of a t by k binary matrix E.
3. computes $y_i = r_i \prod_{e_{ij}=1} s_j$ (modulo n) for $i = 1, \ldots, t$.

The signature on M by user X is then (y, E) where y is the vector $(y_1, \ldots, y_t)$ and E is the matrix (e_{ij}).

To verify this signature, the recipient

1. computes $z_i = y_i^2 \prod_{e_{ij}=1} v_j \text{ (mod } n)$ for $i = 1, \ldots, t$.
2. checks that the first kt bits of $f(M, z_1, \ldots, z_t)$ are the entries e_{ij} of E.

A signature generated by user X will always be verified since

$$z_i = y_i^2 \prod_{e_{ij}=1} v_j = r_i^2 \prod_{e_{ij}=1} s_j^2 \prod_{e_{ij}=1} v_j = r_i^2 \prod_{e_{ij}=1} s_j^2 v_j = r_i^2 = x_i \text{ (mod } n)$$

A forger would need to be able to find a message M, a vector y, and a matrix E such that the first kt bits of $f(M, z_1, \ldots, z_t)$ are the entries of E where

$$z_i = y_i^2 \prod_{e_{ij}=1} v_j \text{ (mod } n) \text{ for } i = 1, \ldots, t.$$

Fiat and Shamir [Fiat 87] suggest that if a forger were able to do this then he could, with non-negligible probability, find a pair of signatures (y, E) and (y', E') such that

$$y_i^2 \prod_{e_{ij}=1} v_j = y'^2_i \prod_{e'_{ij}=1} v_j.$$

But this would then enable him to determine a square root modulo n of some product $\prod_{j=1}^{k} v_j^{c_j} \text{ (mod } n)$ where for $j = 1, \ldots, k$, $c_j = -1$, 0, or 1, not all zero. This would then allow the forger to factor n. Thus, provided n is difficult to factor and f is a suitable hashing function, it is computationally infeasible to forge signatures in this scheme.

3.2.4 The Goldwasser–Micali–Rivest signature scheme. In [Goldwasser 84] Goldwasser, Micali, and Rivest define claw-free permutations and a signature scheme that relies on the fact that squaring modulo n (where $n = pq$ with p and q secret large primes) is a trapdoor one-way function.

Before we can describe their scheme we must introduce some notation.

For any integer n we let Z_n^* denote the integers between 1 and n which are coprime to n. If a is an integer that is coprime to the prime p, the Legendre symbol $(a/p) = 1$ if a is a quadratic residue modulo p, and $(a/p) = -1$ if a is a quadratic nonresidue modulo p. If $n = pq$ then the Jacobi symbol $(a/n) = (a/p)(a/q)$ for $a \in Z_n^*$. Thus a is a quadratic residue modulo n if and only if $(a/p) = (a/q) = 1$. We note that $(-1/n) = 1$ but -1 is not a quadratic residue and $(2/n) = -1$.

If we put $D_n = \{x \in Z_n^* \mid (x/n) = 1 \text{ and } 0 < x < n/2\}$ then we can define two functions $g_0 : D_n \to D_n$ and $g_1 : D_n \to D_n$:

$$g_0(x) \begin{cases} = x^2 \pmod n \text{ if } x^2 \pmod n < n/2 \\ = -x^2 \pmod n \text{ if } x^2 \pmod n > n/2 \end{cases}$$

$$g_1(x) \begin{cases} = 4x^2 \pmod n \text{ if } 4x^2 \pmod n < n/2 \\ = -4x^2 \pmod n \text{ if } 4x^2 \pmod n > n/2 \end{cases}$$

Both are trapdoor claw-free permutations with knowledge of the primes p and q as the trapdoor. If the factorization of n is known then it is easy to determine whether or not an element of D_n is a quadratic residue or a nonresidue. Furthermore, given this knowledge, the computation of the square root (in D_n) of a quadratic residue is also straightforward. Finally, before describing their scheme we note that if x and y in D_n are such that $g_0(x) = g_1(y)$, that is, $x^2 = 4y^2 \pmod n$ then $(x - 2y)(x + 2y) = 0 \pmod n$ but $x - 2y \not\equiv 0 \pmod n$. Thus the greatest common divisor of $x - 2y$ and n is a nontrivial factor of n. This, of course, means that finding a pair with $g_0(x) = g_1(y)$ would result in being able to factor n.

The signature scheme of Goldwasser, Micali, and Rivest has a set of messages $\mathcal{M}$ consisting of (finite) sequences with terms from the set $\{0, 1\}$. The secret information used in signing a message is the factorization of n which is such that $p = 3 \pmod 4$ and $q = 7 \pmod 8$. It is this information that allows g_0 and g_1 to be inverted. The public information used in verifying a signature consists of the functions g_0 and g_1 and a verification parameter r from D_n.

To sign a message $m = m_1 m_2, \ldots, m_k$ where $m_i \in \{0, 1\}$ for $i = 1, \ldots, k$ the signature $S = S_X(m) = g_{<m>}^{-1}(r) = g_1^{-1}(g_0^{-1}\ (g_{m_k}^{-2}(\ldots(g_{m_1}^{-2}(r))\ldots)))$ is computed. To verify a signature S on message m it is checked that $g_{<m>}(S) = g_{m_1}^2(\ldots(g_{m_k}^2(g_0(g_1(S))))\ldots) = r$. Thus $V_X(m, S) =$ True if and only if $g_{<m>}(S) = r$.

Provided that some means of authenticating them can be established, different values for the verification parameter r may be used for different messages. This is certainly feasible and, as an illustration, we now show how a tree structure is used to generate authenticated verification parameters.

Suppose that f_0 and f_1 are a pair of trapdoor claw-free permutations and let $r_0 \in D_n$ be a public, authenticated verification parameter. Then we can use r_0 to authenticate two (randomly chosen) values r_{00}, r_{01}. Let $<r_{00}>$, $<r_{01}>$ be encodings of r_{00}, r_{01} as sequences of terms from $\{0, 1\}$. Put $T_0 = f^{-1}_{<r00><r01>}(r_0)$. Each of r_{00}, r_{01} can now be used to authenticate an authentication parameter r (chosen at random from D_n) which itself can be used to sign a message m. Put $T_r = f^{-1}_{<r>}(r_{00})$ and $S = g^{-1}_{<m>}(r)$. Then the signature on m becomes $(S, T_r, r, T_0, r_{00}, r_{01})$ which is verified by checking that $f_{<r00><r01>}(T_0) = r_0$ (the published verification parameter), $f_{<r>}(T_r) = r_{00}$ and $g_{<m>}(S) = r$.

Clearly r_{00} and r_{01} could then be used to authenticate four values r_{000}, r_{001}, r_{010}, and r_{011} in a similar way. By generating a tree of values "r" in this way any number of verification parameters r may be authenticated and used to sign messages.

3.2.5 Digital signatures using Rabin's public key cryptosystem. In [Rabin 79] Rabin proposed a public key cryptosystem based on the one-way function $x \to x^2 \pmod n$ where $n = pq$ as the product of two large primes. This cryptosystem is provably equivalent to factoring n.

Each user X has a public key p_X consisting of $n(= pq)$ and an integer $B < n$ chosen by user X. The secret key s_X is knowledge of p and q.

The message set $\mathcal{M}$ is a subset of the integers between 0 and $n - 1$. A message $m \in \mathcal{M}$ is encrypted as the cryptogram $C = E_{p_x}(M)) = M(M + B) \pmod n$. (Note that if w is any square root of 1 modulo n then $E_p(M') = C$ where $M' = -\frac{1}{2}B + w(M + \frac{1}{2}B)$. Thus, for C to be uniquely decipherable it is necessary that whenever $M \in \mathcal{M}$, $M' - \frac{1}{2}B + w(M + \frac{1}{2}B) \notin \mathcal{M}$ for w satisfying $w^2 = 1 \pmod n$, $w \neq 1$.

Given a cryptogram C, it may be deciphered by finding a solution $x = M$ to the equation $x^2 + Bx = C \pmod n$ which is equivalent to finding a solution $y = M + \frac{1}{2}B$ to the equation $y^2 = C + \frac{1}{4}B^2 \pmod n$. This latter equation has a solution only if $C + \frac{1}{4}B^2$ is a quadratic residue modulo n. Furthermore, when this condition is satisfied, there are four solutions for y. However only one satisfies $M = y - \frac{1}{2}B \in \mathcal{M}$. With the knowledge of the factors p and q of n (i.e., the secret key s_X) it is possible to extract the square-root of $C + \frac{1}{4}B^2 \pmod n$ and thus decipher C, that is, to compute $M = D_{s_x}(C)$. Moreover, since being able to extract square root modulo n implies being able to factor n, this public key cryptosystem is secure provided n is difficult to factor.

Rabin's public key cryptosystem is not immediately applicable as a digital signature scheme since it is only possible to decipher a value C if $C + \frac{1}{4}B^2$ is a quadratic residue modulo n. So not all messages could be signed using D_{s_x}. However, it is possible to combine this cryptosystem with a suitably chosen hashing function h to provide digital signatures. The hashing function is used to map a subset of the set of messages $\mathcal{M}$ onto the set of quadratic residues modulo n. The guarantee that the hashed version is a quadratic residue is achieved in the following way.

A random string U (of fixed size) is concatenated with the message $M \in \mathcal{M}$ to be signed and $h(M, U)$ is computed. User X, in possession of his secret key s_X (i.e., the factors p and q of n) can determine whether or not $h(M, U)$ is a quadratic residue modulo n. If it is not, then another random string is chosen. The process is repeated until the value $h(M, U)$ is a quadratic residue modulo n. The signature $S_X(M)$ on message M is then the pair (S, U) where $S = D_{s_x}(h(M, U)$. A signature (S, U) is verified by checking that $h(M, U) = E_{p_x}(S) = S(S + B) \pmod n$.

3.4 Merkle's Tree Signature Scheme

In [Merkle90] Merkle proposed a digital signature scheme, based on a symmetric encryption function that provides an infinite tree of one-time signatures.

The basic building block of Merkle's scheme is a one-way function f based on a symmetric encryption function such as the DES. Thus, if the encryption transformation under key k is denoted E_k and X is a fixed (public) plaintext value then $f(k) = E_k(X)$. Merkle proposed the use of f to provide verification parameters in an adaptation of the Diffie–Lamport one-time signature scheme. Let h be a (public) hashing function which maps a message M to a string $h(M)$ of n bits. The hashing function h may be constructed from the one-way function f. If $c(M)$ is a count of the number of zero bits in $h(M)$, then $(h(M), c(M))$ is a string $(a_1, \ldots, a_t)$ of $t = n + \log n$ bits. The signer generates $x_1, \ldots, x_t$ at random and puts $y_i = f(x_i)$ for $i = 1, \ldots, t$. Suppose that $y_1, \ldots, y_t$ can be authenticated. Then the message M can be signed by revealing $y_1, \ldots, y_t$ and those x_i for which $a_i = 1$. A signature $S = (y_1, \ldots, y_t, x_{j_1}, \ldots, x_{j_l})$ is verified by checking that $y_1, \ldots, y_t$ are authentic and that $a_i = 1$ for $i = j_1, \ldots, j_l$ and $a_i = 0$ for $i \neq j_1, \ldots, j_l$. Only the signer, who knows $x_1, \ldots, x_t$

can generate this signature. Moreover, no other message M' can be signed using the knowledge of the revealed $x_{j_1}, \ldots, x_{j_t}$ since either $h(M')$ will have a "1" where $h(M)$ has a "0" (requiring a "new x_j" to be revealed) or $c(M')$ will be greater than $c(M)$ and so $c(M')$ will have a "1" where $c(M)$ has a "0".

The verification parameters $y_1, \ldots, y_t$ could be authenticated by publishing them in a registry. However, since the Diffie-Lamport signature is a one-time signature, to produce a large number of signatures would then require the publication of a large amount of data.

To avoid this complication, verification parameters are generated in a tree structure. Each node of this tree consists of the verification parameters that are used to sign a message and authenticate those verification parameters of subsequent nodes.

The signer generates $u_1, \ldots, u_t$, $x_1, \ldots, x_t$ and $w_1, \ldots, w_t$ at node 1 and puts $v_i = f(u_i)$, $y_i = f(x_i)$, and $z_i = f(w_i)$ for $i = 1, \ldots, t$. The value $R = h(h(v_1, \ldots, v_t), h(y_1, \ldots, y_t), h(z_1, \ldots, z_t))$ is published in a registry and is used to authenticate the verification parameters $v_1, \ldots, v_t$, $y_1, \ldots, y_t$, and $z_1, \ldots, z_t$.

The values $x_1, \ldots, x_t$ and $y_1, \ldots, y_t$ are used to sign the first message as above. (Note that the values $V = h(v_1, \ldots, v_t)$ and $Z = h(z_1, \ldots, z_t)$ must be included in the signature so that the values $y_1, \ldots, y_t$ may be authenticated by checking that $h(V, h(y_1, \ldots, y_t), W) = R$.) Values $u_1', \ldots, u_t'$, $x_1', \ldots, x_t'$, and $w_1', \ldots, w_t'$ and values $u_1'', \ldots, u_t''$, $x_1'', \ldots, x_t''$, and $w_1'', \ldots, w_t''$ are generated for the next two nodes, nodes 2 and 3, and the corresponding images under f computed. The value $R' = h(h(v_1', \ldots, v_t'), h(y_1', \ldots, y_t'), h(z_1', \ldots, z_t'))$ is signed using $u_1, \ldots, u_t$ and $v_1, \ldots, v_t$. Again the values $Y = h(y_1, \ldots, y_t)$ and $Z = h(z_1, \ldots, z_t)$ are included in this signature so that $v_1, \ldots, v_t$ can be authenticated by checking that $h(h(v_1, \ldots, v_t), Y, Z) = R$.

The signature on the second message M' consists of this signature on R' and a signature on M' computed as above using $x_1', \ldots, x_t'$, $y_1', \ldots y_t'$, $V' = h(v_1', \ldots, v_t')$, $Z' = h(z_1', \ldots, z_t')$ and R'.

Similarly the value $R'' = h(h(v_1'', \ldots, v_t''), h(y_1'', \ldots, y_t''), h(z_1'', \ldots, z_t''))$ is signed using $w_1, \ldots, w_t$ and $z_1, \ldots, z_t$ and a third message M'' may be signed.

Now the nodes 2 and 3 can be used to generate authenticated verification parameters at two more nodes each. In this way any number of messages may be signed. The signature on the message at node 2^i has length $i + 1$ times that of the signature on the first message. Only the value R is stored in the registry, all other verification parameters are authenticated by including signatures on them in the signature on the message. Thus the problem of one-time signatures requiring large amounts of verification data is overcome by allowing the size of the signature to grow as the number of messages signed increases.

4 TECHNIQUES FOR HASHING

Many proposals have been made for one-way hash functions, and we do not attempt to list all such proposals here; instead we describe some of the more significant. Unfortunately, a high proportion of the existing proposals have turned out to be flawed in some way, perhaps indicating that designing easily implemented and secure hash functions is not as easy as it appears.

It is interesting to note that most of the published proposals for hash functions fall into one of two categories: those based on use of a block cipher, and those based on modular arithmetic. We use this classification to survey some of the more important proposed schemes and, where relevant, their weaknesses.

4.1 Block Cipher–Based Hash Functions

In this section we consider hash functions built using a block cipher. By *block cipher* we mean here a symmetric key cryptosystem that transforms message (plaintext) blocks of fixed length (say, n bits) into ciphertext blocks of the same length, under the control of a key k (of, say, m bits). Thus, if M is an n-bit message block then

$$C = E_k(M)$$

is the corresponding n-bit ciphertext block when key k is used. In addition, we have

$$M = D_k(C) = D_k(E_k(M))$$

A widely used example is provided by the U.S. Standard DES algorithm, [ANSI3.92], which has $n = 64$ and $m = 56$.

To be secure against known-plaintext attacks, block ciphers require the following property. Given a pair (M, C) with the property that $C = E_k(M)$, or a set of such pairs, no means of discovering k exists that is significantly more efficient than searching through all 2^m possibilities for k. It is also normally assumed that the block cipher will resist chosen ciphertext/plaintext attacks, that is, where the value(s) of M or C can be chosen by the cryptanalyst. We assume here that the block cipher used to construct a hash function has all such desirable properties.

4.1.1 Cipher block chaining-message authentication code. Probably the most obvious way of using a block cipher to construct a message digest is based on the standard mode of use for a block cipher called cipher block chaining (CBC) [ANSI3.106], [ISO8372]. Note that this gives an example of a keyed hash function. In this mode of use the message is divided into a sequence of n-bit blocks,

$$M_1, M_2, \ldots, M_t$$

say. The sequence (C_i) is then derived, where

$$C_i = E_k(C_{i-1} \oplus M_i)$$

where $\oplus$ denotes bit-wise ex-or and C_0 is the all-zero block. The derived message digest is then simply C_t (or part of it), and we call C_t the CBC-MAC of the message. The CBC-MAC method of computing message digests is standardized for banking authentication applications, [ANSI9.9], [ANSI9.19], [ISO8731–1], and is also a draft standard for general authentication purposes, [DIS9797]. Note that, in these simple authentication and integrity applications, the CBC-MAC provides the desired security without subsequent signature; the key used to produce the digest is sufficient to guarantee the integrity and authenticity of the message.

Although satisfactory for simple authentication applications, the CBC-MAC is not, however, a secure means of producing a digital signature, as we now show. First observe that the recipient of a signed message must be equipped with the key k used to

produce the digest. Given knowledge of k, the above hashing function does not satisfy $H2$, that is, it is not one-way. In fact, given any n-bit block, C say, and any sequence of n-bit blocks

$$N_1, N_2, \ldots, N_w$$

it is possible to choose a further n-block N_{w+1} such that the sequence

$$N_1, N_2, \ldots, N_{w+1}$$

has CBC-MAC C. This can be done by setting N_{w+1} to $D_k(C) \oplus C_w$ where C_w is the CBC-MAC derived from $N_1, N_2, \ldots, N_w$.

Another problem with the CBC-MAC is that, when used with an n-bit block cipher it gives a digest of at most n bits. The most well-known block cipher is DES [ANSI3.92] which has $n = 64$, which therefore gives digests of 64 bits—too small for use for computing digital signatures.

4.1.2 Bidirectional message authentication code. Many attempts have been made to overcome the above problems by using a block cipher in different ways, and we consider some of them below. We start by considering a closely related scheme, called the bidirectional message authentication code (BMAC), originally described in Internet RFC 1040 [RFC1040]. The BMAC is again based on use of CBC; this has significant practical advantages since existing hardware implementations of DES can perform CBC very fast. As we shall see, the BMAC is again unsuitable for use in producing digital signatures—the attack that can be made against it is of significance since very similar attacks can be made against many other proposed hash functions.

The BMAC produces a message digest of $2n$ bits (given that the block cipher used to produce it produces ciphertext blocks of length n bits). Hence, when based on DES, the BMAC produces 128-bit digests, thereby avoiding one of the problems of the CBC-MAC. Suppose, as before, that the BMAC is to be computed for the sequence of n-bit blocks

$$M_1, M_2, \ldots, M_t$$

Let C_t be the CBC-MAC derived from this sequence and let B_1 be the CBC-MAC derived from the sequence

$$M_t, M_{t-1}, \ldots, M_1$$

The BMAC is then simply the concatenation of C_t and B_1.

Unfortunately, given knowledge of the key k used to compute it, and given n is of the order of 64, the BMAC is not one-way, that is, given any pair of n-bit blocks a message can be found having BMAC equal to this pair of blocks. Indeed, the method we now describe (previously given in [Mitchell90]) allows the message to be chosen arbitrarily, except that it must contain two 64-bit blocks which are essentially random in nature.

Suppose that the supplied pair of blocks is (C, B) and the given key is k. Suppose also that the message to be "matched" to the BMAC (C, B) is M, where M can be divided into two parts M_1 and M_2. In addition, choose at random an n-bit block X.

First prepare $2^{n/2}$ variants of M_1, each variant consisting of a whole number of n-bit blocks (although the number of blocks in each variant may vary). Davies and

Price ([Davies84], p. 278) illustrate a simple technique by which this may be done so that each variant is valid English and each variant is semantically the same. Basically, if q positions are identified within the message at which two possible wordings have the same meaning, then 2^q different variants of the message may be derived.

For each variant do the following. Suppose that

$$N_2, N_3, \ldots, N_s$$

is the decomposition of the variant into n-bit blocks (note that the first block of the variant is not defined yet). We now choose N_1 so that the "reverse" CBC-MAC of

$$N_1, N_2, \ldots, N_s, X$$

is B. We also let Y be the CBC-MAC of $N_1, N_2, \ldots, N_s$. More formally, for every i $(i = s, s - 1, \ldots, 2)$ let

$$F_i = E_k(N_i \oplus F_{i+1})$$

where $F_{s+1} = X$. Then let

$$N_1 = D_k(B) \oplus F_2$$

N_1 then constitutes the first block of this variant of M_1. For every i $(i = 1, 2, \ldots, s)$ let

$$G_i = E_k(N_i \oplus G_{i+1})$$

where G_0 is the all-zero block. Finally let $Y = G_s$. Each variant $(N_1, N_2, \ldots, N_s)$ is then stored along with its corresponding value of Y.

Second, prepare $2^{n/2}$ variants of M_2, each variant consisting of a whole number of n-bit blocks (again, the number of blocks in each variant may vary). For each variant,

$$P_{s+1}, P_{s+2}, \ldots, P_{r-1}$$

say, we now choose P_r so that the "reverse" CBC-MAC of

$$P_{s+1}, P_{s+2}, \ldots, P_r$$

is X. We also let Z be the value required to ensure that the sequence of blocks Z, $P_{s+1}, P_{s+2}, \ldots, P_r$ has CBC-MAC equal to C. More formally, for every i $(i = s + 2, s + 3, \ldots, r)$ let

$$H_i = D_k(H_{i-1}) \oplus P_{i-1}$$

where $H_{s+1} = X$. Then let

$$P_r = D_k(H_r)$$

P_r then constitutes the last block of this variant of M_2. For every i $(i = r - 1, r - 2, \ldots, s)$ let

$$L_i = D_k(L_{i+1}) \oplus P_{i+1}$$

where $L_r = C$. Finally let $Z = L_s$. This value of Z is then compared with all the stored values of Y resulting from the $2^{n/2}$ variants of M_1. There is a good chance that, before

all $2^{n/2}$ variants of M_2 have been processed, a match, that is, a pair of values Y, Z with $Y = Z$, will be found (we justify this claim below).

Now suppose that the sequences (N_i) $(1 \leq i \leq s)$ and (P_j) $(s + 1 \leq j \leq r)$ give such a match. Then the sequence of n-bit blocks

$$N_1, N_2, \ldots, N_s, P_{s+1}, P_{s+2}, \ldots, P_r$$

will have BMAC (C, B), as desired.

The above attack does require a nontrivial amount of processing time and storage, although neither of these two requirements make the attack infeasible unless n is chosen to be substantially larger than 64. The attack requires the processing of some $2^{n/2+1}$ part-messages, and a number of block cipher encryptions/decryptions are required for each such part-message. However, most block ciphers are designed to run fast and if DES is used as an example (which has $n = 64$), then the amount of computation required to perform the above attack is not excessive, even without massive computing resources.

Finally, note that we asserted that the probability of a match being found was good, without giving any justification for such a claim. In general, if samples of size u and v are drawn independently, at random, and with replacement from a population of size N, then the probability that there will be a match between the two samples is approximately

$$1 - e^{uv/N}$$

given that u and v are small compared with N. Thus given n is reasonably large, the probability of a match in the above argument is approximately $1 - 1/e$ which is around 0.63.

In conclusion, it should be clear that the BMAC is not one-way when used with a 64-bit block cipher such as DES. Moreover, given a 128-bit block cipher (which would defeat the above attack) then other options appear more attractive, such as the Davies–Meyer (DM) scheme that we now describe.

Before proceeding note that the above attack is just one example of a well-known class of attacks against hash functions, namely the so-called "birthday attacks." All these attacks rely on arguments relating to the probability of a match being found within a set of samples from a large set.

4.1.3 The DM scheme. We now describe a different and apparently more secure way of using a block cipher to construct a hash function. This scheme apparently dates back to the early 1980s and is variously attributed to Davies [Winternitz84a] and Meyer [Davies85]; we therefore follow Quisquater and Girault [Quisquater89] and refer to it as the DM scheme. This scheme is also the subject of a draft proposal for an international standard [DP10118].

As before, the message to be signed is first divided into a series of fixed length blocks; this time, however, the block length is m (the key length for the block cipher) rather than n. Suppose the message to be signed is

$$M_1, M_2, \ldots, M_t$$

where each M_i contains m bits. Then let H_i $(1 \leq i \leq t)$ be defined by

$$H_i = E_{M_i}(H_{i-1}) \oplus H_{i-1}$$

where $H_0 = I$, an "initializing value" which may be prearranged (e.g., to 0, the m-bit block of all zeros) or randomly chosen and therefore forms part of the digest, and, as before, $\oplus$ represents bitwise exclusive-or. The message digest is then simply H_t, an n-bit block. At this point note that, if a random initializing value is used then, to prevent certain types of attack, it must be signed along with the digest; this warning applies to all the schemes described below.

This hash function is of particular interest since, if it is assumed that no special properties of the underlying block cipher are used, then it can be proved to be one-way [Winternitz84b]. However, it is still not collision-free unless n (the plaintext/ciphertext block size for the cipher) is sufficiently large, say 128 bits. Thus, for digital signature applications, the DM scheme is not appropriate for use with 64-bit block ciphers such as DES.

Because of the widespread use of 64-bit block ciphers (in particular, DES), efforts have recently been made to modify the DM scheme so as to produce $2n$-bit message digests when using an n-bit block cipher. The following scheme appears in the latest draft standard DP10118 [DP10118].

The message to be signed is again divided into a series of m-bit blocks; this time, however, the number of blocks must be even (as with previous schemes, appropriate padding rules need to be devised for messages that do not divide conveniently). Note also that m and n must satisfy $m \leq n$. Label the sequence of blocks

$$M_1, M_2, \ldots, M_{2t}$$

Then let H_i and G_i $(1 \leq i \leq 2t)$ be defined by

$$\begin{aligned} G_{2j-1} &= E_{M_{2j-1}}(H_{2j-3} \oplus e(M_{2j})) \oplus e(M_{2j}) \oplus H_{2j-2} \ (1 \leq j \leq t) \\ G_{2j} &= E_{M_{2j}}(G_{2j-1} \oplus e(M_{2j-1})) \oplus e(M_{2j-1}) \oplus H_{2j-3} \ (1 \leq j \leq t) \\ H_{2j-1} &= G_{2j} \oplus H_{2j-2} \ (1 \leq j \leq t) \\ H_{2j} &= G_{2j-1} \oplus H_{2j-3} \ (1 \leq j \leq t) \end{aligned}$$

where H_{-1} and H_0 are initializing values (possibly m-bit blocks of all zeros), $e(\ )$ is an "expanding function" mapping m-bit values onto n-bit values (e.g., by padding with zeros), and, as before, $\oplus$ represents bitwise exclusive-or. The message digest is then simply H_{2t-1} concatenated with H_{2t}, a $2n$-bit value.

A similar proposal has been published by Quisquater and Girault [Quisquater89]. Both of these schemes should be treated with great caution; the logic behind their construction is by no means obvious, and they need thorough review before practical use.

4.1.4 Some insecure schemes. We now consider a variety of other schemes based on block ciphers which have been first proposed and then found wanting. Their existence attests to the difficulty of finding secure hash functions that are usable in practice.

The DM scheme described above is a modification of a scheme described by Rabin [Rabin78], Matyas [Matyas79], and Davies and Price [Davies80]; we refer to this earlier scheme as the Rabin–Matyas–Davies–Price (RMDP) scheme. In the RMDP system the message to be signed is first divided into a series of m-bit blocks:

$$M_1, M_2, \ldots, M_t$$

Then let H_i $(1 \leq i \leq t)$ be defined by

$$H_i = E_{M_i}(H_{i-1})$$

where H_0 is an initializing value. The message digest is then simply H_t, an n-bit block. Merkle showed the RMDP function is not one-way when n is of the order of 64 (see, e.g., [Davies80] or Winternitz84a]) using a "birthday attack" argument very similar to that described in Section 4.1.2, above. We now briefly outline Merkle's attack. Before proceeding note that the basic idea behind this attack was given by Yuval in 1979 [Yuval79]; interestingly, Yuval also pointed out the need for a hash function to satisfy the collision-freedom property.

Consider any digest H with initializing value I. Choose any message and divide it into two parts. Devise $2^{n/2}$ variations of the first part and $2^{n/2}$ variations of the second part. Starting with I compute the digest for the $2^{n/2}$ variations of the first part of the message. Using the invertibility of E, start from H and work back to $2^{n/2}$ values using the variants of the second part of the message. As in Section 4.1.2 there will be a better than 50% probability of finding a first and second part that "match" and a message will have been found to fit the given digest. Thus the RMDP scheme is not one-way unless n is chosen to be substantially larger than 64.

It is straightforward to see that the DM scheme is a simple derivative of the RMDP system which avoids the obvious "birthday attack." Other methods of modifying the RMDP function have been proposed, and we now briefly consider two of them.

Davies [Davies83] describes a scheme attributed to Bitzer, where the message digest

$$M_1, M_2, \ldots, M_t$$

(M_i contains m bits) is derived by computing

$$H_i = E_{M_i'}(H_{i-1})$$

where $M_i' = M_i + s(H_{i-1})$, $s(\,)$ is a "selection function" reducing an n-bit block to an m-bit block, and H_0 is an initializing value. The message digest is then H_t, an n-bit block. As Winternitz has pointed out [Winternitz84a], this technique is vulnerable to a very similar attack to that described above for the RMDP scheme.

Davies and Price [Davies80] (quoted by Denning [Denning83]) suggest using the RMDP scheme twice. In 1983, Winternitz ([Winternitz84a], [Akl84]) showed how this method could be attacked if the DES block cipher is used, based on special properties of the DES algorithm. Subsequently, in 1985 Coppersmith [Coppersmith86] gave a general birthday attack applicable regardless of the block cipher employed, this time a little more complex but perfectly feasible for 64-bit block ciphers. Coppersmith also considered the case where the RMDP scheme is iterated three times and again exhibits a feasible birthday attack. Coppersmith's method was then generalized by Girault, Cohen, and Campana [Girault89], who described how to attack a p-times-iterated version of RMDP, which when $p = 4$ and $n = 64$ is just about feasible.

A number of other possibilities have been described; we consider one of them. This is a variant of CBC described by Meyer and Matyas [Meyer82]. Let the message be

$$M_1, M_2, \ldots, M_t$$

(M_i contains n bits). The digest, H, is derived by first computing

$$W = M_1 \oplus M_2 \oplus \ldots \oplus M_t$$

and then computing H as the CBC-MAC of the sequence

$$M_1, M_2, \ldots, M_t, W$$

Unfortunately, as described by Akl [Akl84] this method is yet again susceptible to a birthday attack when n is of the order of 64. Moreover this scheme has a number of other serious weaknesses (see, e.g., [Jueneman83b], [Jueneman83], [Meijer83], [Akl84]). Although Jueneman et al. [Jueneman83] and Akl [Akl84] have described a number of improvements to this scheme, including using addition modulo 2^n instead of bit-wise exclusive-or and the possibility of multiple iterations, it still does not appear an attractive option, since undesirable properties remain.

4.1.5 Merkle's methods. Merkle has described a number of methods for deriving one-way hash functions from block ciphers [Merkle89a]. He starts by describing a "metamethod" (also described in his Ph.D. thesis [Merkle79], [Merkle82]) which builds a hash function h producing k-bit digests using a function h_0 mapping from s-bit values to k-bit values for some fixed $s > k$. It operates as follows. Divide the message to be hashed into a sequence of blocks, each of $s - k$ bits, say,

$$M_1, M_2, \ldots, M_t$$

Then let

$$H_i = h_0(H_{i-1}; M_t)$$

where the semicolon denotes concatenation of data and where H_0 is an initializing value. The message digest is then H_t. This metamethod forms a convenient general description for most of the methods described above; for example, in the DM scheme of Section 4.1.3,

$$h_0(H_{i-1}; M_i) = E_{M_i}(H_{i-1}) \oplus H_{i-1}$$

where $s = m + n$. Note also that this metamethod is the basis of recent theoretical work on hash functions by Damgard [Damgard89].

In [Merkle89a], a modified form of the DM scheme is described which, in the above notation and for some chosen d, uses an h_0 having $s = m + n - 1$ and $k = m + n - d$, that is, it processes a message $d - 1$ bits at a time. It operates as follows.

Suppose, as before, that

$$M_1, M_2, \ldots, M_t$$

is a decomposition of a message into $(d - 1)$-bit blocks. Let h_0^* be the DM meta-function, that is,

$$h_0^*(A; B) = E_B(A) \oplus A$$

where A is an n-bit block, B is an m-bit block, and h_0^* maps $(m + n)$-bit blocks into n-bit blocks. Then define h_0 by

$$h_0(H_{i-1}; M_i) = Tr_d(h_0^*(H_{i-1}; 0; M_i); h_0^*(H_{i-1}; 1; M_i))$$

where Tr_d denotes truncation by omitting d bits of the block and 0 and 1 denote single bits fixed to 0 and 1. The hash function obtained from this metamethod will clearly produce an $(m + n - d)$-bit result.

As Merkle points out, d needs to be chosen so that the value $m + n - d$ is still sufficiently large to rule out collision attacks. For DES (where $m = 56$ and $n = 64$), Merkle suggests using $d = 8$, that is, the message digests will contain 112 bits, while the message is processed in 7-bit blocks.

Unfortunately this method is rather inefficient (two DES encryptions are required to process 7 bits of message), and Merkle [Merkle89a] goes on to describe two other, rather more sophisticated, means for deriving hash functions from block ciphers which are more efficient. These other methods also have the virtue of retaining the ability to transform block ciphers with relatively small m and n (e.g., DES) into apparently collision-free hash functions.

Before proceeding we briefly describe another rather interesting hashing scheme based on block ciphers (see Merkle [Merkle79], [Merkle82] and also Akl [Akl84]). In this scheme the metamethod h_0 is given by

$$h_0(H_{i-1}; M_i) = E_{H_{i-1}; M_i}(I)$$

where I is initializing value. For this method to work, m (the key size) must be larger than n (the plaintext/ciphertext block size), and the message is then divided into $(m - n)$-bit blocks M_i. For this method to be collision-free it is also necessary that n be sufficiently large, that is, at least 100 bits. In the absence of block ciphers satisfying these properties the method is of rather academic interest, but it is nevertheless of importance in suggesting other ways in which practical and secure hash functions may be constructed.

4.1.6 Conclusions. The use of existing block ciphers to construct one-way hash functions is often appealing. However, as should be clear from the above discussion, such an approach requires great care. However, the following recommendations can be made.

If a 128-bit block cipher (i.e., one with $n = 128$) is available then the DM scheme appears secure given that the block cipher has no regularities that a cryptanalyst could exploit (the proof of the one-way property for the DM scheme assumes a "perfect" block cipher). Unfortunately, although designing a practically secure 128-bit block cipher is probably not difficult (e.g., by modifying DES appropriately), no such algorithm exists either as a standard or even a de facto standard. The only widely used and trusted block cipher algorithm is the DES, which has $n = 64$. Therefore, in practice the real problem is to derive a secure hash function from a 64-bit cipher.

If a 64-bit block cipher is to be used, then, as can be seen from the many failures described above, the problem is much more difficult. The only candidate systems currently available are the scheme given in ISO DP10118 [DP10118] (see Section 4.1.3) and the schemes of Merkle (see Section 4.1.5). None of these schemes have been subjected to prolonged public scrutiny, and use of them at this stage would be rather risky. Other possibilities include multiple iterations of existing hash functions, although this approach can also be flawed.

In the context of existing knowledge, Winternitz's comments from 1984 [Winternitz84a] remain extremely appropriate:

System implementors would be well-advised to use as much overkill as they can afford. They should go through the message several times and use a hash value as long as possible. Given our ignorance, safety requires a system several times as complicated as the simplest system yet unbroken.

4.2 Hash Functions Based on Modular Arithmetic

Another widely discussed method for constructing hash functions is the use of modular exponentiation. This is especially attractive if the signature function itself is based on modular exponentiation (e.g., the RSA algorithm), since software or hardware must in any case be present for performing the desired operations.

4.2.1 Jueneman's methods. We start by considering various methods proposed by Jueneman. In 1982, Jueneman proposed the following hash function, which he called the quadratic congruential manipulation detection code (QCMDC) [Jueneman83b], [Jueneman83]. The message is first divided into m-bit blocks

$$M_1, M_2, \ldots, M_t$$

where M_i is regarded as a number between 0 and $2^m - 1$. Then

$$H_i = (H_{i-1} + M_i)^2 \pmod{C}$$

where + denotes integer addition, H_0 is an initializing value (which Jueneman suggests should be randomly chosen for each message and kept secret), and C is a prime satisfying $C \geq 2^m - 1$. Jueneman goes on to suggest using $m = 16$ and $C = 2^{31} - 1$ (a Mersenne prime).

The fact that H_0 needs to be secret implies that QCMDC is a keyed hash function. Note also that Jueneman, Matyas, and Meyer, [Jueneman83] suggest that an additional secret key could be added on to the front of the message as M_1 (the actual message starting at M_2). In fact, the original proposed use of QCMDC was not for digital signature but for simple message authentication; however, the requirements for the function remain very similar.

Because of a variety of birthday attacks that they had discovered in conjunction with Coppersmith, by 1985 Jueneman, Matyas, and Meyer [Jueneman85] were suggesting computing the QCMDC four times for a message (using the digest obtained from the ith iteration as the initializing value for the $(i + 1)$th iteration) and then concatenating the results to obtain a 128-bit digest. However, as discussed in [Jueneman87b], the four-times-iterated version of the QCMDC is still subject to the type of attack devised by Coppersmith in 1985 [Coppersmith86] to attack the doubly iterated RMDP hash function (see Section 4.1.4, above).

As a result of Coppersmith's work, Jueneman, Matyas, and Meyer proposed a revised function called QCMDC Version 4 (QCMDCV4) ([Jueneman83], [Jueneman87b], [Jueneman87]). This is essentially a more complex version of the four-times-iterated QCMDC, although this time the four iterations are done in parallel and are cross-linked. Although designed to resist attack, Merkle [Merkle89a] reports that this function has also been broken by Coppersmith; details of Coppersmith's latest attack do not appear to have been published as yet.

4.2.2 The TeleTrust/Open Shop Information System method. Proposals similar to those of Jueneman's QCMDC were published as early as 1980 by Davies and

Price [Davies80]; an elaborated version is described in their 1985 paper [Davies85]. They suggest dividing the messages into blocks of $(m - d)$ bits, say

$$M_1, M_2, \ldots, M_t,$$

and then computing

$$H_i = (H_{i-1} \oplus M_i)^2 \pmod{C}$$

where $h_0 = 0$ and C is of the order of 2^m. The message digest is then H_t. Note that this differs from Jueneman's QCMDC in two significant ways. First, exclusive-or is used instead of integer addition, and second, Davies and Price suggest the use of a much larger modulus. They suggest choosing m to be 512 and d to be 64.

The reason for choosing message blocks significantly smaller than the modulus is to "add redundancy" and thereby prevent the sort of attack that works on the CBC-MAC (see Section 4.1.1). Thought of in these terms, the effect of requiring M_i to be at most $2^{m-d} - 1$ ensures that the most significant d bits of M_i are set to zero. More specifically, if this redundancy was not present, then, given a prespecified digest, all but one block of the message could be chosen arbitrarily and the remaining block chosen to make the digest "come out right."

Unfortunately, choosing the redundancy in this way does not make the hash function collision-free, as shown by an attack due to Jung and given in Girault [Girault88]. This attack uses ideas previously published by De Jonge and Chaum [DeJonge86]. Alternative simple methods for adding redundancy such as requiring a fixed number of zeros to be at the least significant end of M_i or requiring fixed numbers of ones at either end of M_i also fail to achieve the desired objective ([DeJonge86] and [Girault88]).

As a result a more complex method of adding redundancy was devised as part of the European Open Shop Information System (OSIS)-TeleTrust project, [OSIS85]. It was also subsequently quoted in a non-normative annex of CCITT Recommendation X.509 [X.509] and was proposed for adoption as an international standard [DP10118]. The hash function operates as follows.

First suppose that C (the modulus) satisfies $2^{8m} \leq C$, that is, C contains at least $8m + 1$ bits. The message is divided into blocks of $4m$ bits

$$M_1, M_2, \ldots, M_t$$

and the message digest is then computed by

$$H_i = (H_{i-1} \oplus R(M_i))^2 \pmod{C}$$

where $R(M_i)$ denotes the $8m$-bit block obtained from M_i by adding blocks of four 1's between every set of 4 bits of M_i, and $H_0 = 0$. The message digest is then H_t. Note that the draft International Standard DP10118 [DP10118] allows the exclusive-or operation to be replaced with integer addition, as in the Jueneman QCMDC hash function.

Note also that the version specified in draft standard DP10118 allows H_0 to be set to an initializing value I, which may be randomly chosen (instead of being fixed at 0). As pointed out by Jefferies and Walker [Jefferies88], this allows the construction in a very simple way of two different messages having the same hash (given that the factorization of the modulus is known), albeit that the two initializing values are different. This means that if a random initializing value is used then it must be regarded as part of the message digest, and signed with the digest.

However, all these efforts to standardize the OSIS hash function have been nullified by recent work of Coppersmith [Coppersmith89]. Coppersmith has been able to show that this hash function is not secure when used in conjunction with RSA signatures (and probably any other signature technique based on modular exponentiation). His cryptanalytic methods are based on the observation that the redundancy method used in the hash function is "almost invariant" under integer multiplication by 256, that is, left-shifting by 8-bit positions. This essentially discredits the hash function since it was always intended for use with RSA signatures. Indeed, the advantage of using modular arithmetic for computing the digest disappears when other signature techniques are in use.

Before proceeding note that Girault [Girault88] gives a brief review of other hash functions based on modular exponentiation, including the use of exponents other than 2. However, none of the possibilities given by Girault appears any more promising than the schemes discussed above.

4.2.3 Conclusions. As can be seen from the above discussion, hash functions based on modular squaring have a fairly awful history; indeed, there appear to be no public proposals for hash functions of this type that have not been discredited. One reason for wishing to construct hash functions using modular exponentiation, namely, the fact that the signature algorithm may also use modular exponentiation, may in fact be a good argument *against* their adoption. Coppersmith's attack on the OSIS hash function makes use of the fact that both the hash function and signature technique (namely, the RSA algorithm) are based on modular arithmetic.

It would therefore seem prudent to use hash functions whose method of computation has very little in common with the method of computation used in the signature function. This would mitigate against the use of hash functions based on modular squaring when used with either the RSA or Fiat–Shamir algorithms. Moreover, in other situations the relatively high complexity of computing modular exponents would appear to make the use of such hash functions an unlikely choice. Overall, such hash functions appear to have a very limited future.

4.3 Other Hash Functions

We conclude our discussion of practical examples of hash functions by considering some examples that do not fit so easily into the above two categories. However, on close examination, two of the schemes discussed, namely, the schemes of Rivest and Merkle, can be regarded as being based on the use of novel block ciphers.

We start by mentioning two schemes proposed by Rivest and RSA Data Security, Inc. Both these schemes produce a 128-bit digest from a message of arbitrary length. However, the two methods are very different in their mode of operation, although they are both designed to be implemented very fast in software. The first, called MD2, appears in Internet RFC 1115 [RFC1115]. MD2 has the property that it processes a message 1 byte at a time. The second, called MD4, was posted in February 1990 on the "usenet" electronic bulletin board; this function processes a message 512 bits at a time. Both algorithms are based on complex nonlinear functions which practical experience indicates are difficult to invert, similar perhaps to the philosophy used to design DES-like ciphers. Hence, while they appear secure, formal proofs of the security of these hash functions are unlikely to be forthcoming.

Merkle's "Snefru" algorithms share some of the same "intuitive" design characteristics. In 1989, Merkle published the first version of his Snefru algorithm [Merkle89b] which is a keyed hash function; however, it may also be used as an unkeyed hash function. Recently (November 1989) Merkle released Snefru Version 2.0 [Merkle89b], a modified version, and software implementations of this algorithm are available from Merkle at Xerox PARC. Snefru Version 2.0 has a number of versions, including the option of producing either 128- or 256-bit digests. In April 1990, Merkle announced on Internet that Eli Biham has found two different messages hashing to the same 128-bit value using one version of Snefru. Although other versions of Snefru remain unbroken, this new attack must cast some doubt on the security of all versions of Snefru (particularly while Biham's attack remains unpublished).

Given the demise of Snefru, MD4 appears the most promising candidate for future use; of course, before it is widely adopted it needs to withstand the test of time. It is interesting to note that another hash function of the Snefru/MD2/MD4 type already exists as an international standard! This is ISO 8731-1 [ISO8731-1], adopted as long ago as 1987. Unfortunately, this algorithm was not designed with digital signatures in mind and therefore only produces a 32-bit digest using a 64-bit key.

To complete this discussion we mention certain other work which is of theoretical interest. Damgard [Damgard88] and Gibson [Gibson88a] have discussed examples of hash functions that are in some sense provably collision-free, although at the expense of being rather impractical. Godlewski and Camion [Godlewski89] consider hash functions based on error-correcting codes and "random knapsacks," although their work appears of little immediate practical significance. As in other areas of cryptography, it remains to be seen whether the "provably secure" techniques will become the main practical option.

5 APPLICATIONS FOR DIGITAL SIGNATURES

We conclude this chapter by considering some of the more significant applications of digital signatures.

5.1 Public Key Certification

It has been recognized since their invention that public key cryptosystems offer considerable advantages over conventional cryptosystems when used for key management. Their use for general data encryption is less attractive, particularly as the most widely accepted public key cryptosystem, that is, the RSA algorithm, is nontrivial to implement, and on conventional personal computers (PCs) offers relatively low throughput speeds even when implemented in optimized software.

There is an obvious advantage to the use of public key cryptography for key management in a network of communicating entities that do not have prearranged pairwise keying relationships. If conventional cryptography is used, then one or more online key distribution centers (KDCs) are needed. On the other hand, if public key cryptography is used then every user's public key can be stored in one or more publicly available lists, and no active intervention by third parties is required.

This latter scenario fails to mention one very important requirement, namely, that the user of a public key must have some means of verifying its authenticity. One solution to this problem is the use of *key certification*, an idea that, according to Denning

[Denning83], was originally proposed by Kohnfelder [Kohnfelder78] in 1978. A discussion of key certification may also be found in Davies [Davies83].

In this system, every user's public key is signed by a certification authority (CA); this signed key is then stored in the public list of keys. This CA must be trusted to only sign valid keys, and its public verification transformation must be known (in a trusted sense) to all users of the system. The key together with the signature is then usually referred to as a certificate for that user.

In practice it is necessary for the certificate to contain the name of the key owner as well as the public key itself. In addition, the certificate may also contain an expiration date and/or an identifier for the algorithm with which the public key is to be used ([Denning83], [X.509]). A typical certificate might then have the form

$$A, p_A, T, I, S_C(A, p_A, T, I)$$

where A is the name of the user, p_A is A's public key, T is the expiration date for p_A, I is the identifier of the algorithm with which p_A is to be used, and S_C is the secret signature transformation of the CA C. Note that the importance of the use of one-way hash functions to prevent manipulation of signed data is pointed out in a paper of Gordon [Gordon85], in which an attack on certificates created without the use of a hash function is described.

Certification of public keys appears to be one of the most promising application areas for digital signatures. The 1988 version of the CCITT X.500 Directory Recommendations specifies how public key certificates may be stored in user directory entries; see, in particular, CCITT Recommendation X.509 [X.509]. The idea of digital signature-based public key certificates has also been adopted to provide key management for Internet electronic mail security. Internet RFC 1114 [RFC1114] specifies the use of RSA signatures for certifying RSA public keys.

CCITT Recommendation X.509 also makes provision for the case where more than one CA is used. This is done by allowing CAs to produce certificates for each other's public verification transformations. Sequences of such *cross-certificates* can be used to enable a user to obtain an authenticated copy of a CA's public verification transformation, and hence check a key certificate produced by that CA. For further discussion see, for example, [Mitchell89b].

It is intriguing to note that, unlike Internet RFC 1114, the method to be used for digital signatures is not completely specified in CCITT X.509 [X.509], although it is mandated that the secret signature process should be identical to the decryption process for a public key cryptosystem (see Section 2.6.1). As noted in [I'Anson90], this unnecessarily restricts the choice of signature algorithm.

5.2 Authentication Using Digital Signature

In computer networks it is often necessary for communicating parties to verify one another's identity. Traditionally this is done by the use of passwords; however the security offered by passwords used in the standard way is very limited.

One alternative is the use of cryptographic authentication protocols, standards for which are now emerging. Some of the most important of these protocols are based on the use of digital signatures. CCITT Recommendation X.509 [X.509] specifies three different protocols for authentication, all based on the use of digital signatures. All these protocols are based on the use of a cryptographic data structure called a *token*.

Like a certificate, a token is merely a series of data items with a signature appended. However, tokens are always specific to a single communication between two parties, that is, when required a token is generated by an originator for transmission to a single recipient. The general form of the token specified in X.509 for transmission from user A to user B is:

$$B, D, S_A(B, D)$$

where B designates the name of the recipient, D designates any data that are to be sent as part of the authentication protocol, and S_A designates A's secret signature transformation.

One of the protocols specified in X.509 (*three-way authentication*) has the following general form:

1. A sends to B: $B, R_A, S_A(B, R_A)$ where R_A is a random number.
2. B verifies the signature and checks that B's name is in the token.
3. B sends to A: $A, R_A, R_B, S_B(A, R_A, R_B)$ where R_B is another random number.
4. A verifies the signature, checks that A's name is in the token, and checks for the presence of R_A (protecting against replays).
5. A sends to B: $B, R_B, S_A(B, R_B)$.
6. B verifies the signature, checks that B's name is in the token, and checks for the presence of R_B (protecting against replays).

At the end of this process A and B are convinced of each other's identity. With small modifications the above protocol can also be used to exchange secret keys; these keys can then be used to protect any subsequent exchange of data. Note that the above protocol is a corrected form of the one given in X.509 [X.509]. Most significantly, in step (5) the form given in the standard does not include the name of B; apart from contravening the definition of token this seriously weakens the protocol, as has been pointed out by Burrows, Abadi, and Needham [Burrows89].

Modified forms of two of the three authentication protocols in X.509 are also under consideration for adoption as ISO standards [DP9798–3] (including the one listed above). Unfortunately, although the error in step (5) has been corrected, further shortcomings have been inherited from X.509 in the versions allowing secret key transfer ([Burrows89], [I'Anson90]).

5.3 Electronic Mail Security Based on Digital Signatures

The 1988 versions of the X.400 series of recommendations support a variety of security services, based to a considerable extent on the use of digital signatures and public key cryptosystems; see, for example, [Mitchell89b]. Key management is based on the use of public key certificates, as specified in X.509 (see Section 5.1). Provision of end-to-end security services are almost all based on the use of structures called message-tokens, whose general form is similar to that of the token described in Section 5.2. Tokens are also present in the authentication protocols used by pairs of communicating X.400 entities.

The X.400 approach to key management, namely, the use of public key certificates, has also been followed in the latest triplet of Internet request for comments (RFCs) providing a security option for Internet electronic mail ([RFC1113], [RFC1114], [RFC1115]).

5.4 Resolution of Disputes

A digital signature on a message provides lasting evidence of the content and origin of that message. It therefore finds application where a dispute may arise between sender and receiver over what message (if any) was sent. The digital signature may be presented as evidence to a referee, who can then use it to settle the dispute. Signatures also find application where the sender and receiver are not in dispute but the receiver may wish to save the message and use it at a later date with the assurance that it has not been modified in the interim.

Perhaps the most obvious case where a dispute might arise is when the message constitutes a transaction between sender and receiver, such as a bank and one of its customers, which might profit either side by falsification. Typically a bank might issue its customers with a card that can then be presented to the provider of a service. Information on the card is then used to formulate a request to the bank for a transfer of funds to pay for the service; the service is only provided if the bank responds with a signed message that authorizes the transfer. The signature on the message protects the provider of the service against later repudiation of the transaction.

Since the bank transfers funds out of the customer's account, it is also reasonable for the customer to expect some protection against falsification of transactions. This can be achieved if the customer has a smart card which can provide digital signatures. In this case the request to the bank must be signed by the card, and only when this is verified will the bank send a signed authorization of funds transfer to the service provider. It now would appear that all three parties to the transaction are protected. The customer is protected since only requests signed by his card will result in transfers from his account; the bank is protected since it has the customer's signature authorizing transfer of funds, and the service provider is protected since it has the bank's signed authorization.

There is, however, a loophole in the above analysis if the electronic device that reads the customer's card and sends the request to the bank is fraudulent. There is no way of guaranteeing that the value of the transaction appearing on the device's display is the same as the value of the transaction the smart card is requested to sign! A customer could then unwittingly transfer much larger sums to the service provider than has apparently been agreed. The only solution to this problem (apart from trusting all transaction devices) is to provide a very much more sophisticated smart card that can interface directly with its owner.

A different example of digital signatures being used to settle disputes is provided by the monitoring of remote seismic observatories for nuclear weapon test ban treaties (see Simmons [Simmons88] and the chapter by Simmons "How to Insure That Data Acquired to Verify Treaty Compliance Are Trustworthy," in this volume). In this scenario, two countries have agreed to limit (or stop) underground testing of nuclear weapons. Each country has seismic observatories in the other country which can detect any noncompliance with the agreement. These observatories send messages that either confirm or deny compliance. The messages are signed so that the receiver can be sure both of their origin and that they have not been tampered with, as well as providing evidence that can be presented to a neutral body in the event of disputes between the two countries.

5.5 A Secure Telephone System

Diffie [Diffie89] describes a secure telephone system for use with Integrated Services Digital Network (ISDN) which relies on digital signatures. To make a secure telephone

call a user places a smart card in the telephone and dials the required number. When the receiving telephone answers, the two telephones perform a Diffie–Hellman exponential key exchange [Diffie76]. This exchange provides the telephones with a shared key which is a combination of secret pieces of information chosen at random by the two telephones. No eavesdropper can obtain this key from the information exchanged. This key is now used to encrypt all data subsequently exchanged between the two telephones. However, at this stage the two telephones have not verified each other's identity; to do this they use digital signatures.

The telephones now exchange public key certificates (see Section 5.1) and both parties check the signatures on the certificates, thereby obtaining verified copies of each other's public keys. To check that the other telephone and its user are the legitimate holders of these certificates, each telephone issues a challenge message. The challenge should be a piece of information that has previously been sent, such as the public information from the exponential key exchange, to ensure that the entities owning the secret keys are the same as those performing the encryption. The response to the challenge is a signed version of the challenge, which can then be checked using the public key in the certificate.

Each secure telephone must therefore contain its own secret signature key, together with a copy of the CA's public key and a certificate for its own public key (this certificate will also contain identification information for the telephone). The tamper-resistance of the telephone must therefore be sufficient to prevent extraction of the secret key, and ensure that none of the telephone secret key, the CA's public key, or the certificate can be changed. The telephone must also have a public key belonging to its owner which is used to verify commands from the owner (signed using the owner's smart card). The owner's public key may be changed by giving the phone two messages; one contains the new owner's public key signed by the old owner and the other signed by the new owner.

Before a telephone becomes operational it must be accepted by the network. The owner gives the telephone a signed command that identifies its characteristics and the identity of the network control center with which it will communicate. The telephone sends a request to become affiliated to the network along with a newly generated public key, all signed with its secret key. The network center verifies the signatures of the telephone and its owner, and if all is in order issues a certificate containing the telephone's newly generated public key. With the issue of this certificate the telephone is able to engage in authenticated key exchange with other telephones in the network. The network center saves the request and the corresponding signatures to give an audit trail of network activity.

5.6 Other Applications

Digital signatures can also be used for user authentication and identification. For this application anyone seeking access to a secure site or use of an information system could be issued with a challenge message. The required response is the user's signature on the challenge message, and only if the signature is verified is the user granted physical access or use of the system. All such responses could then be stored to provide an audit trail of authentication attempts.

Another example is in the distribution and validation of software [Merkle80]. Software updates produced at a central source can be signed and then distributed to individual sites. Each site can then verify the signature before using the new software, thereby protecting against use of erroneous or fraudulent software. Indeed, it would be possible for a signature on each program to be verified every time it is executed, thereby protecting against malicious or accidental changes.

A further application relates again to computer networks. Within a single multi-user computer system, control of access to data can normally be provided through careful design of the operating system and application software. However, when similar controls are to be applied to networks of computers, problems arise that require cryptographic solutions [Mitchell88, Mitchell89c]. In particular, service requests and/or objects passed from one machine to another may have associated access control information. In the case of service requests this information might indicate the privileges of the requestor, or in the case of a transferred object the information might list what types of user are allowed access to the object. Protection and validation of this information is therefore of great importance. Work is currently underway within ISO and other bodies to standardize access control procedures, and a proposal from the European Computer Manufacturers Association (ECMA) [ECMA89] calls for the use of access control certificates; such certificates would be lists of access control information with a digital signature appended to enable the authenticity of the information to be checked.

Note added in proof: On August 30, 1991, the U.S. National Institute of Standards and Technology (NIST) published "A Proposed Federal Information Processing Standard for Digital Signature Standard (DSS)" soliciting public comment prior to the adoption of the standard. The DSS is a variant of the El Gamal public key algorithm (Chapter 4, "Public Key Cryptography" of this volume) in a field GF(p), p a prime modulus 512 bits in size, but carrying out the signature and verification computations in a subfield GF(q), q a prime divisor of $p - 1$; $2^{159} < q < 2^{160}$. The complete specifications for the proposed DSS can be obtained from NIST at Gaithersburg, MD 20899.

References

[Adleman79] L. M. Adleman, K. Manders, and G. Miller, "On taking roots in finite fields," in *Proc. 20th Ann. IEEE Symp. Foundations Computer Sci.*, pp. 175–178. Los Angeles, CA: IEEE Computer Society Press, 1979.

[Akl83] S. G. Akl, "Digital signatures: A tutorial survey," *IEEE Computer Magazine*, pp. 15–24, Feb. 1983.

[Akl84] S. G. Akl, "On the security of compressed encodings," in *Advances in Cryptology: Proc. Crypto'83*, D. Chaum, Ed., Santa Barbara, CA, Aug. 22–24, 1983, pp. 209–230. New York: Plenum Press, 1984.

[ANSI3.92] ANSI X3.92-1981, *Data encryption algorithm*, American National Standards Institute, New York, 1981.

[ANSI3.106] ANSI X3.106-1983, *American National Standard for Information Systems—Data Encryption Algorithm—Modes of Operation*, American National Standards Institute, New York, 1983.

[ANSI9.9] ANSI X9.9-1986, *Financial Institution Message Authentication (Wholesale)*, American Bankers Association, Washington, D.C., 1986.

[ANSI9.19] ANSI X9.19-1985, *Financial Institution Retail Message Authentication,* American Bankers Association, Washington, D.C.

[Beker82] H. J. Beker and F. C. Piper, *Cipher Systems,* London: Van Nostrand Reinhold, 1982.

[Burrows89] M. Burrows, M. Abadi, and R. M. Needham, "A logic of authentication," *ACM Operating Systems Review,* vol. 23, no. 5, pp. 1–13, 1989.

[X509] CCITT X.509-1988, *The Directory—Authentication Framework,* Geneva: Consultation Committee, International Telephone and Telegraph, Dec. 1988.

[Coppersmith84] D. Coppersmith, "Fast evaluation of logarithms in fields of characteristic two," *IEEE Trans. Inform. Theory,* vol. IT-30, pp. 587–594, 1984.

[Coppersmith86] D. Coppersmith, "Another birthday attack," in *Lecture Notes in Computer Science 218; Advances in Cryptology: Proc. Crypto'85,* H. C. Williams, Ed., Santa Barbara, CA, Aug. 18–22, 1985, pp. 14–17. Berlin: Springer-Verlag, 1986.

[Coppersmith89] D. Coppersmith, "Analysis of ISO/CCITT Document X.509 Annex D," preprint, June 1989.

[Damgård88] I. B. Damgård, "Collision free hash functions and public key signature schemes," in *Lecture Notes in Computer Science 304; Advances in Cryptology: Proc. Eurocrypt'87,* D. Chaum and W. L. Price, Eds., Amsterdam, The Netherlands, April 13–15, 1987, pp. 203–216. Berlin: Springer-Verlag, 1988.

[Damgård89] I. B. Damgård, "Design principles for hash functions," in *Lecture Notes in Computer Science 435; Advances in Cryptology: Proc. Crypto'89,* G. Brassard, Ed., Santa Barbara, CA, Aug. 20–24, 1989, pp. 416–427. Berlin: Springer-Verlag, 1990.

[Davies80] D. W. Davies and W. L. Price, "The application of digital signatures based on public key cryptosystems," in *Proc. 5th Internat. Conf. Computer Commun.,* J. Salz, Ed., Atlanta, GA, Oct. 27–30, 1980, pp. 525–530. International Council of Computer Communications.

[Davies83] D. W. Davies, "Applying the RSA digital signature to electronic mail," *IEEE Computer Magazine,* vol. 16, no. 2, pp. 55–62, Feb. 1983.

[Davies84] D. W. Davies and W. L. Price, *Security for Computer Networks,* Chichester, England: Wiley, 1984.

[Davies85] D. W. Davies and W. L. Price, "Digital signatures—an update," in *Proc. 7th Internat. Conf. Computer Commun.,* J. M. Bennett and T. Pearcy, Eds., Sydney, Australia, Oct. 30–Nov. 2, 1984, pp. 843–847. Amsterdam: Elsevier/North Holland, 1985.

[DeJonge86] W. De Jonge and D. Chaum, "Attacks on some RSA signatures," in *Lecture Notes in Computer Science 218; Advances in Cryptology: Proc. Crypto'85,* H. C. Williams, Ed., Santa Barbara, CA, Aug. 18–22, 1985, pp. 18–27. Berlin: Springer-Verlag, 1986.

[Denning83] D. E. Denning, "Protecting public keys and signature keys," *IEEE Computer Magazine,* vol. 16, no. 2, pp. 27–35, 1983.

[Denning84] D. E. Denning, "Digital signatures with RSA and other public-key cryptosystems," *Commun. ACM*, vol. 27, pp. 388–392, 1984.

[Diffie76] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inform. Theory*, vol. IT-22, pp. 644–654, 1976.

[Diffie89] W. Diffie, "Digital signatures, electronic negotiations and tamper resistant audit trails," paper given at *COMPSEC 89*, (Computer Security) London, Oct. 1989.

[ECMA89] ECMA, *Security in Open Systems—Data Elements and Service Definitions*, ECMA/TC32/TG9, final draft of July 1989 (output of 12th (Oslo) meeting). Geneva: European Computer Manufacturers Association.

[El Gamal85] T. El Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inform. Theory*, vol. IT-31, pp. 469–472, 1985.

[Fiat87] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Lecture Notes in Computer Science 263; Advances in Cryptology: Proc. Crypto'86*, A. M. Odlyzko, Ed., Santa Barbara, CA, Aug. 11–15, 1986, pp. 186–194. Berlin: Springer-Verlag, 1987.

[Gibson88a] J. K. Gibson, "A collision free hash function and the discrete logarithm problem for a composite modulus," preprint, 1988.

[Gibson88b] J. K. Gibson, "Intertwining a hash function with a public key cryptosystem, Fiat-Shamir style," preprint, 1988.

[Girault88] M. Girault, "Hash-functions using modulo-N operations," in *Lecture Notes in Computer Science 304; Advances in Cryptology: Proc. Eurocrypt'87*, D. Chaum and W. L. Price, Eds., Amsterdam, The Netherlands, April 13–15, 1987, pp. 217–226. Berlin: Springer-Verlag, 1988.

[Girault89] M. Girault, R. Cohen, and M. Campana, "A generalised birthday attack," in *Lecture Notes in Computer Science 330; Advances in Cryptology: Proc. Eurocrypt'88*, C. G. Günther, Ed., Davos, Switzerland, May 25–27, 1988, pp. 129–156. Berlin: Springer-Verlag, 1988.

[Godlewski89] P. Godlewski and P. Camion, "Manipulations and errors, detection and localization," in *Lecture Notes in Computer Science 330; Advances in Cryptology: Proc. Eurocrypt'88*, C. G. Günther, Ed., Davos, Switzerland, May 25–27, 1988, pp. 97–106. Berlin: Springer-Verlag, 1988.

[Goldwasser84] S. Goldwasser, S. Micali, and R. L. Rivest, "A 'paradoxical' solution to the signature problem," in *Proc. 25th Ann. IEEE Symp. Foundations Computer Science*, Singer Island, FL, Oct. 24–26, 1984, pp. 441–448. Los Angeles, CA: IEEE Computer Society Press, 1984.

[Gordon85] J. Gordon, "How to forge RSA key certificates," *Electron. Lett.*, vol. 21, pp. 377–379, 1985.

[I'Anson90] C. I'Anson and C. J. Mitchell, "Security defects in CCITT Recommendation X.509—The directory authentication framework," *Computer Communication Review*, vol. 20, no. 2, pp. 30–34, April 1990.

[ISO8372] ISO 8372, *Information Processing—Modes of Operation for a 64-bit Block Cipher Algorithm*, Geneva: International Organization for Standardization, 1987.

[ISO8731-1] ISO 8731-1, *Banking—Approved Algorithms for Message Authentication—Part 1: DEA*, Geneva: International Organization for Standardization, 1987.

[DIS9797] ISO DIS 9797, *Data Integrity Mechanism Using a Cryptographic Check Function Employing a Block Cipher Algorithm*, Geneva: International Organization for Standardization, 1989.

[DP9798-3] ISO DP.2 9798-3, *Peer Entity Authentication Mechanisms, Part 3: Peer Entity Mutual Authentication Mechanisms Using a Public Key Algorithm*, Geneva: International Organization for Standardization, 1989.

[DP10118] ISO DP 10118, *Hash-Functions for Digital Signatures*, Geneva: International Organization for Standardization, 1989.

[Jefferies88] N. P. H. Jefferies, private communication, 1988.

[Jueneman83b] R. R. Jueneman, "Analysis of certain aspects of output feedback mode," in *Advances in Cryptology: Proc. Crypto'82*, D. Chaum, R. L. Rivest, and A. T. Sherman, Eds., Santa Barbara, CA, Aug. 23–25, 1982, pp. 99–127. New York: Plenum Press, 1983.

[Jueneman83] R. R. Jueneman, S. M. Matyas, and C. H. Meyer, "Message authentication with manipulation detection codes," in *Proc. 1983 IEEE Symp. Security and Privacy*, G. R. Blakley and D. Denning, Eds., Oakland, CA, April 1983, pp. 33–54. Los Angeles: IEEE Computer Society Press, 1983.

[Jueneman85] R. R. Jueneman, "Message authentication," *IEEE Communications Magazine*, vol. 23, no. 9, pp. 29–40, Sept. 1985.

[Jueneman87] R. R. Jueneman, "Electronic document authentication," *IEEE Network Magazine*, vol. 1, no. 2, pp. 17–23, April 1987.

[Jueneman87b] R. R. Jueneman, "A high-speed manipulation detection code," in *Lecture Notes in Computer Science 263, Advances in Cryptology*, Proc. Crypto '86, A. M. Odlyzko, Ed., Santa Barbara, CA, Aug. 11–15, 1986. Berlin: Springer-Verlag, pp. 327–346, 1987.

[RFC1114] S. Kent and J. Linn, "Privacy enhancement for Internet electronic mail: Part II—Certificate-based key management," *Internet RFC 1114*, Washington, D.C.: Internet Activities Board Privacy Task Force, Aug. 1989.

[Kohnfelder78] L. M. Kohnfelder, "A method for certification," MIT Laboratory for Computer Science, Cambridge, MA, May 1978.

[Kranakis86] E. Kranakis, *Primality and Cryptography*, Stuttgart: Teubner/Chichester, England: John Wiley, 1986.

[Lenstra90] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard, "The number field sieve," in *Proc. 22nd ACM Symposium on Theory of Computing*, pp. 564–572, 1990.

[Lenstra90a] A. K. Lenstra and M. S. Manasse, "Factoring by electronic mail," in *Lecture Notes in Computer Science 434; Advances in Cryptology; Proc. Eurocrypt'89*, J.-J. Quisquater and J. Vandewalle, Eds., Houthalen, Belgium, April 10–23, 1989, pp. 355–371. Berlin: Springer-Verlag, 1990.

[Lenstra91] A. K. Lenstra, H. W. Lenstra, M. S. Manasse, J. M. Pollard, "The factorization of the ninth Fermat number," preprint, dated February 27, 1991.

[RFC1040] J. Linn, "Privacy enhancement for Internet electronic mail, Part I: Message encipherment and authentication procedures," *Request for Comments (RFC) 1040* (Internet Activities Board), Jan. 1988.

[RFC1113] J. Linn, "Privacy enhancement for Internet electronic mail, Part I: Message encipherment and authentication procedures," *Request for Comments (RFC) 1113* (Internet Activities Board), Aug. 1989.

[RFC1115] J. Linn, "Privacy enhancement for Internet electronic mail, Part III: Algorithms, modes, and identifiers," *Request for Comments (RFC) 1115* (Internet Activities Board), Aug. 1989.

[Matyas79] S. M. Matyas, "Digital signatures—an overview", *Computer Networks,* vol. 3, pp. 87–94, 1979.

[Meijer83] H. Meijer and S. G. Akl, "Remarks on a digital signature scheme," *Cryptologia,* vol. 7, pp. 183–186, 1983.

[Merkle78] R. C. Merkle and M. E. Hellman, "Hiding information and signatures in trap-door knapsacks," *IEEE Trans. Inform. Theory,* vol. IT-24, pp. 525–530, 1978.

[Merkle79] R. C. Merkle, Secrecy, authentication and public key systems, Ph.D. Thesis, Stanford University, 1979.

[Merkle80] R. C. Merkle, "Protocols for public key cryptosystems," in *Proc. 1980 IEEE Symp. Security and Privacy,* Oakland, CA, April 14–16, 1980, pp. 122–134. Los Angeles, CA: IEEE Computer Society Press, 1980.

[Merkle82] R. C. Merkle, *Secrecy, Authentication and Public Key Systems,* Ann Arbor, MI: University of Michigan Press, 1982.

[Merkle89a] R. C. Merkle, "One way hash functions and DES," in *Lecture Notes in Computer Science 435; Advances in Cryptology: Proc. Crypto'89,* G. Brassard, Ed., Santa Barbara, CA, Aug. 20–24, 1989, pp. 428–446. Berlin; Springer-Verlag, 1990.

[Merkle89b] R. C. Merkle, "A software encryption function," Palo Alto Research Center (PARC) Xerox preprint, July 1989; revised (V2.0 of Snefru) Nov. 1989.

[Merkle90] R. C. Merkle, "A digital signature based on a conventional encryption function," preprint, 1987. An expanded and revised version, "A certified digital signature," appears in *Lecture Notes in Computer Science 435; Advances in Cryptology: Proc. Crypto'89,* G. Brassard, Ed., Santa Barbara, CA, Aug. 20–24, 1989, pp. 218–238. Berlin: Springer-Verlag, 1990.

[Meyer82] C. Meyer and S. M. Matyas, *Cryptography—A New Dimension in Computer Data Security,* New York: John Wiley & Sons, 1982.

[Mitchell88] C. J. Mitchell and M. Walker, "Solutions to the multi-destination secure electronic mail problem," *Computers and Security,* vol. 7, pp. 483–488, 1988.

[Mitchell89c] C. J. Mitchell, "Multi-destination secure electronic mail," *Computer Journal,* vol. 32, pp. 13–15, 1989.

[Mitchell89] C. J. Mitchell, P. D. C. Rush, and M. Walker, "A remark on hash functions for message authentication," *Computers and Security,* vol. 8, pp. 55–58, 1989.

[Mitchell89b] C. J. Mitchell, M. Walker, and P. D. C. Rush, "CCITT/ISO standards for secure message handling," *IEEE J. Selected Areas Commun.*, vol. 7, pp. 517–524, 1989.

[Mitchell90] C. J. Mitchell, "Authenticating multi-cast internet electronic mail messages using a bidirectional MAC is insecure" (submitted for publication).

[OSIS85] OSIS, *OSIS Security Aspects*, OSIS (Open Shops for Information Services) European Working Group, WG1, Final Report, Oct. 1985.

[Pohlig78] S. C. Pohlig and M. E. Hellman, "An improved algorithm for computing logarithms over GF(p) and its cryptographic significance," *IEEE Trans. Inform. Theory*, vol. IT-24, pp. 106–110, 1978.

[Quisquater89] J.-J. Quisquater and M. Girault, "2n-bit hash-functions using n-bit symmetric block cipher algorithms," in *Lecture Notes in Computer Science 434; Advances in Cryptology; Proc. Eurocrypt'89*, J.-J. Quisquater and J. Vandewalle, Eds., Houthalen, Belgium, April 10–13, 1989, pp. 102–109. Berlin: Springer-Verlag, 1990.

[Rabin78] M. O. Rabin, "Digitalized signatures," in: *Foundations of Secure Computation*, R. Lipton and R. DeMillo, eds., New York: Academic, pp. 155–168, 1978.

[Rabin79] M. O. Rabin, "Digitalized signatures and public-key functions as intractable as factorization," Massachusetts Institute of Technology Laboratory for Computer Science, Cambridge, MA, Technical Report, MIT/LCS/TR-212, Jan. 1979.

[Rivest78] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, pp. 120–126, 1978.

[Schöbi83] P. Schöbi and J. L. Massey, "Fast authentication in a trapdoor-knapsack public key cryptosystem," in *Lecture Notes in Computer Science 149; Cryptography: Proc. Workshop Cryptography*, T. Beth, Ed., Burg Feuerstein, Germany, March 29–April 2, 1982. pp. 289–306. Berlin: Springer-Verlag, 1983.

[Silverman87] R. D. Silverman, "The multiple polynomial quadratic sieve," vol. 48, pp. 329–399, 1987.

[Simmons88] G. J. Simmons, "How to ensure that data acquired to verify treaty compliance are trustworthy," *Proc. IEEE* vol. 76, pp. 621–627, 1988.

[Winternitz84a] R. S. Winternitz, "Producing a one-way hash function from DES," in *Advances in Cryptology: Proc. Crypto'83*, D. Chaum, Ed., Santa Barbara, CA, Aug. 22–24, 1983, pp. 203–207. New York: Plenum Press, 1984.

[Winternitz84b] R. S. Winternitz, "A secure one-way hash function built from DES," in *Proc. 1984 IEEE Symp. Security and Privacy*, Oakland, CA, April 29–May 2, 1984, pp. 88–90. Los Angeles, CA: IEEE Computer Society Press, 1984.

[Yuval79] G. Yuval, "How to swindle Rabin," *Cryptologia*, vol. 3, pp. 187–189, 1979.