CHAPTER 4

Public Key Cryptography

JAMES NECHVATAL
National Institute of Standards and Technology
Gaithersburg, Maryland 20899

INTRODUCTION

This chapter presents a state-of-the-art survey of public key cryptography circa 1988–1990. In doing so, it covers a number of different topics including:

- The theory of public key cryptography
- Comparisons to conventional (secret key) cryptography
- A largely self-contained summary of relevant mathematics
- A survey of major existing public key systems
- An exploration of hash functions
- A survey of public key implementations in networks
- An introduction to zero-knowledge computing and probabilistic encryption
- An exploration of security issues and key sizes

In some of these areas there is significant overlap with material covered by other contributors to this volume, notably the coverage of the basic concepts of public key cryptography appearing in the chapters by J. L. Massey and by W. Diffie and of hashing functions in the chapter on digital signatures by C. Mitchell, F. Piper and P. Wild. To make each author's presentation be as complete and cohesive as possible, however, a deliberate decision was made by the editor to tolerate a reasonable degree of repetition of essential notions between the various chapters to make it possible for a reader to refer to any chapter as an essentially self-contained treatise on its subject.

It should be pointed out that the treatment of public key cryptography given in this chapter is more comprehensive than any of the expositions listed in the references. Most of the latter treat either the theory or specific systems and implementations, but not both. The viewpoint here is that the theory and practice are inseparable.

The selection of cryptosystems and hash functions mentioned in this chapter serve only to provide examples. The focus is on issues such as criteria for systems and protocols for use. These are presumably long-term, in contrast, for example, to the set of existing public key systems which is more volatile. Thus we provide information that will hopefully be of use to implementors of systems, but the frameworks we develop are versatile enough to be relevant in a variety of settings. The latter may include, for example, both electronic mail systems and electronic funds transfer systems.

The core of this exposition is Sections 1–5. Sections 1–3 cover the fundamentals of public key cryptography and the related topics of hash functions and digital signatures. The reader may also wish to refer to the chapter on these same topics by C. Mitchell, F. Piper and P. Wild for additional information about these very important (to applications) topics. Section 4 gives some examples of public key systems and hash functions. Section 5 gives some examples of actual or proposed implementations of public key cryptography.

Section 6 gives a sample proposal for a local area network (LAN) implementation of public key cryptography; it draws heavily on the work of the International Standards Organization (ISO) and the Internet Activities Board (IAB) Privacy Task Force which is summarized in Section 5. Section 7 discusses some theoretical issues related to security and choice of key size. Section 8 gives a very brief introduction to zero-knowledge protocols. Section 9 treats probabilistic encryption and identity-based public key systems.

A variety of topics is covered in the appendices, including a summary of relevant mathematics and algorithms. In the following sections, letters refer to appendices; for example, Lemma G.2.1 refers to a lemma appearing in Section 2 of Appendix G.

The author wishes to thank Dr. Ronald L. Rivest for providing many comments and suggestions, and Dr. Burton S. Kaliski, Jr., for providing information on implementations of the Rivest–Shamir-Adleman (RSA) public key system.

1 CRYPTOSYSTEMS AND CRYPTANALYSIS

Cryptography deals with the transformation of ordinary text (plaintext) into coded form (ciphertext) by encryption, and transformation of ciphertext into plaintext by decryption. Normally these transformations are parameterized by one or more keys. The motive for encrypting text is security for transmissions over insecure channels.

Three of the most important services provided by cryptosystems are secrecy, authenticity, and integrity. *Secrecy* refers to denial of access to information by unauthorized individuals. *Authenticity* refers to validating the source of a message; that is, that it was transmitted by a properly identified sender and is not a replay of a previously transmitted message. *Integrity* refers to assurance that a message was not modified accidentally or deliberately in transit, by replacement, insertion, or deletion. A fourth service that may be provided is nonrepudiation of origin, that is, protection against a sender of a message later denying transmission. Variants of these services, and other services, are discussed in [ISO-87].

Classic cryptography deals mainly with the secrecy aspect. It also treats keys as secret. In the past 15 years two new trends have emerged:

1. Authenticity as a consideration that rivals and sometimes exceeds secrecy in importance.
2. The notion that some key material need not be secret.

The first trend has arisen in connection with applications such as electronic mail systems and electronic funds transfers. In such settings an electronic equivalent of the handwritten signature may be desirable. Also, intruders into a system often gain entry by masquerading as legitimate users; cryptography presents an alternative to password systems for access control.

The second trend addresses the difficulties that have traditionally accompanied the management of secret keys. This may entail the use of couriers or other costly, inefficient, and not really secure methods. In contrast, if keys are public the task of key management may be substantially simplified.

An ideal system might solve all of these problems concurrently, that is, using public keys, providing secrecy, and providing authenticity. Unfortunately no single technique proposed to date has met all three criteria. Conventional systems such as the Data Encryption Standard (DES) require management of secret keys; systems using public key components may provide authenticity but are inefficient for bulk encryption of data due to low bandwidths.

Fortunately, conventional and public key systems are not mutually exclusive; in fact they can complement each other. Public key systems can be used for signatures and also for the distribution of keys used in systems such as DES. Thus it is possible to construct hybrids of conventional and public key systems that can meet all of the above goals: secrecy, authenticity, and ease of key management.

For surveys of the preceding and related topics see, for example, ([BRAS88], [COPP87], [DENN83], [DIFF82], [DIFF79], [KLIN79], [KONH81], [LEMP79], [MASS88], [MERK82], [POPE78], [POPE79], [SALO85], [SIMM79]). More specialized discussions of public key cryptography are given, for example, in ([DIFF88], [LAKS83], [MERK82b]). Mathematical aspects are covered, for example, in ([KRAN86], [PATT87]).

In the following, E and D represent encryption and decryption transformations, respectively. It is always required that $D(E(M)) = M$. It may also be the case that $E(D(M)) = M$; in this event E or D can be employed for encryption. Normally D is assumed to be secret, but E may be public. In addition it may be assumed that E and D are relatively easy to compute when they are known.

1.1 Requirements for Secrecy

Secrecy requires that a cryptanalyst (i.e., a would-be intruder into a cryptosystem) should not be able to determine the plaintext corresponding to given ciphertext, and should not be able to reconstruct D by examining ciphertext for known plaintext. This translates into two requirements for a cryptosystem to provide secrecy:

1. A cryptanalyst should not be able to determine M from $E(M)$; that is, the cryptosystem should be immune to ciphertext-only attacks.

2. A cryptanalyst should not be able to determine D given $\{E(M_i)\}$ for any sequence of plaintexts $\{M_1, M_2, \ldots\}$; that is, the cryptosystem should be immune to known-plaintext attacks. This should remain true even when the cryptanalyst can choose $\{M_i\}$ (chosen-plaintext attack), including the case in which the cryptanalyst can inspect $\{E(M_1), \ldots, E(M_j)\}$ before specifying M_{j+1} (adaptive chosen-plaintext attack).

To illustrate the difference between these two categories, we use two examples. First, suppose $E(M) = M^3 \mod N$, $N = p * q$, where p and q are large secret primes. Then (see Section 4) it is infeasible for a cryptanalyst to determine D, even after inspecting numerous pairs of the form $\{M, E(M)\}$. However, an eavesdropper who intercepts $E(M) = 8$ can conclude $M = 2$. Thus a ciphertext-only attack may be feasible in an instance where known- or chosen-plaintext attack is not useful.

On the other hand, suppose $E(M) = 5M \mod N$ where N is secret. Then interception of $E(M)$ would not reveal M or N; this would remain true even if several ciphertexts were intercepted. However, an intruder who learns that $E(12) = 3$ and $E(16) = 4$ could conclude $N = 19$. Thus a known- or chosen-plaintext attack may succeed where a ciphertext-only attack fails.

Deficiencies in (1), that is, vulnerability to ciphertext-only attack, can frequently be corrected by slight modifications of the encoding scheme, as in the $M^3 \mod N$ encoding above. Adaptive chosen-plaintext is often regarded as the strongest attack.

Secrecy ensures that decryption of messages is infeasible. However, the enciphering transformation E is not covered by the above requirements; it could even be public. Thus secrecy, per se, leaves open the possibility that an intruder could masquerade as a legitimate user, or could compromise the integrity of a message by altering it. That is, secrecy does not imply authenticity/integrity.

1.2 Requirements for Authenticity and Integrity

Authenticity requires that an intruder should not be able to masquerade as a legitimate user of a system. Integrity requires that an intruder should not be able to substitute false ciphertext for legitimate ciphertext. Two minimal requirements should be met for a cryptosystem to provide these services:

1. It should be possible for the recipient of a message to ascertain its origin.
2. It should be possible for the recipient of a message to verify that it has not been modified in transit.

These requirements are independent of secrecy. For example, a message M could be encoded by using D instead of E. Then assuming D is secret, the recipient of $C = D(M)$ is assured that this message was not generated by an intruder. However, E might be public; C could then be decoded by anyone intercepting it.

A related service which may be provided is nonrepudiation; that is, we may add a third requirement if desired:

3. A sender should not be able to deny later that he sent a message.

We might also wish to add:

4. It should be possible for the recipient of a message to detect whether it is a replay of a previous transmission.

1.3 Conventional Systems

In a conventional cryptosystem E and D are parameterized by a single key K, so that we have $D_K(E_K(M)) = M$. It is often the case that the algorithms for obtaining D_K and E_K from K are public, although both E_K and D_K are secret. In this event the security of a conventional system depends entirely on keeping K a secret. Then secrecy and authenticity are both provided: If two parties share a secret K, they can send messages to one another that are both private (since an eavesdropper cannot compute $D_K(C)$) and authenticated (since a would-be masquerader cannot compute $E_K(M)$). In some cases (e.g., transmission of a random bit string), this does not assure integrity; that is, modification of a message en route may be undetected. Typically integrity is provided by sending a compressed form of the message (a message digest) along with the full message as a check.

Conventional systems are also known as one-key or symmetric systems [SIMM79].

1.4 Example of a Conventional Cipher: DES

The most notable example of a conventional cryptosystem is DES. It is well-documented (e.g., [DENN83], [EHRS78], [NATI77], [NATI80], [NATI81], [SMID81], [SMID88b]) and will not be discussed in detail here, except to contrast it with other ciphers. It is a block cipher, operating on 64-bit blocks using a 56-bit key. Essentially the same algorithm is used to encipher or decipher. The transformation employed can be written $P^{-1}(F(P(M)))$, where P is a certain permutation and F is a certain function that combines permutation and substitution. Substitution is accomplished via table lookups in so-called S-boxes.

The important characteristics of DES from the standpoint of this exposition are its one-key feature and the nature of the operations performed during encryption and decryption. Both permutations and table lookups are easily implemented, especially in hardware. Thus encryption rates exceeding 40M bits/sec have been obtained (e.g., see [BANE82], [MACM81]). This makes DES an efficient bulk encryptor, especially when implemented in hardware.

The security of DES is produced in classic fashion: alternation of substitutions and permutations. The function F is obtained by cascading a certain function $f(x, y)$, where x is 32 bits and y is 48 bits. Each stage of the cascade is called a round. A sequence of 48-bit strings $\{K_i\}$ is generated from the key. Let $L(x)$ and $R(x)$ denote the left and right halves of x, and let XOR denote exclusive-or. Then if M_i denotes the output of stage i, we have

$$L(M_i) = R(M_{i-1})$$
$$R(M_i) = L(M_{i-1}) \text{ XOR } f(L(M_i), K_i)$$

The hope is that after 16 rounds, the output will be statistically flat; that is, all patterns in the initial data will be undetectable.

1.5 Another Conventional Cipher: Exponentiation

Pohlig and Hellman [POHL78] noted a type of cipher that deviated from the classic methods such as transposition and substitution. Their technique was conceptually much simpler. Let GCD denote greatest common divisor (Appendix G). Suppose $p > 2$ is a prime and suppose K is a key in $[1, p - 1)$ with $\text{GCD}(K, p - 1) = 1$ (i.e., K is relatively prime to $p - 1$). If M is plaintext in $[1, p - 1]$, an encryption function E may be defined by

$$E(M) = M^K \mod p$$

Now the condition $\text{GCD}(K, p - 1) = 1$ implies (Lemma G.1) that we can find I with $I * K \equiv 1 \pmod{p - 1}$. Note that I is not a separate key; I is easily derived from K or vice versa (Appendix H). We may set

$$D(C) = C^I \mod p$$

It may then be shown (Corollary G.2.3) that $D(E(M)) = M$, as required. Furthermore, $E(D(C)) = C$ as well; that is, E and D are inverse functions. This makes exponentiation a versatile cipher; in particular, we can encipher with D as well as E. Later we will note that this can be useful in authentication. However, Pohlig and Hellman used the equation above only as an example of a conventional cryptosystem. That is, since I is easily derived from K, both E and D are generated by the single, secret, shared key K. In Section 4.1 we will see that if p were replaced by a product of two primes, derivation of I from K would be nontrivial. This would cause the key material to be split into two portions.

Despite the relative simplicity of the definitions of E and D, they are not as easy to compute as their counterparts in DES. This is because exponentiation mod p is a more time-consuming operation than permutations or table lookups if p is large.

Security of the system in fact requires that p be large. This is because K should be nondeterminable even in the event of a known-plaintext attack. Suppose a cryptanalyst knows p, M, C, and furthermore knows that $C = M^K \mod p$ for some K. He should still be unable to find K; that is, he should not be able to compute discrete logarithms modulo p. At present there are no efficient algorithms for the latter operation: the best techniques now available take time (e.g., [ADLE79], [COPP86])

$$\exp(c((\log p)(\log \log p))^{1/2})$$

Proceeding mod p is equivalent to using the Galois field GF(p); it is possible to use GF(p^n) instead (Appendix G). There are both advantages and disadvantages to the extension. Arithmetic in GF(p^n) is generally easier if $n > 1$, and especially so in GF(2^n). On the other hand, taking discrete logarithms in GF(p^n) is also easier. The case of small n is treated in [ELGA85b]. The greatest progress has been made for the case $p = 2$ (e.g., [BLAK84], [COPP84]). In [COPP84] it is shown that discrete logarithms in GF(2^n) can be computed in time

$$\exp(c * n^{1/3} * (\log n)^{2/3})$$

For a survey of the discrete logarithm problem see, for example, [ADLE86], [COPP87], [MCCU89], [ODLY84b].

1.6 Public Key Cryptosystems

The notion of public key cryptography was introduced by Diffie and Hellman [DIFF76b]; for a history see [DIFF88]. Public key systems, also called two-key or asymmetric systems [SIMM79], differ from conventional systems in that there is no longer a single secret key shared by a pair of users. Rather, each user has his own key material. Furthermore, the key material of each user is divided into two portions, a private component and a public component. The public component generates a public transformation E, and the private component generates a private transformation D. In analogy to the conventional case, E and D might be termed encryption and decryption functions, respectively. However, this is imprecise: In a given system we may have $D(E(M)) = M$, $E(D(M)) = M$, or both.

A requirement is that E must be a trapdoor one-way function. *One-way* refers to the fact that E should be easy to compute from the public key material but hard to invert unless one possesses the corresponding D, or equivalently, the private key material generating D. The private component thus yields a "trapdoor" which makes the problem of inverting E seem difficult from the point of view of the cryptanalyst, but easy for the (sole legitimate) possessor of D. For example, a trapdoor may be the knowledge of the factorization of an integer (see Section 4.1).

We remark that the trapdoor functions employed as public transformations in public key systems are only a subclass of the class of one-way functions. The more general case will be discussed in Section 3.2.2.

We note also that public/private dichotomy of E and D in public key systems has no analogue in a conventional cryptosystem: In the latter, both E_K and D_K are parameterized by a single key K. Hence if E_K is known then it may be assumed that K has been compromised, whence it may also be assumed that D_K is also known, or vice versa. For example, in DES, both E and D are computed essentially by the same public algorithm from a common key; so E and D are both known or both unknown, depending on whether the key has been compromised.

1.6.1 Secrecy and authenticity. To support secrecy, the transformations of a public key system must satisfy $D(E(M)) = M$. Suppose A wishes to send a secure message M to B. Then A must have access to E_B, the public transformation of B (note that subscripts refer to users rather than keys in this context). Now A encrypts M via $C = E_B(M)$ and sends C to B. On receipt, B employs his private transformation D_B for decryption; that is, B computes $D_B(C) = D_B(E_B(M)) = M$. If A's transmission is overheard, the intruder cannot decrypt C since D_B is private. Thus secrecy is ensured. However, presumably anyone can access E_B; B has no way of knowing the identity of the sender per se. Also, A's transmission could have been altered. Thus authenticity and integrity are not assured.

To support authentication and integrity, the transformations in a public key system must satisfy $E(D(M)) = M$. Suppose A wishes to send an authenticated message M to B. That is, B is to be able to verify that the message was sent by A and was not altered. In this case A could use his private transformation D_A to compute $C = D_A(M)$ and send C to B. That is, A employs D_A as a de facto encryption function. Now B can use A's public transformation E_A to find $E_A(C) = E_A(D_A(M)) = M$; that is, E_A acts as a de facto decryption function. Assuming M is valid plaintext, B knows that C was in

fact sent by A, and was not altered in transit. This follows from the one-way nature of E_A: if a cryptanalyst, starting with a message M, could find C' such that $E_A(C') = M$, this would imply that he can invert E_A, a contradiction.

If M, or any portion of M, is a random string, then it may be difficult for B to ascertain that C is authentic and unaltered merely by examining $E_A(C)$. Actually, however, a slightly more complex procedure is generally employed: an auxiliary public function H is used to produce a much smaller message $S = D_A(H(M))$ that A sends to B along with M. On receipt B can compute $H(M)$ directly. The latter may be checked against $E_A(S)$ to ensure authenticity and integrity, since once again the ability of a cryptanalyst to find a valid S' for a given M would violate the one-way nature of E_A. Actually H must also be one-way; we return to this subject in Section 3.2.

Sending C or S above ensures authenticity, but secrecy is nonexistent. In the second scheme M was sent in the clear along with S; in the first scheme, an intruder who intercepts $C = D_A(M)$ presumably has access to E_A and hence can compute $M = E_A(C)$. Thus in either case M is accessible to an eavesdropper.

It may be necessary to use a combination of systems to provide secrecy, authenticity, and integrity. However, in some cases it is possible to employ the same public key system for these services simultaneously. We note that for authenticity and integrity purposes, D is regarded as an encryptor; for secrecy, E is the encryptor. If the same public key system is to be used in both cases, then $D(E(M)) = M$ and $E(D(M)) = M$ must both hold; that is, D and E are inverse functions. A requirement is that the plaintext space (i.e., the domain of E) must be the same as the ciphertext space (i.e., the domain of D).

Suppose that in addition to E and D being inverses for each user, for each pair of users A and B the functions E_A, D_A, E_B, and D_B all have a common domain. Then both secrecy and authenticity can be accomplished with a single transmission: A sends $C = E_B(D_A(M))$ to B; then B computes $E_A(D_B(C)) = E_A(D_A(M)) = M$. An intruder cannot decrypt C since he lacks D_B; hence secrecy is assured. If the intruder sends C' instead of C, C' cannot produce a valid M since D_A is needed to produce a valid C. This assures authenticity.

In actuality there are no common systems versatile enough for the last usage. In fact there is only one major system (Rivest–Shamir-Adelman[RSA]) that satisfies $E(D(M)) = D(E(M)) = M$. The lack of a common domain between two users creates a technical problem in using such a system for secrecy and authenticity. We discuss some approaches to this problem in Section 4.1.

1.6.2 Applicability and limitations. The range of applicability of public key systems is limited in practice by the relatively low bandwidths associated with public key ciphers, compared to their conventional counterparts. It has not been proven that time or space complexity must necessarily be greater for public key systems than for conventional systems. However, the public key systems that have withstood cryptanalytic attacks are all characterized by relatively low efficiency. For example, some are based on modular exponentiation, a relatively slow operation. Others are characterized by high data expansion (ciphertext much larger than plaintext). This inefficiency, under the conservative assumption that it is in fact inherent, seems to preclude the use of public key systems as replacements for conventional systems utilizing fast encryption techniques such as permutations and substitutions. That is, using public key systems for bulk data encryption is not feasible, at least for the present.

On the other hand, there are two major application areas for public key cryptosystems:

1. Distribution of secret keys
2. Digital signatures

The first involves using public key systems for secure and authenticated exchange of data-encrypting keys (DEKs) between two parties (Section 2). DEKs are secret shared keys connected with a conventional system used for bulk data encryption. This permits users to establish common keys for use with a system such as DES. Classically, users have had to rely on a mechanism such as a courier service or a central authority for assistance in the key exchange process. Use of a public key system permits users to establish a common key that does not need to be generated by or revealed to any third party, providing both enhanced security and greater convenience and robustness.

Digital signatures are a second major application (Section 3). They provide authentication, nonrepudiation, and integrity checks. As noted above, in some settings authentication is a major consideration; in some cases it is desirable even when secrecy is not a consideration (e.g., [SIMM88]). We have already seen an example of a digital signature, that is, when A sent $D_A(M)$ to B. This permitted B to conclude that A did indeed send the message. As we will note in Section 3, nonrepudiation is another property desirable for digital signatures. Public key cryptosystems provide this property as well.

No bulk encryption is needed when public key cryptography is used to distribute keys, since the latter are generally short. Also, digital signatures are generally applied only to outputs of hash functions (Section 3). In both cases the data to be encrypted or decrypted are restricted in size. Thus the bandwidth limitation of public key is not a major restriction for either application.

2 KEY MANAGEMENT

Regardless of whether a conventional or public key cryptosystem is used, it is necessary for users to obtain other users' keys. In a sense this creates a circular problem: to communicate securely over insecure channels, users must first exchange key information. If no alternative to the insecure channel exists, then secure exchange of key information presents essentially the same security problem as subsequent secure communication.

In conventional cryptosystems this circle can be broken in several ways. For example, it might be assumed that two users can communicate over a supplementary secure channel, such as a courier service. In this case it is often the case that the secure channel is costly, inconvenient, low-bandwidth, and slow; furthermore, use of a courier cannot be considered truly secure. An alternative is for the two users to exchange key information via a central authority. This presumes that each user individually has established a means of communicating securely with the central authority. Use of a central authority has several disadvantages as noted below.

In public key systems the key management problem is simpler because of the public nature of the key material exchanged between users, or between a user and a central authority. Also, alternatives to the insecure channel may be simpler; for

example, a physical mail system might suffice, particularly if redundant information is sent via the insecure (electronic) channel.

2.1 Secret Key Management

In a conventional (one-key) system, security is dependent on the secrecy of the key that is shared by two users. Thus two users who wish to communicate securely must first securely establish a common key. As noted above, one possibility is to employ a third party such as a courier; but as remarked there are various disadvantages to this implementation. The latter is the case even for a single key exchange. In practice, it may be necessary to establish a new key from time to time for security reasons. This may make use of a courier or similar scheme costly and inefficient.

An alternative is for the two users to obtain a common key from a central issuing authority (e.g., [POPE79]). Security is then a major consideration: a central authority having access to keys is vulnerable to penetration. Due to the concentration of trust, a single security breach would compromise the entire system. In particular, a central authority could engage in passive eavesdropping for a long period of time before the practice was discovered; even then it might be difficult to prove.

A second disadvantage of a central issuing authority is that it would probably need to be online. In large networks it might also become a bottleneck, since each pair of users needing a key must access a central node. If the number of users is n then the number of pairs of users wishing to communicate could theoretically be as high as $n(n-1)/2$, although this would be unlikely. Also, each time a new key is needed, at least two communications involving the central authority are needed for each pair of users. Furthermore, such a system is not robust: failure of the central authority disrupts the key distribution system.

Some of the disadvantages of a central authority can be mitigated by distributing key distribution via a network of issuing authorities. One possibility is a hierarchical (tree-structured) system, with users at the leaves and key distribution centers at intermediate nodes. However, distributing key management creates a new security problem, since a multiplicity of entry points for intruders is created. Furthermore, such a modification might be inefficient unless pairs of users communicating frequently were associated to a common subtree; otherwise the root of the tree would be a bottleneck as before.

2.2 Public Distribution of Secret Keys

It was noted by Diffie [DIFF76], Diffie and Hellman [DIFF76b], and Merkle [MERK78] that users can publicly exchange secret keys. This is not related a priori to public key cryptography; however, a public key system can be used to implement public distribution of secret keys (often referred to as public key distribution). The underlying public key system must support secrecy, for use in key encryption.

The notion of public distribution of secret keys was originated in 1974 by Merkle, who proposed a "puzzle" system to implement it [MERK78]. However, even before this work was published it was superseded by a public key scheme supporting public key distribution [DIFF76b]. This has come to be known as the Diffie–Hellman exponential key exchange. To begin with, users A and B are assumed to have agreed on a common prime p and a common primitive root g mod p (Appendix K). Then (Lemma

K.2) each number in $[1, p)$ can be expressed as $g^x \bmod p$ for some x. Both p and g may be public. Now A chooses a random $x(\mathrm{A})$ in $[0, p-1]$ and B chooses a random $x(\mathrm{B})$ in $[0, p-1]$; these are the private components of A and B, respectively. Then A computes

$$y(\mathrm{A}) = g^{x(\mathrm{A})} \bmod p$$

while B computes

$$y(\mathrm{B}) = g^{x(\mathrm{B})} \bmod p$$

These are the public components (along with the common p and g) of A and B, respectively. Finally A sends $y(\mathrm{A})$ to B and B sends $y(\mathrm{B})$ to A. Since A knows $x(\mathrm{A})$ and $y(\mathrm{B})$ he can compute

$$K = y(\mathrm{B})^{x(\mathrm{A})} \bmod p$$

We note

$$K = g^{x(\mathrm{A})x(\mathrm{B})} \bmod p$$

Now B knows $x(\mathrm{B})$ and $y(\mathrm{A})$ and can compute

$$y(\mathrm{A})^{x(\mathrm{B})} \bmod p = K$$

The secrecy of this scheme depends, as in Section 1.5, on the difficulty of computing discrete logarithms. That is, knowing p, g, y, and $y = g^x \bmod p$ for some x does not yield x. Thus, if an intruder intercepts $y(\mathrm{A})$ or $y(\mathrm{B})$ or both he cannot find $x(\mathrm{A})$ or $x(\mathrm{B})$ and hence cannot compute K.

A disadvantage of this scheme is lack of support for authenticity. If an intruder C intercepts $y(\mathrm{B})$, sends $y(\mathrm{C}) = g^{x(\mathrm{C})} \bmod p$ to B and B thinks he is receiving $y(\mathrm{A})$, B will inadvertently establish a secret key with C. That is, the underlying public key cryptosystem supports secrecy but not authentication. Thus it may be desirable to augment a secrecy-providing system with one that provides authentication. Examples of the latter will be given later.

2.3 Management of Public Components in a Public Key System

Prior to using a public key cryptosystem for establishing secret keys, signing messages, etc. users A and B must exchange their public transformations (E_A and E_B in Section 1.6), or equivalently their public components. The latter may be regarded as a key management problem. It is a simpler problem than establishment of secret keys, since public components do not require secrecy in storage or transit. Public components can, for example, be managed by an online or offline directory service; they can also be exchanged directly by users. However, authenticity is an issue as in the previous section. If A thinks that E_C is really E_B then A might encrypt using E_C and inadvertently allow C to decrypt using D_C. A second problem is integrity: any error in transmission of a public component will render it useless. Hence some form of error detection is desirable.

Regardless of the scheme chosen for public component distribution, at some point a central authority is likely to be involved, as in conventional systems. However, it may

not be necessary for the central authority to be online; exchange of public components between users need not involve the central authority. An example of how this can be implemented is given in Section 2.5. We also note there that the implications of compromise of the central authority are not as severe as in the conventional case.

Validity is an additional consideration: A user's public component may be invalidated because of compromise of the corresponding private component, or for some other reason such as expiration. This creates a stale data problem in the event that public components are cached or accessed through a directory. Similar problems have been encountered in management of multi-cache systems and distributed databases, and various solutions have been suggested. For example, users could be notified immediately of key compromises or invalidations. This may be impractical, in which case either users should be informed within a given time period of changes needed for cached information on public components, or users should periodically check with a central authority to update validity information.

2.3.1 Use of certificates. A technique to gain a partial solution to both authenticity and integrity in distribution of public components is use of certificates [KOHN78]. A certificate-based system assumes a central issuing authority S as in the secret key case. Again it must be assumed that each user can communicate securely with S. This is relatively simple in the present instance: it merely requires that each user possess E_S, the public transformation of S. Then each user A may register E_A with S. Since E_A is public, this might be done via the postal service, an insecure electronic channel, a combination of these, etc.

Normally A will follow some form of authentication procedure in registering with S. Alternatively, registration can be handled by a tree-structured system: S issues certificates to local representatives (e.g., of employing organizations), who then act as intermediaries in the process of registering users at lower levels of the hierarchy. An example of such a system is given in Section 5.4.

In any case, in return A receives a certificate signed by S (see Section 3 for a more thorough discussion of signatures) and containing E_A. That is, S constructs a message M containing E_A, identification information for A, a validity period, and so on. Then S computes $C_A = D_S(M)$ which becomes A's certificate. The latter is then a public document that both contains E_A and authenticates it, since the certificate is signed by S. Certificates can be distributed by S or by users, or used in a hierarchical system that we return to later. The inclusion of the validity period is a generalization of timestamping. The latter was not treated in [KOHN78], but Denning [DENN81] notes the importance of timestamping in guarding against the use of compromised keys.

In general, the problem of stale data is not wholly solved by timestamping: a certificate may be invalidated before its expiration date, because of compromise or administrative reasons. Hence if certificates are cached by users (as opposed to being redistributed by S each time they are used), S must periodically issue lists of invalidated certificates. Popek and Kline [POPE79] have noted that certificates are analogous to capabilities in operating systems. Management of certificates creates analogous problems, such as selective revocation of capabilities. The public nature of certificates, however, a priori precludes an analogue of a useful feature of a capability system: Users cannot be restricted to communicating with a subset of other users. If a selective capability feature is desired it must be implemented through a controlled distribution of certificates, which would be difficult and contrary to the spirit of public key.

2.3.2 Generation and storage of component pairs. Generation of public/private component pairs is an important consideration. If this service is offered by a central facility, it may result in certain advantages; for example, keys might be longer or chosen more carefully. However, this introduces the same problem noted in Section 2.1: A central authority holding users' private components would be vulnerable to penetration. Also, the central facility would have to be trusted not to abuse its holdings by monitoring communications between users. Both problems are circumvented if users generate and store their own private/public components.

2.3.3 Hardware support for key management. If users store their own components, a management problem is created. One possibility is software storage, for example, in a file with read protection. If greater security is desired, a second option is hardware keys ([DENN79], [FLYN78], [RIVE78]).

In the classic version of this scheme, each public/private key pair is stored on a pair of read-only memory (ROM) chips. The public key is revealed to the owner of the keys. However, the private key need not be known to any user, including the owner.

There are two advantages to this mode of storage, namely, the fact that neither the user nor a central authority need know the private component. We have previously noted the advantage of not having users reveal private keys to a central authority. Also, as noted in [FLYN78], it may also be desirable to protect keys from disloyal employees. Entering keys from a keyboard rather than from ROM is insecure. However, there is also an obvious disadvantage to the classic mode of hardware key storage, namely, the fact that the key manufacturer has access to the user's keys, and may retain copies. An alternative, but still classic, scheme is for the user to be provided with a means of writing to a memory chip and then sealing it from further writing.

A more modern and more secure hardware adjunct is a token, smart card, or other device that contains both memory and a microprocessor. Such a device can both generate and store user keys. Ideally, it should be implemented via a single chip that stores both cryptographic algorithms and data.

In the classic case, encryption and decryption are handled by separate units in a hardware device acting as an interface between a user's computer and the network. As noted in [DENN79], this creates a security risk: An intruder may rig the device. Also, all data entering or exiting a user's computer are automatically decrypted or encrypted. Hence all other users must know a user's public key to communicate with him. On the other hand, a user can easily store encrypted data in a central facility: By setting a toggle switch, a user enciphers data exiting his computer using his own private component. Furthermore, a form of Trojan horse attack is thwarted: If a user runs a program from an external source that attempts to have data illicitly sent back to it, the data will be encrypted and hence worthless.

If a token or smart card is used, it may be possible in the near future to generate and store keys on the same device that executes encryption and decryption algorithms. Again, security is optimal if all of these functions are combined onto one or a minimal number of chips. However, use of such auxiliary devices entails some of the same problems as in the classic case. For example, a token or card reader is needed; this device must be secure. Also, a token or smart card may be stolen, permitting the thief to masquerade as the user. Passwords (personal identification numbers [PINs]) used to access devices can substantially lessen this threat.

A classic hardware system can support signatures (see Section 3), but this requires an additional data path from a user's computer through the deciphering unit and

back to the computer. This creates an additional security risk. A second path is needed through the enciphering unit, but this is more secure.

Advanced capabilities such as signature generation should become feasible on devices such as smart cards in the next few years. This would provide an excellent solution to the problem of using private components without exposing them.

2.4 Using Public Key Systems for Secret Key Distribution

As noted above, one of the major applications of public key cryptosystems is in regard to public distribution of secret keys. Thus, a public key system can be used in conjunction with a conventional system such as DES. We have already seen an example of a public key cryptosystem implementing public key distribution in Section 2.2. Both secrecy and authentication can be provided simultaneously in the distribution process via public key systems. This goal may be achieved using a single public key system if its encryption and decryption functions are inverses, or via a hybrid of two public key systems providing secrecy and authentication separately.

The essence of this usage is very simple: A secret key is merely a particular form of message. Assuming that a system has been established for distribution of public components of users as discussed in the previous section, users can then establish secret keys at will by employing the public system for encryption and signing of secret keys. Thus the latter can be exchanged or changed without difficulty. This is in contradistinction to a system in which secret keys must be established via couriers or a central authority.

If users in a public key system generate their own private/public component pairs, it is in fact never necessary for them to use a "secure" auxiliary mechanism such as a courier service, or to share their private components with a central authority. Users may normally register their public components without use of a secure channel; and two users may use each other's public components to establish common keys for use in message encryption, again without resort to a secure channel. In contrast, in a secret key system a secure communication may be required each time a user initiates communication with a new user or a central authority, or when two users change a master key.

2.4.1 A protocol for key exchange. Suppose A and B wish to establish a shared secret key. Suppose further that they have obtained each other's public components; some ways by which this may be effected are discussed below. In any case, A may now generate a secret key K. For example, this may be a key-encrypting key to be used to exchange session keys and possibly initialization vectors if, for example, DES is used in cipher feedback mode or cipher block-chaining mode [NATI80]. Then A might form an expanded message that contains K and further data, which could include an identification for A, a timestamp, a sequence number, a random number, and so on. This added material is needed for A to authenticate himself to B in real time, and thus prevent replays. For example, Needham and Schroeder [NEED78] suggest a three-way handshake protocol:

First, A may send to B the message $M = E_B(R_A, ID_A)$, where E_B is B's public transformation, ID_A is the identification of A, and R_A is a random number. Now B can decrypt M and obtain ID_A. As noted above, M could also contain other information such as a timestamp or sequence number. Now B generates a random number R_B and sends $M' = E_A(R_A, R_B)$ to A. On decryption of M', A can verify in real time that B

has received R_A, since only B could decrypt M. Finally A sends $M'' = E_B(R_B)$ to B; when B decrypts M'' he can verify in real time that A has received R_B, since only A could decrypt M'. Thus A and B are authenticated to each other in real time; that is, each knows that the other is really there.

Now A sends $E_B(D_A(K))$ to B, who can decrypt to obtain K. This ensures both secrecy and authenticity in exchange of K.

2.5 Protocols for Certificate-Based Key Management

There are a number of different approaches to public key component management and the application to secret key distribution (e.g., [KLIN79], [NEED78]). For simplicity we assume a certificate-based system. We also assume that a system has been established for a central authority to create certificates.

2.5.1 Certificate management by a central authority. In a certificate-based public key system, one possibility is to have the central issuing authority manage certificate distribution. Suppose the authority S has created certificates C_A and C_B for A and B, respectively. These contain E_A and E_B (or equivalently, the public components of A and B) as noted in Section 2.3.1.

We may assume that A and B have cached C_A and C_B, respectively. If A has exchanged certificates with B previously, A and B may have each other's certificates cached. If not, then there are two ways in which A could obtain B's certificate. The simplest is for A to request B's certificate directly from B; this is explored in the next section. The advantage of this simple approach is that online access to a central authority is avoided.

On the other hand, if A requests B's certificate directly from B, or obtains it from a directory listing, security and integrity considerations arise. For example, the certificate A obtains may have been recently invalidated.

Thus A may wish to access B's certificate through S, assuming that the latter is online. In this event A requests C_A and C_B from S. We recall that each C has the form $D_S(M)$ where M contains an E. Thus when A receives the requested certificates, he can validate C_A by computing $E_S(C)$ and recovering E_A from the decrypted form. This validates C_B as well. Hence A may retrieve a duly authenticated E_B from C_B. The disadvantage is the requirement of an online central node; the advantage is that A is assured of the instantaneous validity of the certificate. Later we will note that this has implications for nonrepudiation.

Caching of certificates implies that authentication is not done regularly. Thus the procedure above is only necessary when two users first communicate, or when components are compromised or certificates invalidated. As long as the public components involved are valid, secret keys can be exchanged at will, although a handshake protocol may still be used to ensure real-time authenticity and integrity.

An advantage of this scheme is that the entire process is conducted over insecure channels with excellent security. A second advantage is that distribution of certificates by a central authority assures that certificates are valid at time of receipt.

A disadvantage is that, as in Section 2.1, the central authority may be a bottleneck. The severity is mitigated by the fact that a secure channel is not needed, but the essential problem is similar.

A more significant disadvantage arises from the concentration of trust in one entity [KLIN79]. The security of the central authority might be compromised, for

example, by penetration. The fact that the central authority does not generally access users' private components means that existing messages are not compromised, as might be the case in a secret key system [DIFF82]. If an intruder accesses the central authority's private component, however, the intruder could now forge certificates.

Diffie [DIFF88] notes that the latter situation is not catastrophic. Suppose A requests B's certificate from the penetrated central authority. The intruder might substitute his own public transformation for E_B, then forge the central authority's signature. The intruder could then decrypt messages sent to B by A. However, if the intruder's eavesdropping is passive, B will be unable to decrypt, and the masquerade will be short-lived. To continue the deception, the intruder must intercept, decrypt, and reencrypt the message using E_B and then pass it along to B. When B replies to A the procedure must be repeated. This involves wiretaps and message deletions that, according to Diffie, will most likely expose the intruder.

A more blatant approach would be for the intruder merely to intercept messages intended for B. In either event, the issuance of false certificates is a practice that presumably would be exposed in a short period of time, exposing the intruder and limiting the damage. Also, forged certificates would provide proof of criminal activity. In contrast, we noted above that in a conventional system, compromise of the central authority would permit passive eavesdropping which could proceed undetected for a long period and might be difficult to prove.

A more detailed discussion of recovery from compromised keys may be found in [DENN83b]. There Denning also suggests that the central authority keep a log of all transactions to aid in recovery. In passing she also notes that a potential weak link in the certificate system is that for the central authority to properly identify users, it must assume that users' hosts are secure. Denning also observes that hosts may be vulnerable to Trojan horse attacks in the form of subversion of their operating systems or network interfaces.

Merkle [MERK82b] has suggested a tree-structured system as a means of coping with the compromise of a central authority. However, this scheme is not practical for large networks since the tree must be restructured each time the user population changes.

2.5.2 Decentralized management. Users may be responsible for managing their own certificates. In this event the protocol is much simpler. When A wishes to initiate communication (e.g., exchange of a secret key) with B, he sends a message to B containing A's certificate, A's identification, and other information such as a date, random number, and so on, as described in the protocol in the previous section. This message also requests B's certificate. On completion of the certificate exchange, employing some protocol such as the handshake above, A and B will possess each other's authenticated certificates. A can validate the certificate C_B by computing $E_S(C_B)$ as usual. Then E_B may be retrieved. The certificate must contain information properly identifying B to A, so that an intruder cannot masquerade as B. The certificate must also have a validity period. In turn B may proceed similarly.

The central authority must periodically issue lists of certificates that have become invalid before their expiration dates due to key compromise or administrative reasons. It is likely that in a large system this would be done, for example, on a monthly basis. Hence a user receiving a certificate from another user would not have complete assur-

ance of its validity, even though it is signed by S. Thus this system trades higher efficiency for some security loss, compared to the previous scheme.

For greater assurance of validity, a user could access a centrally managed list of invalid certificates; presumably these would be very current. This would require online access to a central facility, which would create the same type of bottleneck we have noted previously. However, the accesses would be very quick, since presumably only a certificate sequence number would be transmitted.

Yet another online possibility would be for the central authority to enforce coherence by a global search of cached certificates each time a certificate is invalidated. Again there is a trade-off of validity assurance and efficiency.

2.5.3 A phone book approach to certificates. Some of the features of the previous schemes could be combined in a phone-book approach, using an electronic equivalent such as a floppy disk containing certificates. This would optimize ease of use since a user could communicate securely with another by accessing the latter's certificate very rapidly. However, again the central authority would have to issue "hot lists." Periodic distribution of the compilations of certificates would be a separate management process.

Security of such a scheme is clearly in question [KLIN79]. Phone book entries might contain errors, and if phone books are public entries these errors could be altered.

3 DIGITAL SIGNATURES AND HASH FUNCTIONS

Digital signatures were introduced by Diffie and Hellman [DIFF76b]; for surveys see, for example, [AKL-83], [POPE79], [RABI78]. A digital signature is the electronic analogue of a handwritten signature. A common feature is that they must provide the following properties:

1. A receiver must be able to validate the sender's signature.
2. A signature must not be forgeable.
3. The sender of a signed message must not be able to repudiate it later.

We have already seen an example of the usage of digital signatures, namely, when a central authority signs certificates. In this section we are concerned with the signing of arbitrary messages.

A major difference between handwritten and digital signatures is that a digital signature cannot be a constant; it must be a function of the document that it signs. If this were not the case then a signature, due to its electronic nature, could be attached to any document. Furthermore, a signature must be a function of the entire document; changing even a single bit should produce a different signature. Thus a signed message cannot be altered.

There are two major variants of implementation:

1. True signatures
2. Arbitrated signatures

In a true signature system, signed messages are forwarded directly from signer to recipient. In an arbitrated system, a witness (human or automated) validates a signature and transmits the message on behalf of the sender. The use of an arbitrator may be helpful in event of key compromise as noted below.

The notion of authentication by some form of signature can be traced back as far as 1952, as noted by Diffie [DIFF88]. Cryptography was used by the Air Force to identify friendly aircraft through a challenge/response system. The protocols utilized influenced the development of public key cryptography by Diffie and Hellman [DIFF76b].

As we note below, hash functions are useful auxiliaries in this context, that is, in validating the identity of a sender. They can also serve as cryptographic checksums, thereby validating the contents of a message. Use of signatures and hash functions can thus provide authentication and verification of message integrity at the same time.

Numerous digital signature schemes have been proposed. As noted in [DAVI83], a major disadvantage of signature schemes in conventional systems is that they are generally one-time schemes. A signature is generated randomly for a specific message, typically using a large amount of key material, and is not reusable. Furthermore, later resolution of disputes over signed documents requires written agreements and substantial bookkeeping on the part of the sender and receiver, making it more difficult for a third party to adjudicate. A conventional scheme was noted in [DIFF76b] (the Lamport–Diffie one-time signature) and is refined in [MERK82]. Some other conventional schemes are DES-based.

Public key schemes supporting authentication permit generation of signatures algorithmically from the same key repeatedly, although the actual signatures are of course different. That is, in a public key scheme, signatures are a function of the message and a long-term key. Hence key material can be reused many times before replacement. This obviates the necessity of special written agreements between individual senders and receivers. It also makes it easy for a third party to resolve disputes (e.g., involving repudiation) later, and simplifies storage and bookkeeping. As we note below, the bandwidth limitation of public key systems is unimportant, due to the use of hash functions as auxiliaries.

In passing we remark that Goldwasser, Micali, and Rivest [GOLD88] have proposed a nondeterministic public key signature scheme that incorporates an aspect of conventional schemes: A message does not have a unique signature. This scheme is intended to deflect adaptive chosen-text attacks, in which an attacker is permitted to use a signer as an oracle. In such an attack, the attacker specifies a sequence of messages to be signed, and is able to study all previous signatures before requesting the next. In the Goldwasser–Micali–Rivest (GMR) scheme, it can be proved that an attacker can gain no information by studying any number of signatures no matter how they are generated.

This security gain is offset, however, to some extent by data expansion (high ratio of signature to message size) and a negative effect on nonrepudiation. As in the case of the one-time schemes discussed above, the signatures generated in nondeterministic public key schemes tend to be large in comparison to signatures produced by deterministic public key schemes (i.e., those that produce a unique signature for a given message). Adjudication of disputes is made difficult by increased bookkeeping and the necessity of storing large databases of messages and their signatures.

In the following sections, we discuss only deterministic public key signature schemes, in which a message has a unique signature.

3.1 Public Key Implementation of Signatures

We have noted above that a public key system can be used for authentication. There are several distinctive features of a public key implementation of signatures, including:

1. The possibility of incorporating both secrecy and authenticity simultaneously, assuming that the system supports both services.
2. The reuse of key material in generating signatures algorithmically.
3. Nonrepudiation of sending is essentially a built-in service.

These features make public key implementation of signatures very efficient and versatile.

3.1.1 Signing messages. We recall that A signs a document M by sending $S = D_A(M)$ to B; B validates A's signature by computing $M = E_A(S)$. That is, A's signature can be validated. We also noted above that because E_A is a trapdoor one-way function, it should not be possible for an intruder to find S' such that $M = E_A(S')$ for a given message M. Thus A's signature cannot be forged. Also, if A attempts to repudiate the M sent to B above, B may present M and S to a judge. The judge can access E_A and hence can verify $M = E_A(S)$; assuming the trapdoor D_A is indeed secret, only A could have sent S. Thus a priori we have nonrepudiation (but see Section 3.1.2, below).

For both authenticity and secrecy, as before A may send

$$S = E_B(D_A(M))$$

to B; B computes

$$M = E_A(D_B(S))$$

For nonrepudiation, B retains $D_B(S) = D_A(M)$. Again a judge can use E_A to find M. This again assumes common domains for the E's and D's; in Section 4.1 we will note an example of how this point may be treated in practice. In passing we remark that the preceding schemes satisfy another desirable property: In the adjudication process, they do not compromise security by exposing private components to a judge.

3.1.2 The issue of nonrepudiation. Nonrepudiation, that is, preventing senders from denying they have sent messages, is contingent on users keeping their private components secret (e.g., [NEED78], [POPE79]). In the discussion above, if D_A should be compromised, then A might be able to repudiate messages sent even before the compromise. This is an administrative issue, and ultimately a matter for litigation; hence it is beyond the scope of cryptography per se. However, some partial solutions have been proposed that can be incorporated into protocols. Most of these involve use of some form of arbitrator, as noted in [DEMI83].

For example, in [POPE79], notary public machines are employed as arbitrators. These sign messages on top of the senders' signatures. A sender then cannot claim that a message sent via the arbitrator was forged. However, a sender can still claim that his private component was compromised in the past without his knowledge, and a forged message was notarized. A partial solution is for the notary to send a copy of a signed message back to the sender; in this case the notary must know the current address of the sender.

A general protocol for usage of an arbitrator A when U is sending a message M to R is given in [AKL-83]:

1. U computes $S_1 = E_R(D_U(M))$.
2. U computes a header m = identifying information.
3. U computes $S_2 = D_U(m, S_1)$.
4. U sends m and S_2 to A.
5. A decrypts S_2 and validates the identifying information in m, thereby verifying the origin of M.
6. A computes $M' = (m, S_1, \text{timestamp})$.
7. A computes $S' = D_A(M')$.
8. A sends S' to R.

As noted above, a copy of S' may also be sent to U.

The obvious disadvantages of an arbitrator system are inconvenience and loss of security. In particular, as noted in [POPE78], this type of scheme violates a desirable criterion for network protocols, namely, point-to-point communication. It also requires trusted software, which is undesirable.

In [BOOT81] the use of a central authority is suggested for this purpose. In this scheme, the receiver of a message sends a copy to the central authority. The latter can attest to the instantaneous validity of the sender's signature; that is, it has not been reported that the sender's private component has been compromised at the time of sending. The value of such testimony is mitigated by the necessity of users rapidly reporting key compromise to the central authority. Also, the central authority becomes a bottleneck.

Another partial solution involves timestamps ([DENN81], [MERK82b]). This again may involve a network of automated arbitrators, but very simple in nature, since they need merely timestamp messages. In contrast to the notary publics above, however, in [MERK82b] it is suggested that receivers obtain timestamps. If a receiver needs to be sure of the validity of a signature, he may check the validity of the sender's private component by checking with a central authority. As long as the received message is timestamped before the validity check, the receiver is assured of nonrepudiation. To be legally cohesive, however, this system requires the sender to be responsible for signing until a compromise of his private component is reported to the central authority. In analogy to lost credit cards, this may not be a desirable system. Also, it requires an online central authority and real-time validity checks and timestamps, which may again create bottlenecks. Furthermore, such schemes require a networkwide clock and are vulnerable to forgery of timestamps, as noted in [BOOT81].

If users are permitted to change their components, a central authority should retain past components for use in disputes that may arise later. Neither compromise nor change of components of a user should cause catastrophic problems in practice, since credit card systems have established precedents for protocols, both administrative and legal. For example, as noted above, conventions have been established for treatment of cases of lost or stolen credit cards; presumably analogous procedures could be established for component compromise.

3.1.3 The issue of proof of delivery. The literature on public key protocols concentrates on the issues of validity of signatures and the effects of key compromise on

nonrepudiation. However, DeMillo and Merritt [DEMI83] note that the inverse of a signature, namely, protection against the recipient of a message denying receipt, may also be a desired feature of a system. It is mentioned as an optional security service in [ISO-87].

Both nonrepudiation and proof of delivery must be implemented via adjudicable protocols, that is, protocols that can be verified later by a third party. In [DEMI83] it is noted that nonarbitrated adjudicable reception protocols are difficult to design. A simple handshaking protocol might proceed as follows:

1. A computes $X = E_B(D_A(M))$.
2. A sends X to B.
3. B computes $M = E_A(D_B(X))$.
4. B computes $Y = E_A(D_B(M))$.
5. B acknowledges receipt of M by sending Y to A.

If this protocol is standard procedure then A can assume nonreception unless he receives acknowledgment from B. However, to serve as an adjudicable proof-of-reception protocol, B is required to acknowledge anything A sends. Then an intruder C could proceed as follows:

1. C intercepts $Y = E_A(D_B(M))$.
2. C sends Y to A.
3. A thinks this is a legitimate message from C, unrelated to his communication with B. Following protocol:
   a. A computes $M' = E_C(D_A(Y))$.
   b. A computes $Z = E_C(D_A(M'))$.
   c. A acknowledges receipt of M' by sending Z to C.
4. C computes $E_B(D_C(E_A(D_C(Z)))) = E_B(D_C(M')) = E_B(D_A(Y)) = M$.

This of course occurs because of step 3, in which A is required by protocol to acknowledge M' = gibberish. This shows that such an automatic protocol may be undesirable. On the other hand, selective acknowledgment might have an adverse effect on adjudicability.

3.2 Hash Functions and Message Digests

We noted that public key systems generally encrypt more slowly than conventional ciphers such as DES. Other digital signature schemes are also relatively slow in general. Furthermore, some schemes produce signatures comparable in size to, and in some cases larger than, the messages they sign. This results in data expansion and effectively lower bandwidth of transmission. Thus it is usually not desirable to apply a digital signature directly to a long message. On the other hand, we remarked that the entire message must be signed. This is seemingly contradictory, but a heuristic solution can be obtained by using a hash function as an intermediary.

A hash function H accepts a variable-size message M as input and outputs a fixed-size representation $H(M)$ of M, sometimes called a message digest [DAVI80]. In general, $H(M)$ is much smaller than M; for example, $H(M)$ might be 64 or 128 bits,

whereas M might be a megabyte or more. A digital signature may be applied to $H(M)$ in relatively quick fashion. That is, $H(M)$ is signed rather than M. Both M and the signed $H(M)$ may be encapsulated in another message which may then be encrypted for secrecy. The receiver may validate the signature on $H(M)$ and then apply the public function H directly to M and check to see that it coincides with the forwarded signed version of $H(M)$. This validates both the authenticity and integrity of M simultaneously. If $H(M)$ were unsigned only integrity would be assured.

A hash function can also serve to detect modification of a message, independent of any connection with signatures. That is, it can serve as a cryptographic checksum (also known as a manipulation detection code [MDC] or message authentication code [MAC]). This may be useful in a case where secrecy and authentication are unimportant but accuracy is paramount. For example, if a key is sent in unencrypted form, even if the key is only a few hundred bits it might be useful to transmit a checksum along with it. Another instance where this case arises is when secrecy is provided by a system such as DES that does not provide a signature capability. An important distinction is that a hash function such as a MAC used in connection with a conventional system is typically parameterized by a secret shared key, although the latter may be distinct from the session key used in transmitting the message and its MAC. In contrast, hash functions used in connection with public key systems should be public and hence keyless.

We referred above to the use of hash functions as a "heuristic" solution to the problem of signing long messages. This refers to the fact that the signature for a message should be unique, but it is theoretically possible that two distinct messages could be compressed into the same message digest (a collision). The security of hash functions thus requires collision avoidance. Collisions cannot be avoided entirely, since in general the number of possible messages exceeds the number of possible outputs of the hash function. However, the probability of collisions must be low. If a function has a reasonably random output, the probability of collisions is determined by the output size.

A hash function must meet at least the following minimal requirement to serve the authentication process properly: It must not be computationally feasible to find a message that hashes to the same digest as a given message. Thus, altering a message will change the message digest. This is important to avoid forgery.

In many settings this minimal requirement is regarded as too weak. Instead, the requirement is added that it should not be possible to find any two strings that hash to the same value. We return to the distinction between these two criteria in Section 3.2.3.

3.2.1 Use of hash functions. In a public key system augmented by a hash function H, A might send a message M to B as follows: For simplicity ignore secrecy considerations. Then A sends M and $X = D_A(H(M))$ to B. The latter uses E_A to retrieve $H(M)$ from X, then computes $H(M)$ directly and compares the two values for authentication. For nonrepudiation, B retains M, $H(M)$, and X. If A attempts to repudiate M, a judge can use the three items to resolve the dispute as before: He computes $H(M) = E_A(X)$ and recomputes $H(M)$ from M. If B could find M' with $H(M') = H(M)$, B could claim A sent M'. A judge receiving M', $H(M)$ and X would reach a false conclusion.

3.2.2 Relation to one-way functions. Merkle [MERK82] defines a hash function F to be a transformation with the following properties:

1. F can be applied to an argument of any size.

2. F produces a fixed-size output.
3. $F(x)$ is relatively easy to compute for any given x.
4. For any given y it is computationally infeasible to find x with $F(x) = y$.
5. For any fixed x it is computationally infeasible to find $x' \neq x$ with $F(x') = F(x)$.

The most important properties are (4) and (5). In particular, (5) guarantees that an alternative message hashing to the same value as a given message cannot be found. This prevents forgery and also permits F to function as a cryptographic checksum for integrity. Actually (5) can be strengthened, as noted below.

Property (4) states that F is a one-way function (e.g., [DIFF76b]). One-way functions are used in various other contexts as well. For example, we noted above that the security of public key systems depends on the fact that the public transformations are trapdoor one-way functions. Trapdoors permit decoding by recipients. In contrast, hash functions are one-way functions that do not have trapdoors.

The concept of (nontrapdoor) one-way functions originated in connection with log-in procedures ([WILK68], p. 91): Needham [NEED78] noted that if passwords were stored encrypted under a one-way function, a list of encrypted passwords would be of no use to an intruder since the original passwords could not be recovered. When a user logged in, the string entered would be encrypted and compared to the stored version for authenticity. Essentially the same system was rediscovered later in [EVAN74].

Use of one-way functions to produce message digests or encrypt passwords is very different from use of trapdoor one-way functions to generate encryption functions $\{E_A\}$ in a public key cryptosystem. In the latter, (4) above becomes a dichotomy: it is computationally infeasible for anyone except A to find M from $C = E_A(M)$, but it is easy for A, the unique holder of the trapdoor D_A, to compute $M = D_A(C)$.

An example of functions that are not one-way are the private transformations in public key systems: anyone can solve $S = D(M)$ for M; that is, $M = E(S)$. This shows that signature schemes are not one-way functions; that is, they do not satisfy (4). On the other hand, a deterministic signature function S satisfies (5); that is, messages have unique signatures. For example, if a signature is generated via $S = D(M)$ where D is a decryption function of a public key system, then $D(M) = D(M')$ implies $E(D(M)) = M = E(D(M')) = M'$, so (5) is trivially satisfied.

Merkle's requirements are that a hash function must be both one-way (4) and effectively collisionless (5). To satisfy (1) and (2) concurrently, collisions must exist; (5) requires that a cryptanalyst should not be able to find them. This notion is amenable to further refinement.

3.2.3 Weak and strong hash functions. The security of hash functions can be refined further: We may distinguish between weak and strong hash functions. A function satisfying items (1)–(5) in Section 3.2.2, may be termed a *weak hash function*. Property (5) characterizes weak hash functions; it states that a cryptanalyst cannot find a second message producing the same message digest as a fixed message. On the other hand, H may be termed a *strong hash function* if items (1)–(4) still hold, but (5) is modified as follows: It is computationally infeasible to find any $\{x_1, x_2\}$ with $H(x_1) = H(x_2)$.

Strong and weak functions may often be obtained from the same class of functions. Strong functions are then characterized by larger message digests, which reduce

the probability of collisions. Strong functions are thus more secure, but the longer message digests are likely to increase time needed to hash.

Although the two definitions above are superficially similar, computationally they are quite different. For example, suppose H is a hash function with an 80-bit output. Suppose a cryptanalyst starts with a fixed message M and wishes to find a second message M' with $H(M) = H(M')$. Assuming that the 2^{80} outputs of H are totally random, any candidate M' has a probability of only 2^{-80} of meeting the required condition. More generally, if the cryptanalyst tries k candidates, the probability that at least one candidate satisfies $H(M) = H(M')$ is $1 - (1 - 2^{-80})^k$ which is about $2^{-80}k$ by the binomial theorem if the latter is small. For example, the cryptanalyst will probably have to compute H for about $k = 2^{74} = 10^{22}$ values of M' to have even one chance in a million of finding one M' that collides with M. Thus H is secure in the weak sense.

Suppose for the same H the cryptanalyst merely seeks any values $\{M_1, M_2\}$ with $H(M_1) = H(M_2)$. By Example F.1 (see Appendix F), if he examines $H(M)$ for at least $1.17 * 2^{40} < 2 * 10^{12}$ random values of M, the probability of at least one collision exceeds $\frac{1}{2}$. A supercomputer could probably find M_1 and M_2 with $H(M_1) = H(M_2)$ in at most a few days. Thus H is not secure in the strong sense.

The latter attack is called a birthday attack (Appendix F). It should be noted that finding a collision $H(x) = H(y)$ gives the cryptanalyst no valuable information if x and y are random bit strings. For a purpose such as forgery an adversary may need to generate a large number (e.g., 10^{12} above) of variations of a message to find two that collide. Inclusion of timestamps or sequence numbers in messages, according to a fixed format, may make it computationally infeasible to find a collision of use to an adversary.

3.3 Digital Signatures and Certificate-Based Systems

We noted above that digital signatures can be employed for authentication in the process of distributing public components in public key systems. In particular, if the system is certificate-based, the central issuing authority can sign certificates containing public components.

The notion of a central authority can be extended to a hierarchical (tree) structure. The central authority serves as the root of the tree; leaves represent users. Intermediate nodes may also be users. Typically the intermediate nodes will be arranged in several levels representing, for example, organizations and organizational units. Each node of the tree is responsible for authenticating its children. Thus an authentication path is created for each node, ending at the root. For example, the central authority may certify an organization; the organization may certify a unit; the unit may certify an individual user. Certification may be accomplished by having a parent node sign a certificate for the child node. To validate another user's certificate, a user may request the entire authentication path.

It is also possible for the tree to be replaced by a forest. For example, in a multinational system there may be a different tree for each country. In this event the roots must certify each other. An authentication path may then pass through two or more roots.

More generally, an authentication structure can be an arbitrary directed graph, with a directed edge from A to B if A certifies B. Then authentication paths may be constructed by conjoining directed paths from two users to a common trusted node; an example of this more complex structure is given in Section 5.3.

4 EXAMPLES OF PUBLIC KEY SYSTEMS AND HASH FUNCTIONS

Numerous public key systems have been proposed. These may be divided into several categories:

1. Systems that have been broken
2. Systems that are considered secure
   a. systems that are of questionable practicality
   b. systems that are practical
      (1) systems suitable for key distribution only
      (2) systems suitable for digital signatures only
      (3) systems suitable for key distribution and digital signatures

From the standpoint of sheer numbers, most systems have proven to be insecure. This is unfortunate, because some of the techniques producing insecure systems have relatively high bandwidths, where bandwidth refers to rates of encryption and decryption. Knapsack ciphers are an example (Section 4.2.1; Appendix E) of high-bandwidth but generally insecure systems. Of the systems that have not been broken, many are regarded as impractical for reasons such as large key sizes or large data expansion (ciphertext much larger than plaintext).

Only a relative handful of systems are widely regarded as secure and practical. In particular, a cryptosystem is usually regarded as secure if breaking it is essentially equivalent to solving a long-standing mathematical problem that has defied all attempts at solution.

Of the well-known systems that are generally regarded as secure and practical, some are limited to digital signatures and hence unsuitable for public key distribution (e.g., Section 4.2.2). On the other hand, in instances such as the Diffie–Hellman exponential exchange scheme (Section 2.2), public key distribution is supported but authentication is not. Such systems may need augmentation by a system that supports signatures. The only well-known system discovered to date that is secure, practical, and suitable for both secrecy and authentication is described in Section 4.1.

Category (2.b.3) in the list above could in theory be further subdivided into systems with relatively low and high bandwidths, but there are no well-known examples with high bandwidths. There does not exist a secure, practical system suitable for key distribution and digital signatures, and with bandwidth high enough to support bulk data encryption. Prospects for creating radically new systems seem dim; in fact, as noted in [DIFF88], most of the extant systems are based on a small number of hard mathematical problems:

- Discrete exponentiation
- Knapsack problems
- Factoring

According to Diffie, discrete exponentiation was suggested by J. Gill, and the use of the knapsack problem by Diffie. A suggestion to use factoring was apparently made by Knuth to Diffie during the early years of public key cryptography, but factoring was not incorporated into a cryptosystem until several years later (Section 4.1).

The knapsack problem is noted briefly in Section 4.2.1. The other two mathematical problems above are easy to state (although also hard to solve):

1. If p is a given prime and g and M are given integers, find x such that $g^x \equiv M \pmod{p}$.
2. If N is a product of two secret primes:
   a. Factor N.
   b. Given integers M and C, find d such that $M^d \equiv C \pmod{N}$.
   c. Given integers e and C, find M such that $M^e \equiv C \pmod{N}$.
   d. Given an integer x, decide whether there exists an integer y such that $x \equiv y^2 \pmod{N}$.

Gill's suggestion was based on (1), that is, on the difficulty of computing discrete logarithms modulo a prime. This has generated systems (e.g., Section 2.2) suitable for key distribution, and others suitable for signatures (e.g., Section 4.2.2).

Exploitation of the difficulty of factoring has proven to be the most versatile approach to design of public key systems. Factoring is widely regarded as very difficult. If a modulus has unknown factorization, various problems in modular arithmetic become difficult. For example, discrete logarithm (2b) is difficult, as is taking roots (2c) and deciding quadratic residuosity (2d). These problems have generated the most widely known public key system (Section 4.1) and various others. We remark in passing that the probabilistic scheme mentioned in Sections 3 and 9.1 are based on (2d); more generally, quadratic residuosity forms the basis of many zero-knowledge schemes.

Diffie's knapsack suggestion is one of many attempts to utilize nondeterministic polynomial (NP)-complete problems for cryptosystems. Such attempts have met with very limited success. We return to this topic and further discussion of the above problems later. In the meantime we discuss a few of the most well-known public key systems along with some hash functions.

4.1 The RSA Public Key Scheme

Rivest, Shamir, and Adleman [RIVE78] obtained the best-known and most versatile public key cryptosystem. It supports both secrecy and authentication, and hence can provide complete and self-contained support for public key distribution and signatures.

A user chooses primes p and q and computes $n = p * q$ and $m = (p - 1)(q - 1)$. He then chooses e to be an integer in $[1, m - 1]$ with $\text{GCD}(e, m) = 1$. He finds the multiplicative inverse of e, modulo m; that is, he finds (see Appendix H) d in $[1, m - 1]$ with $e * d \equiv 1 \pmod{m}$. Now n and e are public; d, p, q, and m are secret. That is, for a user A, n_A and e_A constitute the public component, and d_A, p_A, and q_A the private component.

After a user has computed p, q, e, and d the private transformation D and public transformation E are defined by (see Appendix M)

$$E(M) = M^e \mod n$$
$$D(C) = C^d \mod n$$

In the equation above, M and C are in $[0, n - 1]$. As in Section 1.5, we have $D(E(M)) = M$ and $E(D(C)) = C$; that is, D and E are inverses. Since d is private, so is D; and since e and n are public, so is E. This constitutes a cryptosystem that can be used for both secrecy and authentication. That is, for secrecy, A sends $E_B(M)$ to B as

usual; for authentication, A sends $D_A(M)$ as usual. For both secrecy and authentication, suppose first that message digests are not employed. Assuming $n_A \leq n_B$, A computes

$$C = E_B(D_A(M))$$

and sends C to B. Then B recovers M as usual by

$$M = E_A(D_B(E_B(D_A(M))))$$

As noted in Section 3.1.1, for nonrepudiation B may retain $D_A(M)$.

This implementation only works because the range of D_A is a subset of the domain of E_B; that is, $[0, n_A - 1]$ is a subset of $[0, n_B - 1]$. In the event that $n_A \geq n_B$, Kohnfelder [KOHN78b] notes that A can instead transmit

$$C' = D_A(E_B(M))$$

Then B can recover M as

$$M = D_B(E_A(D_A(E_B(M))))$$

This works since the range of E_B is a subset of the domain of D_A. For adjudication of possible disputes, B must retain C' and M. Then to prove that A sent M, the judge can compute $E_A(C')$ and $E_B(M)$ and test for equality.

However, in [DAVI83] and [DENN83b] a disadvantage to this solution is noted: A signs $E_B(M)$ rather than M. In Section 3.1.1 the judge was able to apply E_A to the stored quantity $D_A(M)$ and M is obtained. In Kohnfelder's protocol both C' and M must be stored, doubling the storage requirement.

The use of message digests eliminates the preceding problem. Suppose H is a hash function. Then to send M to B, A can create a new message M' containing M and $D_A(H(M))$ and send $C = E_B(M')$ to B. This gives both secrecy and authentication; also, C may be retained for nonrepudiation. Moreover, M and $D_A(H(M))$ are reblocked in forming M', so that no problem arises from the sizes of n_A and n_B.

4.1.1 Choice of p and q. To use the RSA system, a user must first choose primes p and q. As noted in Section 4.1.3, p and q should be at least 75–80 decimal digits; for the sake of discussion we assume p and q to be on the order of about 100 digits. The user begins by choosing a random odd b of around 100 digits; b is a candidate for p. Now b is acceptable only if b is prime. There are two approaches to this problem. The deterministic approach decides with certainty whether b is prime (see below). An alternative is to use the probabilistic approach as implemented, for example, by Solovay and Strassen [SOLO77]: Randomly choose a set of about 100 numbers in $[1, b - 1]$. For each number a in the set, test to see if $\text{GCD}(a, b) = 1$, where GCD = greatest common divisor. If this condition holds, compute the Jacobi symbol (a/b) (Appendix J). Both GCDs and Jacobi symbols are easy to compute via well-known algorithms (Appendices H and J). Now check to see if

$$(a/b) \equiv a^{(b-1)/2} \pmod{b}$$

If b is not prime, this check will fail with probability at least $\frac{1}{2}$ (Appendix L). Thus, if 100 random a's are tested and all pass the test above, the probability of b not being prime is about 1 in 2^{100}. Hence it is possible to test in polynomial time whether b is probably a prime. The probability of b being prime is about 1 in 115; hence

repeated applications of the preceding algorithm will probably quickly yield b which is probably a prime. This can be taken to be p; a second application of the algorithm will yield q.

The Solovay–Strassen algorithm is discussed further in Appendix L, which also mentions alternative algorithms of Lehman [LEHM82] and Miller and Rabin ([MILL76], [RABI80]).

The advantage of such probabilistic algorithms is that the algorithms run in polynomial time. They do not guarantee a correct answer, but the possibility of error is negligible as noted above. If absolute certainty were required, deterministic algorithms for primality testing could be used ([ADLE83], [COHE87], [COHE84]). These guarantee primality, but have runtimes of the form

$$\exp(c(\log \log n)(\log \log \log n))$$

If a sufficient amount of computing power is available, primality of 100-digit numbers can usually be established deterministically in minutes. However, the code needed is complicated; moreover, most users do not have access to high-performance computers. The probabilistic algorithms have acceptable runtimes even on small computers, particularly if hardware support is available.

4.1.2 Further notes on implementation. Once p and q have been chosen, e is chosen to be relatively prime to $m = (p - 1)(q - 1)$; that is, $\text{GCD}(e, m) = 1$. For example, e can be chosen to be a random small prime > 2. If e is relatively small, $E(M)$ can be computed relatively rapidly; that is, only a small number of modular multiplications are needed. The condition $\text{GCD}(e, m) = 1$ presents no problem; in fact the probability that two random numbers are relatively prime is about $\frac{3}{5}$ ([KNUT81], p. 324). Now d can be found in polynomial time; however, d may be large. Hence a version of the Chinese remainder theorem is employed for decryption (Appendix M).

There are some restrictions on p and q in addition to the size specification mentioned above. Some of these are noted in Section 4.1.3.1. Also, the time and space requirements of RSA are considerably larger than, for example, DES. Storage for each of p, q, and d requires at least 300 bits; n is about 600 bits. Only e can be chosen to be small.

Exponentiation mod n is a relatively slow operation for large n, especially in software. Efficient algorithms for computing products mod n have been given, for example, in [BLAK83], [BRIC82], [MONT85], and [QUIS82]. In particular, [BRIC82] shows how multiplication mod n can be done in m + 7 clock pulses if n is m bits. In [MONT85] it is shown how modular multiplication can avoid division.

At the present time RSA decryption (encryption is slower) with a 508-bit modulus has been performed at about 225K bits/sec in hardware [SHAN90]. Decryption with a 512-bit modulus has been effected at about 11K bits/sec in software [DUSS90].

4.1.3 Security of RSA. The security of the private components in the RSA cryptosystem depends on the difficulty of factoring large integers. No efficient algorithms are known for this problem. Consequently, knowing $n = p * q$ does not yield p and q. Thus it is computationally infeasible to find $(p - 1)(q - 1) = m$, and hence the relation $e * d \equiv 1 \pmod{m}$ is useless for determining d from e.

The difficulty of computing discrete logarithms (see Section 1.5) deflects known-plaintext attacks. Suppose an intruder knows M, C and $M = D(C)$ for some plaintext/

ciphertext pair $\{M, C\}$. To find d he must solve $M = C^d \mod n$; that is, he must find a discrete logarithm.

Ciphertext-only attacks are equivalent to taking roots modulo a composite number with unknown factorization, that is, solving $C = M^e \mod n$ for M. This is probably about as difficult as discrete logarithms, even in the case of square roots (e.g., [ADLE86]). In fact (Lemma N.3.2; see [KNUT81], p. 389; [RABI79]), taking square roots modulo such n is, loosely speaking, as hard as factoring n.

It should be noted that the security of RSA is not provably equivalent to the difficulty of factoring or the other problems mentioned here. Also, security of RSA or other public key systems does not preclude breaking specific protocols for their use; see, for example, [MOOR88] or [DOLE81]. An example of a protocol failure is given in [MOOR88]: Suppose two users use a common modulus n and encryption exponents e and e' with $GCD(e, e') = 1$. If the same message M is sent to both, then let $C = M^e \mod n$ and $C' = M^{e'} \mod n$. If both C and C' are intercepted, an intruder can find (Appendix H) r and s with $r * e + s * e' = 1$; now $M = C^r * C'^s \mod n$.

4.1.3.1 Restrictions on p and q. There are two major restrictions on p and q in order to foil attempts at factoring $n = p * q$:

1. p and q should be about the same length.
2. p and q should be at least 75 digits in length.

Present factoring methods will be foiled by such a choice. For longer-term security, p and q should be, for example, around 100 digits. Condition (1) is needed against the elliptic curve method (see below).

An attempt to predict the period of time for which a given modulus length will remain secure is necessarily guesswork. Intelligent guesses are the best that can be hoped for by implementors; these are connected with the status of progress in factoring, which is highly dynamic.

4.1.3.2 Notes on factoring. Choosing n to be about 200 digits will foil any present attempt at factoring n. The most powerful general-purpose factoring algorithm is Pomerance's quadratic sieve [POME84] and its variations. Using a parallel version of the quadratic sieve, Caron and Silverman [CARO87] factored 90-digit integers in several weeks on a network of several dozen Sun-3's. Recently, 106-digit integers have been factored on a larger network [LENS89], using the multiple polynomial variant of the quadratic sieve, and 111- and 116-digit composite integers have been factored by Lenstra and Manasse [BRILL88, update 91] using the quadratic sieve and the two-large-prime method.

Pomerance, Smith, and Tuler [POME88] discuss a prototype of a special pipelined architecture, optimized for the quadratic sieve, which they claim is extensible to a machine that might factor 125-digit integers in a year. However, this presents no threat to 200-digit moduli, since all of the best general-purpose factoring algorithms known, including the quadratic sieve, run in time roughly

$$\exp(((\log n)(\log \log n))^{1/2})$$

The fastest present computers only execute on the order of 10^{10} operations per second, at a cost of $10 million or more. Even if a future computer reached 10^{12} operations per second (and such a machine would presumably have a price tag of $1 billion

or more) it would take on the order of 1000 years to factor a 200-digit number using existing algorithms.

The strongest results on factoring obtained to date have utilized networks of computers. The power of such networks is more difficult to extrapolate than for supercomputers, since they are expandable and the power of personal computers and workstations has been rising at a higher rate than at the supercomputer level. In theory it would be preferable to know that a cryptosystem would be immune to an attack by an assemblage of every computer in the universe. It seems difficult to estimate the total amount of computing power this would bring to bear; furthermore, the question in practice is what fraction of this total amount can be realistically allotted to a given factoring problem (Appendix B).

It follows that implementors may have a high level of confidence in 200-digit moduli at the present. An interesting issue at the time of this writing is whether a modulus with a length between 150 and 200 digits will provide security for a given length of time. It seems certain that 154-digit (i.e., 512-bit) integers will be factored eventually, but the question is when. If the preceding runtime estimate is used, the conclusion is that 154 digits are likely to be secure until the late 1990s, barring major algorithmic advances (see Appendix B). RSA moduli of around 512 bits are generally preferable to 200 digits (around 660 bits) from a hardware-implementation standpoint.

A potential qualifier to the preceding analysis is that the runtime estimate above is generic; in particular it assumes that runtime is essentially independent of the integer being factored. There are, however, algorithms that have the runtime estimate above as their worst-case time [DIXO81]. For example, the Schnorr–Lenstra algorithm [SCHN84] factors n by using quadratic forms with discriminant $-n$. The equivalence classes of such forms form a group (the class group) whose order is denoted by $h(-n)$. The algorithm factors n quickly if $h(-n)$ is free of large prime divisors. The algorithm is Monte Carlo in the sense that the n's that will be factored quickly are evidently fairly random. No algorithms are known to generate class numbers with large prime divisors, although interestingly RSA comes close: Class numbers $h(-n)$ of discriminants n with one or two prime factors have a higher probability of having large prime factors than the class numbers of discriminants with only small prime factors.

The net result is that, for example, perhaps 1 in 1000 n's will factor 1000 times more quickly than average. If this method were practicable it would argue for the 200-digit modulus in RSA. However, the Monte Carlo phenomenon has not produced noticeably lower factoring times in practice as yet.

Some other factoring methods are applicable to numbers that do not have the form required for an RSA modulus. For example, the elliptic curve method [LENS87] is oriented to integers whose second-largest prime factor is considerably smaller than the largest prime factor. This motivates the condition that an RSA modulus should have factors roughly equal in size. A recent algorithm, the number field sieve [LENS90], applies to integers of the form $r^d + s$ or $r^d - s$ where r and s are small. Again this will not affect properly chosen RSA moduli.

4.1.4 Low-exponent versions of RSA. A priori the encryption exponent e in the RSA scheme is arbitrary. However, it need not be random. It has been suggested that e be chosen to have a common value by all users of an RSA-based system. Using $e = 2$ is not feasible per se, since 2 is not relatively prime to $p - 1$ or $q - 1$. However, extensions of this type have been made. These are of some theoretical interest, since Rabin [RABI79] and Williams [WILL80] have noted modifications of RSA with expo-

nent 2 for which successful ciphertext-only attacks are essentially as difficult as factoring the modulus (Appendix N).

The exponent 3 can be used directly with RSA. It has the advantage of greatly simplifying the encryption (but not the decryption) process. In fact Knuth [KNUT81], Rivest [RIVE84], and others recommend its use. However, 3 and other very low exponents suffer from some serious flaws. The case $e = 3$ is an illustration. For one thing, as Knuth notes, users must make certain that each block M to be encrypted satisfies $M^3 \gg n$. This is because $C = M^3 \bmod n$ becomes simply $C = M^3$ if $M^3 < n$; in this event finding M reduces to the trivial problem of taking ordinary cube roots of integers. Thus the use of $e = 3$ causes the domain of E to be a subset of $[0, n)$ rather than the whole interval as would be preferable.

A related but more subtle flaw has also been noted if, for example, $e = 3$ [HAST88]. Suppose A sends the same message M to each of $\{B_i\}$, $i = 1, 2, 3$. Suppose B_i uses modulus n_i, and $M < n_i$ for each i (this will be true, for example, if the sender uses a modulus smaller than each of the $\{n_i\}$). Assuming that each of $\{n_1, n_2, n_3\}$ is generated as a product of two random primes, the probability of duplicates among the six primes used is near zero. Hence it may be safely assumed that the $\{n_i\}$ are pairwise relatively prime. Let $C_i = M^3 \bmod n_i$. Suppose all three $\{C_i\}$ are intercepted. Using the Chinese remainder theorem (Appendix I), the intruder can find x in $[0, n')$, $n' = n_1 * n_2 * n_3$, with $x \equiv C_i \pmod{n_i}$, $i = 1, 2, 3$. Thus $x \equiv M^3 \pmod{n'}$. But $M^3 < n'$, so $M^3 = x$, so $M = x^{1/3}$ (ordinary cube root). Hence the plaintext M can be easily recovered from the three ciphertexts. Thus the use of $e = 3$ or other small exponents makes RSA vulnerable to ciphertext-only attacks. The sender may attempt to modify M for each recipient to deflect this attack, but Hastad shows that more generally, a low exponent is vulnerable to linear dependencies in portions of messages.

4.2 Other Public Key Systems

Several public key systems other than RSA have been proposed [MERK78b, MCEL78, ELGA85]. These are briefly surveyed here. As noted earlier, most of these are either insecure, of questionable practicality, or limited in scope. In some cases their security and/or practicality has not been established. None of these systems rivals RSA if a combination of versatility, security, and practicality is the criterion. However, this does not preclude their use for specific applications such as digital signatures.

4.2.1 Knapsack systems. These systems were proposed by Merkle and Hellman [MERK78b]. The essence, however, is the use of NP-complete problems (e.g., [GARE79], [HORO78]) that had been suggested for use in designing public key systems in [DIFF76b]. The Merkle–Hellman approach was to employ a superincreasing sequence $\{a_i\}$, that is, a sequence of positive integers for which

$$a_i > a_{i-1} + \ldots + a_1$$

The associated knapsack problem [HORO78] is to find nonnegative integers $\{M_i\}$ such that for a given Y,

$$Y = a_1 * M_1 + \ldots + a_n * M_n$$

The above is an instance of integer programming. In the present context the $\{M_i\}$ are binary; thus the above reduces to sum-of-subsets. Solving the above is easy for

superincreasing $\{a_i\}$; in fact the solution is unique. However, even deciding existence of a solution of sum-of-subsets, or more generally integer programming, is NP-complete ([GARE79], pp. 223, 245). Now for any w and u satisfying

$$u > a_1 + \ldots + a_n$$
$$\text{GCD}(u, w) = 1$$

a disguised knapsack problem may be produced from $\{a_i\}$ by defining

$$b_i = w * a_i \mod u$$

Then the associated knapsack problem

$$C = b_1 * M_1 + \ldots + b_n * M_n$$

appears to a cryptanalyst to be hard since the $\{b_i\}$ appear random. In actuality, it is easily solved using the connection with the $\{a_i\}$: since $\text{GCD}(w, u) = 1$, there exists W (Appendix H) such that

$$w * W \equiv 1 \pmod{u}$$

Then

$$W * C \equiv a_1 * M_1 + \ldots + a_n * M_n \pmod{u}$$

Since $0 \leq M_i \leq 1$,

$$0 \leq a_1 * M_1 + \ldots + a_n * M_n < u$$

from which

$$W * C = a_1 * M_1 + \ldots + a_n * M_n$$

As noted above, the latter knapsack problem has an easily found unique solution, from which C may be retrieved. This is called a trapdoor; it permits a legitimate user to easily solve what appears to be a hard problem. The modular disguise above can be iterated; that is, new w' and u' chosen and the $\{b_i\}$ disguised, etc.

The trapdoor knapsack yields a public key system: if the binary representation of plaintext M is $M_n \ldots M_1$, let

$$E(M) = b_1 * M_1 + \ldots + b_n * M_n$$

If $C = E(M)$ then decrypting C is equivalent to solving what appears to be a general knapsack problem, although decryption is easy using the trapdoor. The public component is $\{b_i\}$ and the private component is u, w, and $\{a_i\}$ (see [MERK78b] for details).

The advantage is that the arithmetic is much quicker than in RSA: about 200 additions are needed, as opposed to about 500 squarings and 150 multiplications in RSA. In fact knapsacks may rival DES in speed [HENR81]. Unfortunately the trapdoor knapsack above and most variations on the above have been broken (Appendix E).

It should also be noted that even if a knapsack approach proves to be secure and practical, such schemes are generally limited to support for either secrecy or authenti-

cation but not both. In contrast to RSA, the encryption and decryption functions are not inverses, although $D(E(M)) = M$. A trivial example suffices to show this: Suppose

$$E(M) = 2M_1 + 3M_2$$

To be invertible a function must be both injective and surjective. However, E is not surjective (onto). For example, there is no M such that $E(M) = 1$; hence $D(1)$ is undefined. Thus we could not employ D for signatures as in RSA, since, for example, 1 could not be signed by computing $D(1)$.

4.2.2 The El Gamal signature scheme. A signature scheme derived from a modification of exponentiation ciphers was proposed by El Gamal [ELGA85].

First, a prime p and a base a which is a primitive root modulo p (Appendix K) are public. Now suppose A wishes to send a signed message to B. The private component of A consists of two portions, one fixed and the other message-dependent. The fixed portion is a random x(A) from $[1, p - 2]$. The public component of A also has fixed and message-dependent components. The fixed portion is

$$y(\mathrm{A}) = a^{x(A)} \mod p$$

Now for a given message M in $[0, p - 1]$, A chooses a secret k in $[0, p - 1]$ with $\mathrm{GCD}(k, p - 1) = 1$. Thus k is the message-dependent portion of A's private component. Also, A finds (Appendix H) I with

$$k * I \equiv 1 \pmod{(p - 1)}$$

The message-dependent portion of A's public component consists of r and s, where

$$r = a^k \mod p$$
$$s \equiv I * (M - r * x(\mathrm{A})) \pmod{(p - 1)}$$

Now A sends M, r, and s to B. For authentication B computes

$$C = a^M \mod p$$
$$C' = y(\mathrm{A})^r * r^s \mod p$$

We note that

$$M \equiv r * x(\mathrm{A}) + k * s \pmod{(p - 1)}$$

Hence if M is authentic and valid,

$$C = (a^{x(a)})^{\mathrm{r}} * (a^k)^s \mod p = C'$$

A different k must be chosen for each M; using any k twice determines x(A) uniquely. The security of the method then depends mainly on the difficulty of computing discrete logarithms in GF(p): suppose an intruder intercepts M, r, and s; a and p are public. Let $d = a^r \mod p$ and $u * r^s \equiv 1 \pmod p$. Then the intruder knows

$$a^M \equiv y(\mathrm{A})^r * r^s \pmod p$$

Hence

$$u * a^M \equiv y(\mathrm{A})^r \pmod{p}$$

Thus

$$d^{x(\mathrm{A})} \equiv (a^{x(\mathrm{A})})^r \equiv y(\mathrm{A})^r \pmod{p}$$

Finally

$$d^{x(\mathrm{A})} \equiv u * a^M \pmod{p}$$

Finding $x(\mathrm{A})$, which permits the intruder to masquerade as A, is thus equivalent to the above discrete logarithm problem.

It is easy to solve this problem if $p - 1$ has only small factors [POHL78], hence $p - 1$ should have a large prime factor as in RSA. An alternative approach is to seek k and m satisfying

$$M = r * x(\mathrm{A}) + s * k + (p - 1) * m$$

but this is underdetermined, so that there are an exponential number of possible solutions to check. A third approach at cryptanalysis seeks r and s satisfying

$$a^M \equiv y(\mathrm{A})^r * r^s \pmod{p}$$

This is easier than discrete logarithm since r and s can both be varied, but it is not clear how much easier.

It is also not clear whether this scheme has a substantial advantage over RSA. Both employ exponentiation, so their speeds of encryption and decryption should be comparable. Generating keys in the two methods is similar; finding appropriate primes is the main step in either.

Cryptanalytically, the last attack (finding r and s simultaneously) may have a complexity substantially lower than factoring or discrete logarithm; hence security of El Gamal is at best no better than RSA, and possibly much inferior, with respect to secrecy of components. The use of message-dependent portions of components is a plus and minus: It increases bookkeeping, and makes it more difficult for a third party to adjudicate a dispute. On the other hand, it may produce greater security against certain types of attacks. In fact the k above could be chosen differently for different blocks of one message.

In passing we note that, like knapsacks, this scheme goes in one direction only: We have in effect

$$M = E(r, s) = (x(\mathrm{A}) * r + k * s) \mod (p - 1)$$

This maps the ordered pair (r, s) to a unique M. The condition $\mathrm{GCD}(k, p - 1) = 1$ guarantees that an (r, s) can always be found; that is, E is surjective. But it is not injective (one-to-one): for example, if $p = 7$, $k = 5$, $x(\mathrm{A}) = 5$, then $E(1, 2) = E(2, 1) = 3$. Thus text M cannot be represented uniquely by a pair (r, s), which would lead to ambiguity in an attempt to use this scheme in two directions.

Also, El Gamal notes that extension to $\mathrm{GF}(p^n)$ (Appendix G) is feasible. However, it is not clear that extensions to finite fields are desirable (see Section 1.5); gains in efficiency may be offset by lower security for comparable key sizes.

4.3 Examples of Hash Functions

To be useful in conjunction with public key systems, a hash function should ideally be keyless. This is in contradistinction to message authentication codes used in connection with secret key systems. Several hash functions have been proposed for use in public key settings. A few are mentioned here, but it seems to be difficult to produce secure, keyless, efficient hash functions. Thus we include some examples of functions that have keys, are insecure, or both.

4.3.1 Merkle's metamethod. Merkle ([MERK82], [MERK89]) has proposed a general technique for obtaining hash functions and digital signatures from existing conventional cryptosystems such as DES. Although secret key systems are used as adjuncts, the resulting hash functions are keyless; keys needed for DES are generated from the message to be hashed. The hash functions are precertified in the sense that their security can often be proven to be the same as that of the underlying conventional function.

Merkle assumes that encryption in a system such as DES defines a function E_K, where K is a key, that produces a random output. This is technically impossible, and in fact DES is not a random function since it is a permutation. Nonetheless, small deviations from randomness may be tolerable. Another requirement is that for a given key/plaintext (or ciphertext) pair, the ciphertext (or plaintext) returned will not vary with time. Normally a cryptosystem would meet this criterion.

Assume that a function $E_K(M)$ is available that satisfies these requirements. In deference to the potential use of DES to define E, assume K is 56 bits, M is 64 bits, and $E_K(M)$ is 64 bits. Since E may be assumed to be an encryption function, it may be assumed that there exists a decryption function with $E_K(D_K(C)) = C$. This is undesirable in producing one-way hash functions. This problem may be mitigated as follows: given a 120-bit input x, write $x = \mathrm{CAT}(K, M)$ where CAT denotes concatenation, K is the first 56 bits of x, and M is the last 64 bits. Let

$$f(x) = E_K(M) \text{ XOR } M$$

where XOR is exclusive-or. Then f hashes 120 bits to 64 bits. Furthermore, Merkle claims that f produces a random output even if x is chosen nonrandomly. A strong one-way hash function, as defined in Section 3.2.3, meets this criterion. Thus f might be termed a fixed-size strong one-way hash function, with the term "fixed-size" referring to the fact that f does not accept arbitrary inputs.

There are various ways in which f can be extended to a one-way hash function accepting arbitrary inputs (see [MERK89]). Furthermore, it is desirable to have a function that outputs more than 64 bits, to deflect birthday or square root attacks (Appendix F), which are based on the statistical phenomenon that the probability of collisions produced when hashing becomes significant when the number of items hashed is around the square root of the total number of hash function values.

In Section 3.2.3 we noted how such attacks affect the security of hash functions. For example, a 64-bit output (i.e., 2^{64} hash values) would be vulnerable to an attack using only $2^{32} = 4$ billion messages. On the other hand, a 112-bit output would require exhaustive search of on the order of 2^{56} values, producing a security level comparable to DES. Merkle discusses several constructions for producing an output exceeding 64 bits from f. Only the simplest construction will be discussed here. Given an input x of

119 bits, let

$$\mathrm{CAT}(f(\mathrm{CAT}(\text{``0''}, x)), f(\mathrm{CAT}(\text{``1''}, x))) = \mathrm{CAT}(y, z)$$

where y is 112 bits. Then define $F_0(x) = y$. In the equation above, `` '' denotes a constant bit string. We note that F_0 hashes 119 bits to 112 bits. This is not very efficient; however, the 112-bit output will deter a birthday attack. The latter would require $2^{56} = 7 * 10^{16}$ messages, which is computationally infeasible; more precisely, adequate computing power to effect a birthday attack would also break DES, thereby obliterating DES-based hash functions.

This produces F_0 which is also a fixed-size strong one-way hash function. Finally F_0 is extended to a one-way hash function F accepting inputs of arbitrary size by cascading copies of F_0. Given an input x, suppose

$$x = \mathrm{CAT}(x_1, \ldots, x_n)$$

where n is arbitrary and each x_i is 7 bits (pad x with a few zeros if need be). Let

$$\begin{aligned} R_0 &= 0 \\ R_i &= F_0(\mathrm{CAT}(R_{i-1}, x_i)) \qquad (1 \le i \le n) \end{aligned}$$

where in the first equation the right side is a string of 112 zeros. Let $F(x) = R_n$.

Now Merkle claims that F is as secure as F_0. That is, if $x \neq x'$ can be found with $F(x') = F(x)$, then F_0 can also be broken. The proof is inductive: if $n = 1$ we have $F(x) = F_0(\mathrm{CAT}(0, x))$. If $F(x) = F(x')$ and $x \neq x'$ let $z = \mathrm{CAT}(0, x)$ and $z' = \mathrm{CAT}(0, x')$; then $F_0(z) = F_0(z')$ and $z \neq z'$, so that F_0 is broken. Now suppose the claim is true for n. Suppose

$$\begin{aligned} z_i &= \mathrm{CAT}(x_1, \ldots, x_i) \qquad (1 \le i \le n+1) \\ x &= z_{n+1} \\ z_i' &= \mathrm{CAT}(x_1', \ldots, x_i') \qquad (1 \le i \le n+1) \\ x' &= z_{n+1}' \\ R_i &= F_0(\mathrm{CAT}(R_{i-1}, x_i)) \qquad (1 \le i \le n+1) \\ R_i' &= F_0(\mathrm{CAT}(R_{i-1}', x_i')) \qquad (1 \le i \le n+1) \end{aligned}$$

where each x_i and x_i' is 7 bits. Suppose $x \neq x'$ but $F(x) = F(x')$, that is, $R_{n+1} = R_{n+1}'$. Let $y = \mathrm{CAT}(R_n, x_{n+1})$ and $y' = \mathrm{CAT}(R_n', x_{n+1}')$. Then $F_0(y) = F_0(y')$. If $x_{n+1} \neq x_{n+1}'$ then $y \neq y'$ and F_0 is broken. Suppose $x_{n+1} = x_{n+1}'$. Then $R_n = R_n'$; but $z_n \neq z_n'$ since $x \neq x'$. Now we observe that by definition $R_i = F(z_i)$ and $R_i' = F(z_i')$. Hence $F(z_n) = F(z_n')$. By the induction hypothesis, F_0 can be broken, completing the proof.

A disadvantage of this F is that it hashes 7 bits per stage of the cascade, and each stage involves two DES calculations. Thus 3.5 bits are hashed per DES calculation. Merkle gives other constructions that are more complicated but hash up to 17 bits per application of DES.

4.3.2 Coppersmith's attack on Rabin-type functions. The Merkle meta-method is an attempt to use a conventional system as a generator for hash functions. This type of approach has its origin in some predecessors that have been broken by combinations of birthday attacks and meet-in-the-middle attacks as formulated by Coppersmith [COPP85].

An early attempt to construct a hash function in support of signature schemes was given by Rabin [RABI78]. Let H_0 be random (Rabin uses 0) and suppose a message M is divided into fixed-size blocks $M_1, \ldots, M_n$. Suppose $E(K,N)$ represents encryption of block N with key K using a conventional system such as DES. For $i = 1, \ldots, n$ let

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

Then a hash value is given by (H_0, H_n). Coppersmith's attack assumes that an opponent knows H_0 and H_n. He then constructs a bogus message whose hash value is also (H_0, H_n). Assume blocksize is 64 bits. The opponent begins by specifying $M_1, \ldots, M_{n-2}$ using H_0. He then generates 2^{32} trial values X. For each X he computes a trial H_{n-1} of the form

$$H(n-1, X) = E(X, H_{n-2})$$

He sorts these 2^{32} values and stores them. Now he generates 2^{32} trial values Y. For each Y he computes a trial H_{n-1} of the form

$$H'(n-1, Y) = D(Y, H_n)$$

where D is the decryption function corresponding to E. We note that $H(n-1, X)$ is the value that H_{n-1} would have if $M_{n-1} = X$, and $H'(n-1, Y)$ is the value H_{n-1} would have if $M_n = Y$. Each time the opponent tries a Y he searches through the sorted H-list (about 32 operations per search) and tries to find an X and Y with $H(n-1, X) = H'(n-1, Y)$. By the birthday phenomenon (Appendix F), the probability of finding X and Y is at least $\frac{1}{2}$. If X and Y are found, the opponent has obtained a bogus message with the proscribed hash value; furthermore, this takes at most about 2^{33} encryptions, 2^{32} storage, and $64 * 2^{32}$ comparison operations. This effectively breaks the scheme. Actually Coppersmith's attack was directed against a refinement by Davies and Price [DAVI80]; Rabin's original scheme had been broken earlier.

4.3.3 Quadratic congruential hash functions. Another attempt to create hash functions, differing from both Merkle- and Rabin-type constructions, was made by Jueneman [JUEN82]. Unlike the Merkle metamethod, this method uses no external system as an adjunct. It is not keyless; an initialization vector is used. This limits the scope of use; in particular, such a function is useless for signature-only schemes. Nonetheless, quadratic congruential constructions are simple and efficient, and hence would be useful in some contexts if secure. Unfortunately it seems as though Coppersmith's attack applies to these as well.

Jueneman uses the same partition into fixed-size blocks as above, and again begins by choosing $H_0 = 0$. He also chooses a secret seed M_0, however, which is changed every message and transmitted as a prefix to the message. Thus assuming 32-bit blocks (to use the Mersenne prime $2^{31} - 1$), for $i = 0, \ldots, n$ he computes

$$H_{i+1} = (H_i + M_i)^2 \mod(2^{31} - 1)$$

Then the hash value is H_n. Coppersmith broke this first scheme. A revised scheme was proposed in [JUEN86]. The text is split into 128-bit blocks, which are further divided into four words. Recent results, as described in detail in the chapter on Digital Signatures by Mitchell, Piper, and Wild in this volume, suggest that this scheme may be

insecure also. This is especially unfortunate in light of the fact that a hash function given in Annex D of X.509 [CCIT87] is defined similarly, via

$$H_{i+1} = M_i + H_i^2 \pmod{n}$$

4.4 Hardware and Software Support

As noted at the beginning of Section 4, RSA and similar exponentiation-based algorithms suffer from relatively low bandwidths. At a minimum this implies that supporting software needs to be carefully designed; hardware support is probably a necessity in many settings. As noted by Sedlak [SEDL87], bandwidths of less than 64K bits/sec will degrade performance in many networks. Furthermore, arithmetic modulo numbers of an adequate number of bits (at least 512) should be supported. Essentially the requirement is a chip (or chip set) that can quickly compute quantities such as a mod b, a^i mod b, and perhaps multiplicative inverses or greatest common divisors. As Sedlak further notes, a single chip is preferable for security reasons. Since off-chip communication is slower than on-chip, single-chip implementations should also yield higher bandwidths.

4.4.1 Design considerations for RSA chips. Some general trade-offs in such design schemes are discussed by Rivest [RIVE84]. He classifies architectures as sequential (S), serial/parallel (S/P), or parallel/parallel (P/P). He then notes the following time and hardware costs:

1. Multiplying two k-bit numbers modulo a k-bit number:
   a. $O(k^2)$ time and $O(1)$ gates on an S
   b. $O(k)$ time and $O(k)$ gates on an S/P
   c. $O(\log k)$ time and $O(k^2)$ gates on a P/P
2. Exponentiating a k-bit number modulo a k-bit number:
   a. $O(k^3)$ time and $O(1)$ gates on an S
   b. $O(k^2)$ time and $O(k)$ gates on an S/P
   c. $O(k \log k)$ time and $O(k^2)$ gates on a P/P
3. Key generation, k-bit primes:
   a. $O(k^4)$ time and $O(1)$ gates on an S
   b. $O(k^3)$ time and $O(k)$ gates on an S/P
   c. $O(k^2 \log k)$ time and $O(k^2)$ gates on a P/P

This is a somewhat oversimplified version presented only for comparison purposes. Also, there are two basic assumptions used above: Exponentiation is an inherently sequential process; and the 100 or so primality tests used in key generation are done sequentially. The first assumption seems intrinsic. However, the second is pragmatic: The tests all use the same hardware, and replicating the latter 100-fold for parallel execution would be highly cost-ineffective. Rivest uses the time and cost estimates above to give some sample timings:

1. Decryption rate, 200-digit modulus:
   a. 0.005K bits/sec on an S
   b. 1.3K bits/sec on an S/P
   c. 95K bits/sec on a P/P

2. Key generation, 100-digit primes:
   a. 1,200,000 msec = 20 minutes on an S
   b. 5000 msec = 5 seconds on an S/P
   c. 70 msec on a P/P

It may be seen that reasonably high bandwidths can be achieved, but the time/cost trade-off is significant.

4.4.2 Proposed designs for RSA chips. Various authors have proposed designs for chips supporting RSA. Of course, such chips could also support other exponential-based algorithms.

Orton et al. [ORTO86] discuss an implementation of RSA in 2-μm CMOS that should encrypt at 40K bits/sec for 512-bit moduli.

Sedlak [SEDL87] discusses a highly optimized chip which makes substantial use of lookahead. Thus the number of cycles required for exponentiation is not fixed; analysis of expected time is performed using a probabilistic finite-state machine (Appendix D). Support is provided not only for RSA but also for the ISO hash function (see above). Sedlak claims a deciphering rate of nearly 200K bits/sec is achievable for 780-bit moduli using a single 160,000-transistor chip with dual ciphering units, in 1.5 μm* CMOS. He also claims a key generation time of 2 seconds. A 5000-transistor, 5-μm CMOS prototype has been realized.

In [BRIC89] it is noted that chips capable of up to 450K bits/sec are being designed.

5 IMPLEMENTATIONS OF PUBLIC KEY CRYPTOGRAPHY

We examine here some existing implementations of public key cryptography, some implementations that are in progress, and some that have been proposed.

5.1 MITRENET

One of the earliest implementations of public key cryptography was in the MITRE encrypted mail office (MEMO) system, a secure electronic mail system for MITRENET ([SCHA82], [SCHA80]). MITRENET is a broadband cable system with a bandwidth of 1M bits/sec. It uses a carrier-sense protocol (e.g., [TANE81]); that is, each station can sense transmissions of all other stations. In fact the protocol requires all parties to monitor all communication, in effect requiring passive eavesdropping. A priori this provides no secrecy, authentication, or integrity services. Furthermore, it employs distributed switching, creating a potential for active intrusion. This is a good setting for a privacy enhancement testbed.

The MEMO system is a hybrid public/private cryptosystem. DES is used for data encryption. The Diffie–Hellman exponential key exchange of Section 2.2 is used for

---

* μm = micro meter, much more commonly called a micron. VLSI design rules are always stated in microns; 1 micron or 1 μm etc.

establishment of secret keys. To implement the Diffie–Hellman system, use of GF(2^n), with $2^n - 1$ a prime (called a Mersenne prime), was chosen for efficiency of implementation via linear feedback shift registers. Unfortunately the MEMO implementors used $n = 127$; the work of Adleman [ADLE79] rendered this choice insecure even before the system was implemented. This is noted by Schanning; in fact the use of $n = 521$ is recommended in [SCHA82], suggesting that the MEMO system was intended mainly for experimental purposes.

In the MEMO system, a public key distribution center (PKDC) is a separate network node containing public components in erasable programmable read-only memory (EPROM). Private components can be generated by users or by the system.

Each user workstation establishes secure communication with the PKDC. A session begins with the user requesting the file of public keys from the PKDC. The request is honored if it passes an identification test involving the user's private component. The file of public keys is then downloaded to the user's workstation, encrypted with DES to ensure integrity.

When the user sends a message to another user, the workstation generates a random document key. The latter is used to DES-encrypt the message. The public key of the recipient is used to generate a key-encrypting key which is used to DES-encrypt the document key.

There is no provision for lost keys. Some provision is made for detecting modifications to messages, using checksums. The use of Diffie–Hellman alone, however, does not permit authentication to be introduced.

5.2 Integrated Services Digital Network

In [DIFF87] a testbed secure Integrated Services Digital Network (ISDN) terminal developed at Bell-Northern Research is described. It can carry voice or data at 64K bits/sec. Public key cryptography is used for both key exchange and authentication. Reference to the Diffie–Hellman exponential key exchange is made. Evidently it is used in conjunction with DES. The article also alludes to signatures, but implementation is unclear.

As noted in [DIFF88], the use of public key cryptography in conjunction with DES provides good security. In particular, the exponential key exchange permits a session key unique to each call. Thus if long-term keys are compromised, recordings of calls made prior to compromise cannot be decoded. In conventional systems the compromise of a long-term key may compromise communications made previously with derived keys.

Public key computations in the terminal are implemented via a digital signal processing (DSP) chip, while an off-the-shelf integrated circuit implements DES, which is used for data encryption. Key management involves keys installed in phones, carried by users, and exchanged electronically.

Each Bell-Northern Research, Inc. (BNR) terminal incorporates a Northern Telecom M3000 Touchphone.

5.2.1 Keys. A public/private component pair is embedded in the phone; this pair is called the *intrinsic key.* The private component is contained in a tamper-resistant compartment of the phone. The public component serves as the name of the phone. These cannot be altered.

A second long-term public key stored in the phone is the *owner key.* This key is used to authenticate commands from the owner. It can be changed by a command signed with the current owner key, thus permitting transfer to a new owner.

A third long-term public key in the phone is the *network key.* This identifies the network with which the phone is associated. It validates commands signed with the private component of the network's key management facility (KMF). It can authenticate calls from network users, and it can be changed by a command signed by the owner key.

A short-term component pair stored in the phone is the working pair. This component is embodied in a certificate signed by the key management facility. During the setup process for a call, phones exchange certificates. The network key is used to authenticate certificates. This permits station-to-station calls.

Further information is needed for authenticated person-to-person calls. This information is contained on a hardware "ignition key" which must be inserted into the phone. This key contains the user's private component encrypted under a secret password known only to the user. It also contains a certificate signed by the KMF which contains the user's public component and identifying information. The latter is encrypted as well. Decryption of the information on the ignition key is effected via a password typed on the telephone touchpad.

Further certificates authorizing users to use particular phones are acquired by the phone from the key management facility; these are cached and replaced in first in, first out (FIFO) fashion.

5.2.2 Calling. A person-to-person call begins as follows: The caller inserts his ignition key and dials the number. The phone interrogates the ignition key to check the caller's identity. The phone then checks its cache for an authorization certificate for the caller. If not found it is acquired by the phone from the KMF. The phone then dials the number of the other phone.

The two phones then set up. This begins with an exponential key exchange. The calling phone then transmits its certificate and user authorization. The receiving phone authenticates the signatures on both of the latter using the network key. The receiving phone then employs challenges. It demands real-time responses to time-dependent messages; the responses must be signed. This ensures that certificates are not played back from a previous conversation. One response is signed by the calling phone; another with the user's ignition key. A further level of security may be obtained via the ISDN D channel, which makes calling party identification available from the network without interrupting the primary call.

Now the called phone sends its certificates and responds to challenges as above. If the called party is home he inserts his ignition key. If the called phone does not have authorization for the called party to use the phone, it must obtain a certificate. Again this can be accomplished via the D channel without interrupting the call. The called party then undergoes challenge and response. Finally the conversation ensues.

5.3 ISO Authentication Framework

Public key cryptography has been recommended for use in connection with the ISO authentication framework, X.509 [CCIT87]. This is based on the directory, a collection of services and databases that provide an authentication service. Authentication refers to

verification of the identity of a communicating party. Strong authentication uses cryptography. The credentials of a communicating party may be obtained from the directory.

No specific cryptosystem or hash function is endorsed in support of strong authentication; however, it is specified in [CCIT87] that a cryptosystem should be usable for both secrecy and authenticity. That is, the encryption and decryption transformations should be inverses, as is the case with RSA. Multiple cryptosystems and hash functions may be used in the system.

A user must possess a distinguished name. Naming authorities are responsible for assigning unique names. The crux of the authentication framework is the binding of user names and public components. Assuming for the moment that such binding has occurred, subsequent authentication in the ISO framework consists of locating a chain of trusted points within the directory. Such a chain exists if there is a common point of trust between two authenticating parties.

5.3.1 Use of certificates. The X.509 public component management system is certificate-based. Binding of a user's name and public component occurs when a certification authority (CA) issues a certificate to a user. The certificate contains the user's public component and distinguished name, and the certificate's validity period. It is generated offline and signed by the CA using the CA's private component. Normally a user would generate his own public/private component pair and transmit the public component to the CA for inclusion in the certificate. At the user's request the CA may also generate the user's public/private component pair.

The CA vouches for the binding of the user's name and public component. The CA must not issue multiple certificates with one name. Also, a CA must keep his private component secret; compromise would affect not only the CA but also the integrity of communications involving users certified by the CA.

Obtaining a certificate from a CA requires a secure communication between the user and CA. In particular, the integrity of the user's public component must be assured. This communication can be online, offline, or both for redundancy.

The user receives a copy of the certificate obtained from the CA. This can be cached; for example, a component pair could be stored on a smart card along with the user's certificate and the CA's certificate. Additionally, certificates are entered in the directory. The user may place a copy of the certificate in the directory, or the CA may be authorized to do this. A user's directory entry may contain multiple certificates. A CA's entry in the directory information tree (DIT) contains the certificates issued for it by other CAs, as well as certificates it issues for other nodes.

Semiformally a certificate may be defined as follows:

```
certificate : : =
  {
  signature algorithm identifier;
  issuer name;
  validity period;
  subject name;
  subject information
  }
validity period : : =
  {
```

```
    start date;
    finish date
    }
subject information :: =
    {
    subject public key;
    public key algorithm identifier
    }
```

This format permits use of different algorithms. For a more formal description of relevant formats using ASN.1 see Annexes G and H of [CCIT87].

5.3.2 Certification paths. An associated data structure is the directory information tree (DIT). Certification authorities are otherwise undistinguished users who are nodes in the DIT. A user may have certificates issued by several CAs. Thus the authentication structure, despite the use of the term DIT, is not tree-structured. Instead it may be modeled as a directed graph.

A certification path consists of certificates of nodes in the DIT. The public component of the first node is used to initiate a domino-type process that ultimately unlocks the whole path, leading to recovery of the public component of the final node. The simplest path is of the form (A, B), where A is a CA for B. Then a knowledge of the public component of A permits recovery of the public component of B from the certificate issued by A for B, since the latter is signed using the private transformation of A. This is readily extended by recursion: In a certification path $A_1, \ldots, A_n$, knowing the public component of A_i permits recovery of the public component of A_{i+1} from the certificate for A_{i+1} issued by A_i.

For a user A to obtain B's certificate involves finding a common trusted point, that is, a node that has certification paths to the two users individually. Joining these two paths produces a certification path between A and B. The paths, if they exist, may be obtained from the directory.

Although there is no restriction placed on the structure of the authentication graph, an important special case is when it is tree-structured. In this event the CAs are arranged hierarchically; hence each node (except the root) has a unique CA (its parent in the tree). Then each CA stores the certificate obtained from its superior CA, as well as various certificates issued by it. The common trusted point for a pair of users is the root of the DIT. User A may cache the certificates of nodes along the path from A to the root. The other half of the path to another user B is obtained by conjoining the path from the root to B.

More generally, a user A who communicates frequently with users certified by a particular CA could store paths to and from that CA (these may be different in the nonhierarchical case). Then to communicate with a given user B certified by that CA, A need only consult the directory entry for the CA and obtain the certificate of B.

A related concept is cross-certification: If two CAs C_1 and C_2 have certified users who communicate frequently, C_1 and C_2 may certify each other. Then the certificate issued by C_1 for C_2 can be stored in the directory entry of C_1 and vice versa. We note that in a hierarchical system, a priori a directory entry for a node contains only the certificate of a node's superior and the reverse certificate; if cross-certification is permitted then entries contain an indefinite number of certificates.

A user may cache certificates of other users. A CA may add another's certificate to its directory entry.

5.3.3 Expiration and revocation of certificates. When a certificate expires it should be removed from the directory. Expired certificates should be retained by the issuing CAs for a period of time in support of the nonrepudiation service.

Revocation of certificates may occur because of component compromise involving either the user or the issuing CA. Revocation may also be necessary for administrative reasons, for example, when a CA is no longer authorized to certify a user. A CA keeps a time-stamped list of revoked certificates which the CA had issued, and a list of revoked certificates issued by other CAs certified by the first CA. Entries for CAs in the directory should contain revocation lists for users and CAs.

5.3.4 Authentication protocols. Suppose A wishes to engage in communication with an authenticated B. Suppose further that A has obtained a certification path from A to B, for example, by accessing the directory, and has utilized the path to obtain B's public component. Then A may initiate one-, two-, or three-way authentication protocols.

One-way authentication involves a single communication from A to B. It establishes the identities of A and B and the integrity of any communicated information. It also deflects replay in communication of authentication information.

Two-way authentication adds a reply from B. It establishes that the reply was sent by B and that information in the reply had integrity and was not replayed. It also establishes secrecy of the two communications. Both one-way and two-way authentication use timestamps. Three-way authentication adds a further message from A to B, and obviates the necessity of timestamps.

Let

I_A = identity of A
I_B = identity of B
D_A = private transformation of A
E_A = public transformation of A
D_B = private transformation of B
E_B = public transformation of B
T_A = timestamp by A
T_B = timestamp by B
R_A = random number generated by A
R_B = random number generated by B
C_A = certification path from A to B

Identities refer to the distinguished names of A and B. A timestamp included in a message M includes an expiration date for M. Optionally, it also may include the time of generation of M. Random numbers may be supplemented with sequence numbers; they should not be repeated within the expiration period indicated by a timestamp in the same communication.

The one-way protocol is as follows:

1. A:
   a. generates an R_A.
   b. constructs message $M = (T_A, R_A, I_B, <\text{data}>)$ where <data> is arbitrary. The latter may include data encrypted under E_B for secrecy, for example, when A is sending a DEK to B.
   c. sends $(C_A, D_A(M))$ to B.
2. B:
   a. decrypts C_A and obtains E_A. Also checks certificate expiration dates.
   b. uses E_A to decrypt $D_A(M)$, verifying both A's signature and the integrity of the signed information.
   c. checks the I_B contained in M for accuracy.
   d. checks the T_A in M for currency.
   e. optionally, checks the R_A in M for replay.

The two-way protocol is:

1. as above.
2. as above.
3. B:
   a. generates an R_B.
   b. constructs $M' = (T_B, R_B, I_A, R_A, <\text{data}>)$ where R_A was obtained previously and <data> may include data encrypted using E_A.
   c. sends $D_B(M')$ to A.
4. A:
   a. decrypts $D_B(M')$, verifying B's signature and the integrity of the enclosed information.
   b. checks the I_A contained in M' for accuracy.
   c. checks the T_B in M' for currency.
   d. optionally checks R_B for replay.

The three-way protocol is:

1. A:
   a. generates an R_A.
   b. constructs $M = (T_A, R_A, I_B, <\text{data}>)$. Unlike the previous cases, T_A may be zero.
   c. sends $(C_A, D_A(M))$ to B.
2. B:
   a. decrypts C_A and obtains E_A. Also checks certificate expiration dates.
   b. uses E_A to decrypt $D_A(M)$, verifying both A's signature and the integrity of the signed information.
   c. checks the I_B contained in M for accuracy.
   d. optionally, checks the R_A in M for replay.
   e. generates an R_B.
   f. constructs $M' = (T_B, R_B, I_A, R_A, <\text{data}>)$ where R_A was obtained previously; T_B may be zero.
   g. sends $D_B(M')$ to A.

3. A:
   a. decrypts $D_B(M')$, verifying B's signature and the integrity of the enclosed information.
   b. checks the I_A contained in M' for accuracy.
   c. optionally checks R_B for replay.
   d. checks the received version of R_A against the version sent to B.
   e. sends $D_A(R_B)$ to B.
4. B:
   a. decrypts $D_A(R_B)$, verifying the signature and data integrity.
   b. checks the R_B received against the value sent to A.

Remarks: It has been noted that there are some potential problems with these protocols. For example, in the three-way version, it would be better to have A send $D_A(R_B, I_B)$ to B rather than just $D_A(R_B)$. This would prevent a third party from intercepting random numbers and using them to subvert the protocol.

Also, since in both of the multipass protocols authentication tokens may be of the form $M = (T_A, R_A, I_B, R_B, E_B(\text{encdata}))$. Then encrypted data are signed, which may cause problems. For example, suppose C intercepts $D_A(M)$ on its way to B. Then C may decrypt M and construct $(T_C, R_C, I_B, R_B, E_B(\text{encdata}))$. In a communication between B and C, B may think that $E_B(\text{encdata})$ was generated by C and inadvertently reveal encdata to C. In particular, encdata may have included a data-encrypting key to be used by A and B. A solution to this problem is to let $M = (T_A, R_A, I_B, R_B, \text{encdata})$, $M' = (T_A, R_A, I_B, R_B, E_B(\text{encdata}))$, and $M'' = (M', D_A(M))$, then send M''.

5.3.5 Further notes. Authentication uses a hash function. Signed information includes identifications of the hash function used to produce the message digest and the decryption function used to generate the digital signature. Timestamps and random numbers are included in messages to prevent replays and forgery.

Annex C of [CCIT87] mentions RSA as an example of a public key system. A recommendation of a 512-bit modulus is made. It is also recommended that a common public exponent of $e = 2^{16} + 1$ be used (the fourth Fermat number). In particular it is noted that a smaller e would be vulnerable to ciphertext-only attacks.

Other annexes specify a strong hash function as defined in Section 4, and the Abstract Syntax Notation 1 (ASN.1) syntax for the authentication framework. Algorithm identifiers are included in ASN.

5.4 Defense Advanced Research Projects Agency (DARPA)-Internet

The IAB Privacy Task Force is in the process of developing a proposal for privacy-enhanced electronic mail for the DARPA-Internet community ([IAB-88], [IAB-88b], [LINN89]). The summary presented here is of work in progress.

Public key cryptography has been recommended for distribution of secret keys and in support of authentication of communicating parties (i.e., for signature generation). It is anticipated that a conventional secret key cryptosystem will be used for encryption of messages and for message integrity check (MIC) computation. RSA has been recommended for key distribution; it may also be used for signatures, although other possibilities have been suggested. A universal hash function is not currently supported. Fields

are included in messages to identify cryptographic algorithms and hash functions. It is specified that all implementations should support RSA and DES; the role of other systems has not been finalized at the time of this writing.

Much of the authentication framework is borrowed from a subset of X.509. In particular, the recommended public component management system is certificate-based.

5.4.1 Services. Services to be provided in the DARPA-Internet system include:

- Data confidentiality
- Data origin authentication
- Message integrity assurance
- Nonrepudiation of origin

The framework for providing these services is X.400. End-to-end service is supported between user agent (UA) processes. To send mail, a user calls on a UA, which is an application program that interacts with the message transfer system (MTS). The MTS delivers the message to one or more UAs. No special requirements are made on the MTS.

Privacy enhancement occurs at the application layer only, with integration at the UA level or above. End-to-end encryption is employed. Encapsulation of privacy-enhanced messages within an enclosing layer of headers interpreted by MTS is supported. Lack of involvement of lower layers implies a corresponding limitation of services. For example, access control and routing control are not addressed; nor are security services aimed at attacks such as traffic analysis. Also, no provision is made for nonrepudiation of receipt.

Message text may or may not be encrypted, at the sender's option; encryption of only subsets of message text is also supported. Authentication always uses the whole message.

5.4.2 Key management. DEKs are shared secret keys that are used for text encryption. They are used in connection with secret key data encryption systems such as DES. They are also employed when secret key cryptography is used to compute MICs. New DEKs are generated for each message. They can be generated by a KDC or at endpoints. No form of predistribution is required for these keys.

Interchange keys (IKs) are used to encrypt DEKs and MICs. The same IKs can be used by two users for a period of time. Assuming that public key cryptography is used for key distribution, the recipient's public component is used as an IK to encrypt DEKs; the sender's private component is used as an IK to sign MICs. If secret key cryptography is employed for key distribution, an IK functions as a master key.

A message contains X-Sender-ID and X-Recipient-ID control fields which are used to identify the IK used to encrypt the DEKs or MICs for a given recipient. If public key cryptography is employed, and a message is intended for several recipients, multiple X-Recipient-ID fields are required to hold the IKs (recipient public keys) used for encrypting DEKs. Again in the case of public key cryptography, a single IK (the originator's private component) may be employed directly in MICs.

5.4.3 Encapsulation and encoding. A semiformal description of message format is as follows:

```
message : : =
  {
  enclosing header;
  encapsulated message
  }
encapsulated message : : =
  {
  encapsulation boundary;
  encapsulated header;
  blank line;
  encapsulated text;
  encapsulation boundary
  }
```

The enclosing header contains RFC-822 header fields. It also contains user-supplied data, for example, "subject." These fields pertain only to message transport, and not privacy or authentication.

The encapsulated header contains encryption control fields such as DEKs and IKs. It also contains user address information needed for encryption. These fields, however, pertain only to privacy and/or authentication, and not to message transport. The division of fields between the enclosing and encapsulated headers is related to the end-to-end encryption employed; intermediate nodes and gateways need to access unencrypted routing information.

Encapsulated text contains encrypted user message text. Optionally, it may contain encrypted copies of protected fields of the enclosing header, for purposes of redundancy in case the enclosing header is accidentally or deliberately modified. Separation of the two headers ensures that modification of header fields used for transport will not affect privacy.

A message is accepted in local form, using the host's character set and line representation. The local form is then converted to canonical form. The latter is padded per the encryption mode, and a MIC is computed. Assuming that data confidentiality is desired, the padded canonical representation is encrypted. The encrypted form is encoded into printable form, universal across all sites. The printable form is combined with header fields. Finally the result is passed to the mail system for encapsulation as the text portion.

5.4.4 Use of certificates. Certificate-based key management is supported. This is compatible with a subset of X.509 [CCIT87]. Certificates can be distributed through insecure channels. A distributed directory service has been suggested, but implementation prospects are unclear at the moment. A sender must obtain a certificate for each recipient of a privacy-enhanced message.

Semiformally, a certificate has the following format:

```
certificate : : =
  {
  serial number;
  issuer name;
  user name;
```

```
validity period;
user public component / algorithm identifier;
issuer signature / algorithm identifier
}
```

The issuer's signature is computed on a hash value based on certificate contents; the certificate also contains an identifier for the hash algorithm used. Components of the subject (user) name include country, administration domain (DARPA-Internet), personal name, and a domain-defined attribute to identify the subject's mailbox address. Organization and organizational unit affiliations are optional. Issuer names follow a similar format. Validity period is at most 2 years.

5.4.5 Authentication framework. The authentication structure here deviates somewhat from X.509 in that minimal assumptions are made regarding a directory system infrastructure. Thus, for example, it could not be assumed that certification paths through an arbitrary directed graph could be located by users. This dictates the use of a hierarchical (tree-structured) certification framework, with users at the leaves. Each node (except the root) then possesses a unique parent node and a unique path leading to the root. The root thus functions as a common point of trust for all users.

Each user has exactly one certificate. The certificate for node A is issued by the parent node of A, or other nodes lying between A and the root of such a hierarchical system. The latter typically has three or four levels, so that certification paths are correspondingly short. An issuing node is called a certification authority (CA). The latter generates a certificate from a subject's public component and identification and signs a message digest computed on the certificate. Thus a CA vouches for the binding between user ID information and public component. The CA for a user may be the root itself.

A simple hierarchy has users at the leaves, with only one intermediate level between leaves and root, consisting of organizations. In this case a CA may represent an organization; then users affiliated with that organization can obtain certificates through it. The binding between the user's ID and public component implicitly affirms the user's affiliation with the organization.

It is also possible to add a layer to the hierarchy by having organizational units or organizational notaries (ONs) within organizations. Further refinement is discouraged to shorten certification paths. Whether or not or organizational units are present, the root certifies organizations. Organizations may certify users, or they may certify organizational divisions or notaries which in turn certify users. Organizational units and notaries may possess a second certificate issued by the root.

This tree-structured framework may be extended to a forest by permitting multiple roots. For example, the U.S. government might have its own certifying facility. Other roots may exist in other countries as well. If multiple roots are present they should cross-certify.

Public components of CAs may be obtained recursively from certification paths as usual; that is, each certificate in the path is used to decrypt the next, until finally the user's public component is recovered.

5.4.6 Obtaining certificates. The process by which a user obtains a certificate is independent of whether the certificate is acquired through affiliation with an organization, or directly from the root. In any case the root does the actual certificate generation. Intermediate nodes merely vouch for the binding between IDs and public components.

When a user obtains a certificate through an organization, it is assumed that the user properly identifies himself to the organizational notary for the organization. The user may generate his own public/private pair.

To obtain a certificate, the following information pertaining to the user is sent to the root:

- Distinguished name
- Postal address
- Electronic mail address
- Message digest

Distinguished names are formed from legal names, organizations, and so on; they should be unique. The message digest is computed on the four items above concatenated with the user's public key. Canonical forms are needed for these items. The form containing these items is taken to an organizational notary if the user is affiliated with an organization possessing such a notary. Otherwise the user must take the information to a notary public.

In either case the user information is then sent to the root via the postal service. The file containing the user's public component and the information above, including the message digest, is sent concurrently to the root via electronic mail. A receipt and message digest computed on the user's public component and identifying information is returned to the user by the root via the postal service; the certificate is sent via electronic mail. The public component of the root is also sent to the user.

For an organization to issue certificates it must register with the root, obtaining a certificate as above. One organizational official represents the organization; this is merely a distinguished user. The organizational official becomes the first organizational notary (ON) for the organization; he may authorize others. Then there is one ON per organizational unit. Any ON may obtain certificates for users. The ON may vouch for only a portion of the user's information, for example, name.

Private components of organizations, rather than those of ONs, are used to sign certificates. That is, there is a distinction between ONs and CAs: Although ONs are certified to perform gathering and validating of data, they are otherwise undistinguished users. Only the root and organizations are certified to act as CAs. The root acts as CA for nonaffiliated users and users affiliated with organizations lacking an ON.

5.4.7 Revocation. Revocation of certificates may be necessitated by compromise of a user's private component or a change of distinguished name caused by a change of the user's affiliation with an organization or organizational unit.

The root should maintain a hot list of invalidated certificates or compromised keys. This may be available online. Hot lists might be issued about once per month.

In addition, each CA should maintain a time-stamped list of revoked certificates for users and other CAs. It is assumed that certificates contain serial numbers, unique for a given CA. Lists of revoked certificates can contain only serial numbers. It is recommended that each CA publish hot lists periodically, with each hot list containing the date of release of the next one.

5.4.8 Authentication and key exchange. Validation of certificates involves verification that the affixed signature is valid; that is, the hash value computed on cer-

tificate contents matches the value obtained by decrypting the signature field using the issuer's public component. This assumes that the latter has already been validated, that is, by possession of a validated certificate issued to the issuer in the event that the issuer is not the root. More generally, this involves recursive decryption of the certificates in a certification path; however, the authentication framework is assumed to have at most four levels, meaning that at most four certificates are involved.

To communicate with another user with whom the sender has not corresponded previously, the sender forwards a message to the recipient, with the X-certificate field of the message header containing the sender's certificate C_1. The message header may also contain the certificate C_2 of the issuer I of C_1 (see below). Assuming for simplicity that C_2 is present and was issued by the root, the recipient uses the root's public component to validate C_2 as above. The recipient then extracts the public component of I from C_2 and uses it to validate C_1 as above. The recipient in turn extracts the public component of the sender which is now validated.

The recipient now uses the sender's public component to decrypt the MIC, contained in the X-MIC-Info field of the message header. This permits verification of the authenticity and integrity of the message, by comparing the recovered MIC to the locally calculated MIC. We recall that the sender signed the MIC with his private component.

Once a user has validated the public component of another user, this may be used to encrypt a DEK which can in turn be used to encrypt messages. An encrypted DEK is placed in the X-Key-Info field of the message header. It is decrypted with the private component of the recipient.

The preceding process can be speeded up by caching of validated public components. Also, it has been suggested that multiple issuer fields could be contained in certificate headers; then entire certification paths could be represented, as assumed above in the case where the root is the issuer of the sender's certificate. This could serve as a substitute for a directory service (possibly on an interim basis).

5.4.9 Further notes. The X.509 Annex C recommendation of using the Fermat number $F_4 = 65{,}537$ as the common public exponent of RSA has received partial support. However, the use of 3 has also been suggested as a user option. Modulus size of 512–1200 bits has been suggested. Suggested hash functions include MD4 [RIVE90]. It must be emphasized that this is work in progress, however.

Uniqueness of the root has not been decided as yet. It is possible that several roots may be needed; these could cross-certify each other.

6 A SAMPLE PROPOSAL FOR A LAN IMPLEMENTATION

We present here a sample proposal for the implementation of privacy enhancement in a packet-switched LAN, using public key cryptography for key management and authentication. The main purpose is to explore the relevant decision-making process. In particular, some services will be needed in some settings but not others, and hence a monolithic structure incorporating all conceivable services would be inappropriate. One approach to the design of a generic structure is layered, with a kernel consisting of the most basic services and outer layers added as desired. This is the paradigm we adopt here.

A hybrid of public key cryptography and conventional cryptography is recommended. The former is used for signatures and for distribution of secret keys used in the latter; the latter is used for bulk data encryption. In addition, a hash function is needed so that only a compressed form of long text need be signed. We do not endorse specific public key systems or hash functions. The conventional system may be taken to be DES.

6.1 Integration into a Network

There are basically two modes of implementation of encryption in a network (e.g., [DIFF84], [DIFF86], [TANE81]), namely, link and end-to-end.

Link encryption provides good protection against external threats such as traffic analysis because all data flowing on links can be encrypted, including addresses. Entire packets, including addresses, are encrypted on exit from, and decrypted on entry to, a node. Link encryption is easy to incorporate into network protocols.

On the other hand, link encryption has a major disadvantage: A message is encrypted and decrypted several times. If a node is compromised, all traffic flowing through that node is also compromised. A secondary disadvantage is that the individual user loses control over algorithms used.

In end-to-end encryption, a message is encrypted and decrypted only at endpoints, thereby largely circumventing problems with compromise of intermediate nodes. However, some address information (data link headers) must be left unencrypted to allow nodes to route packets. Also, high-level network protocols must be augmented with a separate set of cryptographic protocols.

Here we assume end-to-end encryption. In terms of the OSI model, encryption can occur at various levels, including application, presentation, network, or transport. As noted in [ISO-87], the appropriate level depends on desired granularity of protection. In particular, high granularity refers to separate keys for each application or user and assurance of integrity. For this granularity the presentation or application layers are most appropriate. These two layers will be assumed here. In particular, integration at the application layer gives the individual user complete control over the algorithms used.

6.2 Security Threats

Some basic security threats (see Annex A of [CCIT87]) include:

- Masquerade
- Replay
- Interception of data
- Manipulation of messages
- Repudiation

Masquerade refers to users representing themselves as other users. *Replay* refers to recording and resending a message at a later time. *Interception of data* refers to passive eavesdropping on communications. *Manipulation* refers to unauthorized insertions, de-

letions, or other changes to messages. *Repudiation* refers to denial of sending (or possibly receipt) of a message.

There are other threats that are outside the present scope per se. For example, misrouting of packets naturally occurs in OSI layers 1–3, and we are restricting ourselves to higher layers.

6.3 Security Services

Again from Annex A of [CCIT87], some of the most basic security services that can be provided include:

- Authentication
- Secrecy
- Integrity
- Nonrepudiation

Authentication refers to verification of the identity of the sender or receiver of a communication. It can protect against masquerade. It may also provide protection against replay, depending on implementation. *Secrecy* refers to protection against interception of data. *Integrity* refers to protection against manipulation (including accidental) of data. *Nonrepudiation* refers to protection against denial of sending (or possibly receipt) of a message.

The kernel of a proposed system would normally support at least authentication, secrecy, and integrity. Nonrepudiation is somewhat more complex as we have noted. A public key signature system implements basic nonrepudiation of sending automatically, but does not a priori protect against repudiation of sending due to compromised private components, nor does it provide proof of receipt. Thus nonrepudiation is an example of a service (beyond the basic) which is most appropriately treated as an outer layer of an implementation, whose existence and structure depends on considerations such as contractual agreements that cannot be incorporated into a generic structure.

There are other services that could also be provided. One example is access control with differing capabilities for different classes of users. A public key system, however, does not provide this type of service per se, since it would require restricted access to public components. This type of service is thus another example of a layer which could be added in a given implementation, for example, by centralized key distribution which might be considered undesirable in many settings.

6.4 Security Mechanisms

Once more we follow Annex A of [CCIT87], which identifies some of the basic mechanisms that can implement security services. These include:

- Encryption
- Manipulation detection codes
- Signatures
- Authentication framework

Encryption refers to application of cryptographic transformations to messages or keys; it implements secrecy. *Manipulation detection* can be effected via hash functions producing a compressed version of the text; this implements integrity. *Signature* refers to application of private components to message digests (output of hash functions); it implements authentication and basic nonrepudiation, in concert with an authentication framework. The authentication framework is the system protocol and procedures within which authentication is achieved. Its form is implementation-dependent. For example, a directory service might be provided, along with hot lists of compromised keys.

The four mechanisms above may be regarded as the kernel of an implementation. Various other mechanisms that could be provided are noted in [ISO-87]. For example, to guard against replay, timestamps using synchronized clocks might be provided. Another possibility is the addition of a handshaking protocol. For enhancement of the basic nonrepudiation mechanism (the signature), a notary system could be used. Again these auxiliary services are best added at the discretion of implementors. For example, synchronized clocks may not be present, and notaries violate the desirable feature of point-to-point communication, and hence should not be included in standard specifications.

6.5 Criteria for Cryptosystems

There are various criteria that could be employed in selection of a cryptosystem (or systems) to implement key distribution and signatures. Logically these are separate functions and a hybrid of two separate systems could be used. Some relevant criteria (including suggestions from Dr. Dennis Branstad of the National Institute of Standards and Tests (NIST) and Dr. Ronald Rivest) are:

- Security
- Versatility
- Bandwidth
- Data expansion
- Key size
- Key generation time
- Patent restrictions/license fees
- Software support
- Hardware support
- Number of pseudorandom bits needed
- Interoperability

Most of these are self-explanatory. The reference to pseudorandom bits, above, is related to key generation, which requires a stream of pseudorandom bits generated from a random seed. *Interoperability* refers to current use and endorsement within and outside of the United States.

6.5.1 Security. The first and foremost criterion is of course security. It may take 5 years or more for a given method to be thoroughly cryptanalyzed, starting from the time it receives widespread public exposure.

One subcriterion in this regard is that a system should be published in an outlet such as a refereed journal with a reasonably large circulation, or a book or conference

proceeding that is present in libraries at larger academic institutions and research laboratories. This subcriterion is somewhat vague and not intrinsic, but may serve to avoid future embarrassment on the part of implementors.

A second subcriterion connected with security deals with mathematical and computational infrastructure. For example, the security of systems such as RSA is connected with the problem of factoring. It is safe to assume that thousands (or perhaps millions) of hours have been spent on this problem. This does not preclude major advances over the next decade or so, but at least guarantees that such advances will not occur simply because experts have suddenly become aware of a problem. In particular, systems based on long-standing problems such as factoring, discrete logarithm, and discrete root extraction have a certain degree of security in this regard. In contrast, for example, the El Gamal signature scheme can be broken if the "transcendental" equation $c = b^r r^s \pmod{n}$ can be solved for some r and s. This is easier than solving the logarithm or root problem $y = x^a \pmod{n}$, and furthermore in all likelihood the El Gamal equation has not been studied as extensively.

6.5.2 Numeric criteria. Quantitative criteria for cryptosystems include bandwidth (encryption and decryption rates), data expansion (relative sizes of plaintext and ciphertext), key size, and key generation time. We have already noted the phenomenon that systems that appear to be secure are also characterized by low bandwidths. Exponentiation-based methods such as RSA and the Diffie–Hellman exponential key exchange are examples. Other systems suffer from large data expansion, large key size, or long key generation time.

There seem to be some inherent trade-offs in this regard. That is, it does not seem possible to construct a system that is secure and also scores well on all numeric counts. The classic example is key size; for example, small key size in RSA would produce a high bandwidth, but would also produce insecurity. That is, in this case high bandwidth would produce low cryptanalysis time. Data expansion seems to have a similar impact. For example, in Appendix E it is noted that Shamir's knapsack attack runs in time which rises as n^d where d = expansion factor. Again high bandwidth produced insecurity. It would be interesting to know whether this trade-off notion could be formalized.

6.5.3 Other criteria. Versatility is an important criterion. The RSA system is distinguished in this regard since it supports both key distribution and signatures. All other major systems are limited to one or the other.

Patent restrictions and license fees may be a major factor in practice. Software and hardware support is another important practical matter.

6.6 Criteria for Hash Functions

In Section 3.2 we discussed some of the characterizations that have been proposed for hash functions, including measures of security. Some of the discussion in Section 6.5 applies here as well. For example, a hash function should be widely publicized for a period of time before it is trusted. Bandwidth is also an important criterion. Software and hardware support is relevant as well.

6.7 Some Recommendations

Finally we give a brief outline of a framework for incorporating public key cryptography into a LAN.

6.7.1 Key management. The recommended framework for key management is compatible with a subset of [CCIT87]. Public components of receivers are used as key-encrypting keys; private components of senders are used to encrypt message digests. DEKs are generated for individual sessions. These keys may be associated to an arbitrary conventional cryptosystem, although DES is recommended. The public and private components may be associated to different public key systems if different algorithms are used for key encryption and signatures.

Certificate-based key management is recommended. Since we are focusing on LANs, a simple tree structure is proposed, consisting of a root and one additional level containing all users. In particular, the root issues all certificates. Certification paths are thus trivial.

6.7.2 Component generation and storage. It is recommended that users generate their own public/private component pairs, using trusted software or hardware supplied by the system. Key pairs could be stored on smart cards [HAYK88] or tokens, along with the user's certificate. Such kernel mechanisms could be augmented by involvement of a central key distribution facility. However, we have noted this would negate a major advantage of public key cryptography, since compromise of the central facility would compromise all keys of users who obtained their keys from it.

6.7.3 Secret key generation. Numerous schemes exist for generation of data-encrypting keys. For example, in [MATY78] it is noted that keys can be generated by a conventional system such as DES. A master key might be used to encrypt DEKs or other key-encrypting keys. The master key, presumably long-term, is generated randomly. Other key-encrypting keys can be generated using DES as a pseudorandom generator. These are then encrypted under the master, which can also be involved in generation of other key-encrypting keys. A whole set of key-encrypting keys can be generated from a random 64-bit seed, with every eighth bit adjusted for parity. DEKs can be produced dynamically by pseudorandom number generators that are time-variant.

An example of a generator for DEKs, as well as initializing vectors (IVs), is given in Appendix C of [ANSI85]. Let $E(K, Y)$ be encryption by data encryption algorithm (DEA) (essentially equivalent to DES) with key K. Let K be a DEA key reserved for use in key generation and let V_0 be a 64-bit secret seed. Let T be a date/time vector, updated on each key or IV generation. Let

$$R_i = E(K, E(K, T_i) \text{ XOR } V_i)$$
$$V_{i+1} = E(K, R_i \text{ XOR } E(K, T_i))$$

Then R_i may be employed as an IV. To obtain a DEA key from R, reset every eighth bit to odd parity.

Routines for generating DES keys are supplied with various products. Schemes for pseudorandom number generation include [AKL-84], [BLUM84], [BRIG76]. The last reference gives a pseudorandom bit generator whose security is equivalent to discrete logarithm.

6.7.4 Issuance and distribution of certificates. Public components are registered with the root. The root generates a certificate containing the user's public component and identifying information, and a validity period. Distribution of certificates by users is recommended; certificates may be cached. This eliminates the necessity of having the root be online.

However, a user may wish to send a privacy-enhanced message to a user with whom he has not previously communicated, and who is currently unavailable. Thus it may be desirable to augment this kernel mechanism with a supplementary source of certificates. There are disadvantages to any augmentation. For example, if a phone book approach to certificates is used, entries may be inaccurate or altered. If a central directory mechanism is involved in key distribution it must be online. On the other hand, a central mechanism can provide instantaneous validity checks of public components and certificates.

The root should maintain a list of old public components for a period of time in event of disputes, for example, over-attempted repudiation.

6.7.5 Compromised or invalidated certificates. Assuming that certificates are cached, the root must periodically issue hot lists of invalidated certificates. This kernel service may be augmented in various ways to provide more current validity information. Again, however, additional mechanisms have drawbacks. As noted above, a central directory service could provide real-time validity checks, but it must be online. Furthermore, such checks a priori do not account for compromised private components, during the period following compromise but before a report is filed with the root. Even if users are required to report known compromises within a specified time period, a compromise may not become known to the user until later. As we noted, this creates an administrative and legal problem, since a user could disavow a signature on the basis of the latter type of compromise. A notary system can be used to add a layer of validity by having secondary signatures attest to the validity of senders' signatures, but this violates the desired criterion of point-to-point communication.

This is clearly an area in which solutions must be customized to some degree. For example, authentication in a system used for financial transactions may require more stringent controls than a kernel provides.

The root should maintain a time-stamped list of revoked certificates.

6.7.6 Authentication. A hash function is used to produce a message digest. This digest is signed with the private component of the sender. Timestamps and random numbers may also be included to prevent replay. If more than one hash function is permitted, identification of the function used should be included.

Privacy-enhanced communication between two users begins when A requests the certificate of B. This initial message contains A's certificate. As noted above, it is desirable if this request can be met directly by B. In this event, B first validates A's certificate. This uses the public component of the root, which is assumed to be known to all users. Then B forwards his certificate to A, who validates it. Each may cache the certificate of the other.

Now A and B may communicate securely. If A sends a message to B, B uses A's validated public component, extracted from A's certificate, to decipher the message digest. Then B may decrypt the key used for data encryption, using B's private component. Now B may decrypt the message text using this session key, then recompute the message digest and compare to the transmitted form.

7 MATHEMATICAL AND COMPUTATIONAL ASPECTS

We discuss here some issues related to the computational complexity of public key cryptography. The foundation of the security of such systems is the computational

infeasibility of performing cryptanalysis, in contradistinction to the relative ease of encryption/decryption. We analyze some of the issues that arise in this context. Also included is some background theoretical computer science needed to discuss the issue of computational complexity of cryptography and cryptanalysis, as well as zero-knowledge proofs and related schemes.

This discussion may aid in understanding the security basis of public key cryptography. It may also shed some light on the more practical matter of choosing key sizes, that is, the number of bits in private and public components. This section may safely be skipped by readers who do not wish to gain an in-depth understanding of such issues.

7.1 Computational Complexity and Cryptocomplexity

An ideal cryptosystem would have the property that encryption and decryption are easy, but cryptanalysis is computationally infeasible. In practice, this is commonly interpreted (e.g., [DIFF76b]) to mean that encryption/decryption should be executable in polynomial time, while ideally cryptanalysis should take exponential time. More generally, cryptanalytic time should be an exponential function of encryption/decryption time.

Unfortunately, it is difficult to determine when this criterion holds in practice. This is because it is usually very difficult to determine a nonpolynomial lower bound for the time required for cryptanalysis (or computations in general), even in instances when this process reduces to simple and well-known problems. In some cases the latter problems have been studied for centuries, but their computational complexity is still unknown.

In fact, whenever we try to determine the relative complexity of cryptanalysis and encryption, we encounter the problem of determining accurate lower bounds on computations.

Also, an analysis of security and efficiency of public key systems should take into account not only encryption/decryption time versus anticipated time for cryptanalysis, but also other parameters such as key size, key generation time, and data expansion. Developing a theory to characterize both security and practicality of cryptosystems seems very challenging.

7.2 Classic Complexity Theory

Here we briefly introduce some notions from the theory of computation. Some of these notions are given a more formal treatment in Appendix C.

One attempt to formalize the study of hard problems is the theory of NP-completeness (e.g., [GARE79], [HORO78]). The class P is defined (loosely) to be the class of decision problems solvable in polynomial time via a deterministic algorithm. The latter is essentially an algorithm executable on an ordinary sequential computer. A nondeterministic algorithm is a more ephemeral concept: It is essentially executable on a machine with unbounded parallelism. This means that an unlimited number of possibilities can be explored in parallel. For example, suppose a set of n items is to be checked for the presence or absence of a given item. The worst-case deterministic complexity is n, the number of operations needed by a sequential machine to check all n items. In contrast, the nondeterministic complexity is 1 since a machine with unbounded parallelism could check all n items in parallel regardless of n.

The class NP is (loosely) the class of decision problems solvable in polynomial time via a nondeterministic algorithm. Perhaps the single most important problem in computer science is to decide whether P = NP. The NP-complete problems are the hardest subclass of NP, having the property that if one instance of the subclass is in P then P = NP. Many classic combinatorial search problems can be given NP-complete formulations. The traveling salesman and knapsack problems are examples (see [GARE79] for a long list).

The class of NP-hard problems are those problems, decision or otherwise, that are at least as hard as NP-complete problems. Some NP-hard problems are so difficult that no algorithm will solve them; they are undecidable. Such problems cannot be in NP. An example is the halting problem (e.g., [HORO78]).

A more formal treatment of the classes P and NP requires a framework such as Turing machines (Appendix C).

7.3 Public Key Systems and Cryptocomplexity

The use of NP-complete problems to generate public key cryptosystems was suggested in [DIFF76b]. Later this approach materialized in the form of trapdoor knapsacks [MERK78b]. As we have noted, however, most knapsacks have been broken, despite the continuing intractability of the underlying NP-complete problem (integer programming). There are two separate explanations for this phenomenon. First of all, in most cases it has not been proven that the security of the public key system is equivalent to the intractability of the underlying problem.

Second, it is important to note that the classic theory of computational complexity has been founded around the cornerstone of worst-case complexity, with average-case complexity as a secondary criterion. As Rabin [RABI76] notes, these measures are of limited relevance in analyzing the complexity of cryptanalysis, since a cryptosystem must be immune to attack in almost all instances.

Attempts to formalize the notion of "almost-everywhere hardness" have been made (e.g., [GROL88]). An early characterization was suggested by Shamir [SHAM79], who quantifies this notion by defining a complexity measure $C(n, a)$ for algorithms as follows: $C(n, a)$ is a measure of an algorithm if at least a fraction a of problem instances of size n can be solved by the algorithm in time $C(n, a)$. For example, an algorithm for finding one factor of n could have complexity $C\left(n, \frac{1}{2}\right) = 0(1)$, since n has a 50% chance of being even. However, such investigations have thus far failed to produce a comprehensive framework, although they may be useful in a few special instances.

Another simple example illustrates the difference between worst-case and average-case complexity: As noted in [RABI76], algorithms to sort n items are often predicated on the assumption that the items are arranged in random order; that is, all $n!$ arrangements are equally likely. If the file has a real-world origin it is more likely that the file is already partially sorted. An optimized algorithm might anticipate this and utilize an adaptive strategy.

In practice, a cryptosystem for which cryptanalysis has an average-case polynomial complexity is generally worthless; in fact this remains true if any measurable fraction of all instances permits polynomial-time cryptanalysis (this is difficult to make precise, since the fraction may vary with key size). Thus there is no a priori connection between the breaking of a system and the difficulty of the underlying problem, since the

latter is characterized in terms of worst-case complexity. For example, there often exists a heuristic technique that yields an approximate solution to an NP-complete problem. Such techniques may not converge in polynomial time to an exact solution, but may break the corresponding cryptosystem.

In passing we remark that some authors (e.g. [WAGN84]) have attempted to go beyond the Diffie–Hellman notion of utilizing NP-complete problems to construct systems by using the even more intractable class of undecidable problems instead. It would be interesting to know if such systems can be made practical.

7.4 Probabilistic Algorithms

Another important distinction between the classic theory of complexity and cryptocomplexity is that the classic theory of algorithms does not encompass probabilistic algorithms (e.g., [RABI76]), which again may produce rapid solutions to problems in many instances but not even terminate in a few instances. They may also produce answers that are probably but not necessarily correct. Such algorithms are inappropriate in many contexts, for example, real-time control settings in which a response must be guaranteed in a fixed period of time, or where provable solutions are required. However, they are powerful tools in both cryptography and cryptanalysis.

As noted in [RABI76], probabilistic algorithms cannot be measured by classic criteria, which focus on worst-case runtime. Instead, a probabilistic algorithm may involve a trade-off between execution time and confidence. For example, most numbers have all small factors. If an algorithm exploits this fact it may terminate quickly for most inputs but take exponential time when large primes appear as factors.

Gill [GILL77] models probabilistic computation by extending the classic Turing machine model (e.g., [HORO78]) to the probabilistic Turing machine (Appendix D). This has proven valuable in the analysis of cryptographic systems, and probabilistic encryption schemes in particular (see Section 9.1).

7.5 Status of Some Relevant Problems

The security of several major cryptosystems mentioned here depends on the hardness of problems whose complexity status is unresolved. This includes factoring and discrete logarithm. The importance of the subclass of NP-complete problems emerges in this regard: If a problem is known to be in NP but is not known to be NP-complete, a solution to the problem in polynomial time (placing the problem in P) would not imply that problems such as traveling salesman are in P. The latter proposition is widely disbelieved.

A second relevant class is co-NP, the complements of problems in NP. For example, the complement of deciding whether n is prime is deciding if n is composite. In fact, primality and compositeness are in both NP and co-NP. Some experts have speculated (e.g., [GARE79]) that P is the intersection of NP and co-NP. For example, linear programming was known to be in both of the latter long before its status was resolved; eventually it was shown to be in P, via the ellipsoid method [KHAC79]. This illustrates that it would indeed be desirable to have available cryptosystems based on NP-complete problems (but see below).

Good surveys of the status of many problems important in security of public key systems are given in [ADLE86] and [POME86]. Let

$$L(n) = \exp((1 + o(1))(\ln n * \ln \ln n)^{1/2})$$

Then Adleman notes that many algorithms for factoring n have probabilistic execution times believed to be of the form $L(n)^c$ (e.g., [SCHN84]). However, only the algorithm of Dixon [DIXO81] has been rigorously analyzed. Similarly, various algorithms for discrete logarithms mod p are believed to take time $L(p)$. These results would be viewed with mistrust in the classic theory due to their probabilistic nature and lack of rigorous upper bounds for worst-case or average-case times. In contrast, Adleman [ADLE83] gives a deterministic algorithm for primality testing which executes in time

$$O((\ln n)^{c \ln \ln \ln n})$$

In the classic theory this result would be valuable, but we have noted that probabilistic algorithms are more efficient and far less complex, hence much more relevant in the present setting. Again this is indicative of the divergence between classic complexity and cryptocomplexity.

The status of the problem of taking roots mod n, that is, solving $x^e \equiv c \pmod{n}$, depends on the parameters (e.g., [SALO85]). A polynomial-time probabilistic algorithm to find x exists if e, c, and n are fixed and n is prime. If n is composite with unknown factors, no polynomial-time algorithm exists, even probabilistic. If $e = 2$ the complexity is essentially equivalent to factoring n (Lemma N.3.2). If $e > 2$ the status is less clear; as a consequence it is not clear that the security of RSA is equivalent to factoring.

Brassard [BRAS79] has shown that the discrete logarithm has an associated decision problem which lies in the intersection of NP and co-NP. Thus, if either factoring or taking discrete logarithms is NP-hard, it follows from the definition of this class that the associated decision problem is NP-complete and in co-NP. In turn this would imply NP = co-NP, which is widely disbelieved.

More generally, in [BRAS83] a function f is called restricted one-way if f is easily computable, f^{-1} is not, and f is injective and polynomially bounded. It is shown that if restricted one-way functions exist then P is not the intersection of NP and co-NP as many believe; and if f is restricted one-way and f^{-1} is NP-hard then NP = co-NP. Discrete logarithm is thought to be restricted one-way.

We conclude that there is little hope for fulfilling the Diffie–Hellman quest for public key systems based on NP-complete problems. The same may be true of the search for one-way functions to employ as hash functions. Prospects for use of NP-hard problems seem difficult to assess at this time.

8 AN INTRODUCTION TO ZERO-KNOWLEDGE

The notion of zero-knowledge proofs was introduced in [GOLD89]. The essence of zero-knowledge is that one party can prove something to another without revealing any additional information. In deference to the depth and scope of this area, we give only an illustrative example in this section. However, there is a close connection between zero-knowledge schemes and probabilistic public key encryption. Furthermore, some zero-knowledge schemes that have drawn attention because of applicability to smart card implementations, such as the one used as the basis of the example in this section, use Shamir's notion of identity-based public key systems. Thus the reader desiring a more formal introduction to these topics may wish to skip to Section 9.

Suppose that Alice knows a fact P. She wants to convince Bob that she knows P, but she does not trust Bob. Thus, Alice does not want to reveal any more knowledge to Bob than is necessary. What Alice needs is a zero-knowledge proof of P.

For example, suppose that Alice wants to prove to Bob that she really is Alice. Suppose for convenience that there is some authority that verifies identities. One possibility is that the authority could issue Alice an identification. If this were contained on a device such as a smart card, Alice could simply show it to Bob. However, if Alice and Bob are communicating over a network, then Alice's identifying information would have to be transmitted to Bob over the network. On receiving it, Bob could use it to impersonate Alice. Even if Bob were trusted, an eavesdropper such as Alice's adversary Carol could do the same.

This situation also arises commonly in computer access control: Bob might then be a host computer or network server, and Alice's identification might be a password. If Alice uses her password to identify herself, her password is exposed to the host software as well as eavesdroppers; anyone who knows this password can impersonate Alice.

It is thus desirable for Alice to be able to prove her identity without revealing any private information. More generally, we need a scheme through which Alice can prove to Bob that she possesses something (e.g., a password) without having to reveal it. Such a scheme is an example of a zero-knowledge proof. In fact, this example is the major practical use of zero-knowledge that has been suggested to date.

Here is one way that such a system could be organized: The authority decides on a number N used for everyone; for example, take $N = 77$. Everyone knows this number. The authority may then choose, for example, two numbers that form an ID for Alice. Suppose these are {58, 67}. Everyone knows Alice's ID. The authority then computes two other numbers {9, 10} that are given to Alice alone; she keeps these private. The latter numbers were chosen because $9^2 * 58 \equiv 1 \pmod{77}$ and $10^2 * 67 \equiv 1 \pmod{77}$.

Now Alice can identify herself to Bob by proving that she possesses the secret numbers {9, 10} without revealing them. Each time she wishes to do this she can proceed as follows: She can choose some random numbers such as {19, 24, 51} and compute

$$19^2 \equiv 53 \pmod{77}$$
$$24^2 \equiv 37 \pmod{77}$$
$$51^2 \equiv 60 \pmod{77}$$

Alice then sends {53, 37, 60} to Bob. Bob chooses a random 3 by 2 matrix of 0's and 1's, for example,

$$E = \begin{matrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{matrix}$$

Bob sends E to Alice. On receipt, Alice computes

$$19 * 9^0 * 10^1 \equiv 36 \pmod{77}$$
$$24 * 9^1 * 10^0 \equiv 62 \pmod{77}$$
$$51 * 9^1 * 10^1 \equiv 47 \pmod{77}$$

Alice sends {36, 62, 47} to Bob. Finally, Bob can check to see that Alice is who she says she is. He does this by checking that

$$36^2 * 58^0 * 67^1 \equiv 53 \pmod{77}$$
$$62^2 * 58^1 * 67^0 \equiv 37 \pmod{77}$$
$$47^2 * 58^1 * 67^1 \equiv 60 \pmod{77}$$

The original numbers {53, 37, 60} that Alice sent reappear. Actually, this doesn't really prove Alice's identity; she could have been an impersonator. But the chances of an impersonator succeeding would have been only 1 in 64.

In an actual system, the number N would have been much larger (e.g., 160 digits). Also, Alice would have been assigned an ID consisting of more numbers, for example, 4, by the authority, with a secret also consisting of four numbers. Furthermore, Alice would have generated more random numbers, for example, 5, to send to Bob. The ID numbers, secret numbers, and random numbers would have been about as large as N. This would have reduced an impersonator's chances of cheating successfully to about 1 in a million (more precisely 2^{-20}) if 4 and 5 are the parameters, which certainly would have convinced Bob of Alice's identity.

Why does this work? Because the authority chose {58, 67} and {9, 10} so that

$$9^2 * 58 \equiv 1 \pmod{77}$$
$$10^2 * 67 \equiv 1 \pmod{77}$$

This says that 9^2 and 58 are multiplicative inverses modulo 77, as are 10^2 and 67.

Thus

$$\begin{aligned}
36^2 * 58^0 * 67^1 &\equiv 19^2 * 9^{2*0} * 58^0 * 10^{2*1} * 67^1 \\
&\equiv 19^2 * (9^2 * 58)^0 * (10^2 * 67)^1 \\
&\equiv 19^2 \equiv 53 \pmod{77} \\
62^2 * 58^1 * 67^0 &\equiv 24^2 * 9^{2*1} * 58^1 * 10^{2*0} * 67^0 \\
&\equiv 24^2 * (9^2 * 58)^1 * (10^2 * 67)^0 \\
&\equiv 24^2 \equiv 37 \pmod{77} \\
47^2 * 58^1 * 67^1 &\equiv 51^2 * 9^{2*1} * 58^1 * 10^{2*1} * 67^1 \\
&\equiv 51^2 * (9^2 * 58)^1 * (10^2 * 67)^1 \\
&\equiv 51^2 \equiv 60 \pmod{77}
\end{aligned}$$

Thus the checks that Bob uses serve their purpose properly; that is, Alice is identified. Also, Bob has learned nothing that would permit him to masquerade as Alice; nor has Carol, who may have been eavesdropping. Either would need to know that 9^2 and 10^2 are the multiplicative inverses of 58 and 67, respectively, modulo 77. To see this, suppose Carol tries to convince Bob that she is really Alice; that is, Carol pretends that {58, 67} is her own ID. For simplicity, suppose Carol tries to identify herself as Alice by generating the same random {19, 24, 51}. Then she sends {53, 37, 60} to Bob. Again for simplicity suppose Bob generates the same E and sends it to Carol. Now Carol is in trouble; she doesn't know the numbers 9 and 10, and can only guess them. The protocol requires Carol to send three numbers, say $\{x, y, z\}$, to Bob. Then Bob will check:

$$x^2 * 58^0 * 67^1 \equiv 53 \pmod{77}$$
$$y^2 * 58^1 * 67^0 \equiv 37 \pmod{77}$$
$$z^2 * 58^1 * 67^1 \equiv 60 \pmod{77}$$

Carol will have succeeded in her masquerade if she chose $\{x, y, z\}$ to make these checks come out right. But $67^{-1} \equiv 10^2 \equiv 23 \pmod{77}$ and $58^{-1} \equiv 9^2 \equiv 4 \pmod{77}$, so $x^2 \equiv 53 * 23 \equiv 64 \pmod{77}$, $y^2 \equiv 37 * 4 \equiv 71 \pmod{77}$, and $z^2 \equiv 60 * 23 * 4 \equiv 53 \pmod{77}$. Could Carol solve these quadratic equations to find, for example, $x = 36$, $y = 62$, $z = 47$? For a small value of N such as $N = 77$, she could indeed. However, if N is a product of two appropriately chosen large primes (e.g., each 80 digits or more), and if these primes are kept secret by the authority, then the answer is no. That is, computing square roots modulo a composite is computationally infeasible (Appendix N). Thus, anyone who does not know Alice's secret ($\{9, 10\}$ in the example above) cannot impersonate her when N is large.

Another possibility is that Alice's interaction with Bob might give Bob information that could allow impersonation of Alice, at least on one occasion, by replay. Suppose Bob tries to convince Carol he is really Alice. Bob might try to imitate Alice by sending $\{53, 37, 60\}$ to Carol to start the protocol. Now Carol doesn't necessarily select the E above. Suppose she selects

$$F = \begin{matrix} 1 & 1 \\ 0 & 0 \\ 0 & 1 \end{matrix}$$

and sends F to Bob. Now Bob is in trouble; he might try to imitate Alice again by sending $\{36, 62, 47\}$ to Carol. Then Carol will check

$$36^2 * 58^1 * 67^1 \equiv 71 \pmod{77}$$

Since $71 \not\equiv 53 \pmod{77}$, Carol knows Bob is a fraud even without the other two checks. Can Bob send some $\{r, s, t\}$ instead of $\{36, 62, 47\}$? This will only work if

$$r^2 * 58^1 * 67^1 \equiv 53 \pmod{77}$$
$$s^2 * 58^0 * 67^0 \equiv 37 \pmod{77}$$
$$t^2 * 58^0 * 67^1 \equiv 60 \pmod{77}$$

As before, this gives $r^2 \equiv 58^{-1} * 67^{-1} * 53 \equiv 25 \pmod{77}$, and similarly $s^2 \equiv 37 \pmod{77}$, $t^2 \equiv 71 \pmod{77}$. As above, Bob can solve these quadratic equations to find r, s, t because $N = 77$ is small, but could not do so if N were large. Also, in the example above there is one chance in 64 that Carol would choose $F = E$, in which case Bob's deception would go undetected; but for larger parameters such as 4 and 5 instead of 2 and 3, this would again be improbable (one chance in a million (2^{-20}) instead of 64).

Another possibility is that when Alice identified herself to Bob, she may have inadvertently revealed some information that could enable Bob to learn her secret, that is, $\{9, 10\}$, that would permit impersonation. For this protocol to be zero-knowledge this should not be possible. Can Bob deduce the numbers $\{9, 10\}$ from the two sets of numbers $\{53, 37, 60\}$ and $\{36, 62, 47\}$, and E? He knows Alice started with three numbers $\{a, b, c\}$, but he doesn't know these were $\{19, 24, 51\}$. He knows Alice's secret is

$\{u, v\}$ but doesn't know these are $\{9, 10\}$. He knows the authority computed u and v from

$$u^2 \equiv 58^{-1} \equiv 4 \pmod{77}$$
$$v^2 \equiv 67^{-1} \equiv 23 \pmod{77}$$

He also knows Alice computed

$$a^2 \equiv 53 \pmod{77}$$
$$b^2 \equiv 37 \pmod{77}$$
$$c^2 \equiv 60 \pmod{77}$$
$$a * u^0 * v^1 \equiv 36 \pmod{77}$$
$$b * u^1 * v^0 \equiv 62 \pmod{77}$$
$$c * u^1 * v^1 \equiv 47 \pmod{77}$$

This gives Bob eight equations from which to deduce $\{u, v\}$. However, the last three are redundant, and as we have noted, the first five cannot be solved when N is large.

The following list shows informally that the protocol above works:

1. It identifies Alice by proving she possesses the secret $\{9, 10\}$.
2. It identifies Alice uniquely: Anyone who doesn't know this secret cannot impersonate Alice.
3. Alice's secret is not revealed in the process of proving she possesses it.

Actually, (1) means the possessor of $\{9, 10\}$ is identified as the system user who has ID = $\{58, 67\}$ assigned by the authority. Also, no matter how many times Alice proves her identity, her secret will not be revealed (if N is sufficiently large); that is, (3) states loosely that the protocol is zero-knowledge. All this depends on the assumption that equations of the form $x^2 \equiv y \pmod{N}$ cannot be solved if N is the product of two large primes that are not known, but can be solved easily if the primes are known (Appendix N). Such equations lie at the heart of most concrete zero-knowledge schemes.

The formal definition of zero-knowledge is more complicated (see [GOLD89]), but the example above demonstrates its essence in a context which has been proposed as a candidate for actual implementation in smart card–based identification schemes.

9 ALTERNATIVES TO THE DIFFIE–HELLMAN MODEL

Thus far this chapter has been based on the work of Diffie and Hellman [DIFF76b]. This model of public key cryptography has received some criticism on two grounds:

- Security of most Diffie–Hellman-type systems is difficult to characterize formally.
- Security of Diffie–Hellman-type systems is dependent on a superstructure that binds user IDs and public components.

In Section 7 we noted that it has proven difficult to develop a comprehensive axiomatic framework in which to establish the security of Diffie–Hellman-type public key systems. For example, it is difficult to guarantee that partial information about plaintext (e.g., its least significant bit) cannot be recovered from the corresponding ciphertext even though the entire plaintext cannot be found.

In Section 5 we noted some examples of authentication frameworks (e.g., use of certificates) that bind user IDs and public keys. Without such a superstructure a public key system is useless, since a user would not be certain that he was employing the correct public key for encryption or decryption. The security of the public key system thus depends on proper functioning of the authentication framework, which is a priori unrelated to the underlying cryptosystem.

In this section we briefly examine two modifications of the basic Diffie–Hellman model. In one case the goal is to incorporate the binding between a user ID and public key directly into the cryptosystem, thereby eliminating the separation between the cryptosystem and the authentication framework. The other scheme addresses the subject of knowledge concealment; it is closely related to zero-knowledge proofs. Both schemes have received considerable attention not only because of possibly enhanced security, but also because of their potential relevance to smart card implementations of public key cryptography.

9.1 Probabilistic Encryption

Goldwasser and Micali [GOLD84] note that the public key systems of Diffie and Hellman are not provably secure. They observe that use of a trapdoor function f to generate such systems does not exclude the possibility that $f(x) = y$ may be solvable for x without the (original) trapdoor under certain conditions. Also, even if x cannot be found from $f(x)$, it may be possible to extract partial information about x. One problem is that such trapdoor systems proceed block by block. This makes it difficult to prove security with respect to concealment of partial information such as least significant bit.

In [GOLD84] Goldwasser and Micali suggest an alternative to trapdoor-based public key systems. Their procedure is to encrypt bit by bit. They call this probabilistic encryption. It introduces several advantages, namely, uniformly difficult decoding and hiding of partial information. Their scheme has the following properties:

1. Decoding is equivalent to deciding quadratic residuosity modulo a composite N, whose factorization is not known to an adversary.

2. Suppose a predicate P has probability p of being true in message space M. For $c > 0$, assuming intractability of quadratic residuosity, an adversary given ciphertext cannot decide with probability $> p + c$ whether the corresponding plaintext satisfies P; that is, the adversary does not have a c-advantage in guessing P.

In item (1) the reference is to the problem of deciding for a given x whether there is a y such that $y^2 \equiv x \pmod{N}$. As in the case of computing discrete logarithms or extracting roots, this is computationally infeasible if the factorization of N is unknown (Appendix N).

An example of item (2): If the messages consist of uniformly distributed bit strings and P is "least significant bit $= 0$" then $p = \frac{1}{2}$. If the Goldwasser–Micali scheme is employed, an adversary cannot guess the least significant bit of plaintext with probability greater than $\frac{1}{2} + c$. It may be very difficult to verify that traditional public key systems conceal such partial information.

The public key system proposed by Goldwasser and Micali is as follows: A user A chooses primes p and q and makes public $N = p * q$; p and q are private. Also, A selects a random y with $(y/N) = 1$ (Appendix N.2) with y a quadratic nonresidue modulo N; y is public. By Lemma N.4.6, y can be found in probabilistic polynomial time.

To send the binary string $m = (m_1, \ldots, m_k)$ to A, B randomly selects $\{x_1, \ldots, x_k\}$ in Z_N^* and computes

$$z_i = x_i^2 \mod N \qquad (i = 1, \ldots, k)$$

Then B sets $e_i = z_i$ if $m_i = 0$, $e_i = y * z_i$ otherwise. The encoding of m is $e = (e_1, \ldots, e_k)$, which B sends to A. To decode e, A sets $m_i = 1$ if e_i is a quadratic residue modulo N, and $m_i = 0$ otherwise. This can be effected by A in polynomial time since A knows p and q (Lemmas J.4.1, N.1.1). It is correct because $y * z_i$, the product of a quadratic nonresidue and residue, is a quadratic nonresidue modulo N (Lemma N.2.2).

The security of partial information follows from the fact that each bit of the message m is encoded independently. A disadvantage of the technique is data expansion: if $|N| = n$, then n bits are transmitted for each bit of a message.

One application is coin flipping by telephone: A randomly chooses r in Z_N^* with $(r/N) = 1$, where (r/N) is the Jacobi symbol (Appendix N.2). Then B guesses whether or not r is a quadratic residue modulo N; B wins if and only if he guesses correctly. The probability of the latter is $\frac{1}{2}$; that is, exactly half of the elements of Z_N^* satisfying $(r/N) = 1$ are quadratic residues modulo N (Lemma N.2.1). The correctness of B's guess can be checked by A; the result can be verified by B if A releases the factorization of N to B.

A second application is to mental poker (e.g., [DENN83b] pp. 110–117). Goldwasser and Micali [GOLD82] show that their implementation corrects some deficiencies in hiding partial information in a scheme of Shamir, Rivest, and Adleman.

Quadratic residuosity modulo a composite is an example of a trapdoor function. Yao [YAO-82] showed that under certain conditions, trapdoor functions in general can be used to construct provably secure probabilistic public key cryptosystems. An important role is also played by probabilistic encryption in zero-knowledge proofs, especially for problems in NP.

We remark briefly that the scheme above is only one of a class of encryption schemes that Rivest and Sherman [RIVE82] term randomized encryption techniques; various other schemes are summarized there. We have characterized schemes such as the one above as bit-by-bit; more generally, a *randomized scheme* is any scheme in which a ciphertext for a given plaintext is randomly chosen from a set of possible ciphertexts. Rivest and Sherman also develop a classification for such schemes. A major aspect of randomized schemes is often significant data expansion. For example, the Goldwasser–Micali scheme would probably have ciphertext around 512 times as large as the plaintext. On the other hand, probabilistic schemes may be much faster than their deterministic counterparts, an attractive feature for smart card implementations.

9.2 Identity-Based Schemes

In [SHAM84], Shamir suggests yet another modification of traditional public key systems. In Shamir's framework a user's public key coincides with his system ID. This eliminates the need for any superstructure for distribution of public keys. It also trivial-

izes the authentication of public keys. However, unlike a traditional public key scheme this modification requires a trusted key generation center.

The center issues a smart card to each user, containing the user's private key. Thus the "private" key is really a shared secret key. A card can generate the user's digital signature. The center does not maintain a database of keys; in fact it need not even keep information beyond a list of user IDs (needed to ensure that there are no duplicate IDs).

Security for such a system differs from the usual requirement that a user keep his private key a secret. The latter condition is retained, in that a user must guard his card. As in a certificate-based traditional system, the center must ascertain a user's identity before issuing a card. In addition, however, the cryptographic function used by the center to generate private keys from IDs must be kept secret. Typically this function would employ trapdoor information such as the factorization of a modulus. The requirement is that computing the private key from an ID is easy if the trapdoor information is known, but knowing any polynomial number of public/secret pairs should not reveal the trapdoor.

A weakness in such a scheme is that anyone (intruder or insider) possessing the trapdoor information can forge the signature of any user. This is in contradistinction, for example, to a credit card scheme in which a transaction is generally valid only if accompanied by a user's physical signature. Also, the center is not required to maintain any information on lost or stolen cards; thus the loss of a card is disastrous since the possessor can forge the legitimate user's signature indefinitely. Furthermore, the center may find itself involved in litigation produced by any such security problems, since it is providing the means by which users generate signatures. Again this is in contrast to the credit card situation: If a credit card is stolen, the thief cannot forge the legitimate holder's physical signature, providing a means of distinguishing between legal and illegal use of the card. Use of passwords (PINs) with cards could largely eliminate the problems accruing from lost or stolen cards, but compromise of the trapdoor would still be disastrous.

Shamir's notion was later ([FEIG87], [FIAT86]) incorporated into zero-knowledge schemes. One of these was used as the basis for the example in Section 8.

APPENDIX A
TIMINGS

We briefly summarize some timings related to RSA obtained by Keller at NIST [KELL89]. These were obtained by using either software or a combination of software and hardware. Tests were run on an International Business Machines (IBM) PC AT.

Time to generate RSA public/private key pairs was as follows:

- Software
  –Keysize = 32 bytes: 18 sec
  –Keysize = 64 bytes: 122 sec
- Software/hardware:
  –Keysize = 32 bytes: 4 sec
  –Keysize = 64 bytes: 12 sec

Rates of encryption and decryption can be found in [KELL89].

APPENDIX B
ALGORITHMS AND ARCHITECTURES

We briefly survey some of the considerations relevant to determining what type of hardware and software support for cryptanalysis, or to a lesser extent encryption, may be forthcoming from the creators of algorithms and architectures of the future. We also mention some examples already in existence. This represents an excursion into high-performance computing, only a small fraction of which is applicable to cryptography. Nonetheless, to evaluate security of methods or ascertain key sizes, it is necessary to make some educated guesses as to how this field will progress over the next few decades.

B.1 TECHNOLOGY

Computing at present is silicon-based. Thus all estimates of achievable computer performance are geared to this technology. The question arises as to whether a radically different technology such as superconductivity or optical computing will make silicon-based performance standards obsolete, and if so, when this might occur. We have no idea, and hence we ignore these questions.

Gallium arsenide (GaAs) technology is another matter. It has already been integrated into some existing supercomputers. Some of the differences between GaAs and silicon very large-scale integration (VLSI) are (e.g., [MILU88]):

- GaAs gates have a higher switching speed.
- Off-chip communication in GaAs pays a relatively higher penalty.
- GaAs chips have lower density.

Gate delays in GaAs (DCFL E/D-MESFET*) may be as low as 50 psec, as opposed to at least 1 nsec in silicon (NMOS). Similarly, on-chip memory access in GaAs may take as little as 500 psec, as opposed to at least 10 nsec in silicon. This indicates that performance of GaAs-based computers could theoretically be as much as 20 times greater than even the fastest silicon-based supercomputers. However, GaAs levels of integration are currently much lower: at most about 50,000 transistors per chip, as opposed to 1,000,000 in silicon. This is due to problems in GaAs of power dissipation, yield, and area limitations. Thus the number of chips necessary to build systems using GaAs is higher; minimizing chip count is important for high performance.

Off-chip communication in GaAs is another factor at the system level. The peak performance of a computer system is limited by the bandwidth of the slowest subsystem [HWAN84]. Interchip signal propagation is not significantly different for silicon or GaAs, but the relative effect is different: off-chip communication is more of a bottleneck in GaAs because of its ratio to on-chip speed. Silicon solutions to the problem of central processing unit (CPU) speed versus other subsystem speeds include cache and multilevel memory hierarchies; these may not carry over mutatis mutandis to GaAs.

---

*DCFL E/D-MESFET is the acronymn for direct coupled FET logic enhancement/depletion metal-semiconductor field effect transistor.

We conclude that at the moment, GaAs technology does not present a real threat to silicon performance standards. Furthermore, it does not appear likely that problems such as yield and integration will be solved in the near future. Thus, for the remainder of Appendix B we assume no radical change in technology from the present.

B.2 COMPUTING MODES

Classically, computing was dominated by the Von Neumann model, with a single processor executing a single instruction scheme on a single data stream. This paradigm reached its peak with computers such as the Cray-1 [CRAY80] and CYBER 205 [CONT80]. However, the performance of a single processing unit is limited by its clock speed. It does not seem feasible to reduce major cycles much below 1 nsec with silicon technology. Furthermore, at these speeds memory access becomes a bottleneck, not to mention input/output (I/O). Thus it is not likely that uniprocessors will exceed 10^9 operations per second, barring radical new technologies.

Two alternatives to the Von Neumann model are parallel and distributed computing (e.g., [HWAN84]). Parallel computing generally refers to processors in one box; distributed means each processor is in its own box. Parallel computers may (loosely) be subdivided into shared-memory, in which all processors share a common address space, and distributed memory, in which each processor has its own address space. These modes remove the restriction on the number of instruction and/or data streams that may be processed concurrently. In theory these modes of computing can produce unlimited computing power, by splicing together single processing nodes and memory units (or processor/memory pairs) into a unified system. However, there are three major limitations in this regard:

- Cost-effectiveness
- The interconnection network
- Parallel algorithms.

Cost-effectiveness alone has been a deterrent to the development of parallel systems. The Denelcor HEP (e.g., [KOWA85]), Floating Point T-Series [HAWK87], and ETA-10 [STEI86] are examples of parallel systems that have not proven to be commercially successful. Also, Cray and other supercomputer manufacturers have been reluctant to expand into large-scale parallelism, partially because of cost-related concerns. This creates an interesting situation from a cryptographic point of view: There may be cases where a computer powerful enough to break a given system could be built in theory. Security of the system might then rest on the assumption that no one will spend the money to build it, or that the only units built will belong to wealthy government agencies and will be subject to tight controls.

Even if computers with a thousand or more powerful processors are constructed, the question arises as to whether such a configuration can achieve computing power in proportion to the number of processors, that is, linear speedup. The mode of interconnecting the processors and memories is critical. If the shared-memory configuration is used, with a collection of processors accessing common memory partitioned into mod-

ules, interference in paths through the network or in simultaneous attempts to access a module will cause a bottleneck if a low-cost, low-bandwidth network such as a bus is used. If a high-bandwidth network such as a crossbar is used (i.e., with concurrent data transfers possible between many processors and memory modules), the number of units interconnected is limited by cost which rises as the square of the number of processors. Thus such systems seem to be inherently limited in the extent to which they can improve on uniprocessor performance.

An alternative is to connect processor/memory pairs using a network such as a hypercube [SEIT85]. Machines of this type with up to 65,536 weak processing elements (e.g., the Connection Machine [HILL85]) or up to 1024 powerful processors (e.g., the NCUBE/10 [HAYE86]) have been constructed, and systems with up to 32,000 Cray-1 level processors have been proposed. There is debate (and some controversy) over the speedups that such distributed-memory machines can achieve. The latter consideration is related to cost-effectiveness, which requires nearly linear speedup. Also, the cost of interconnection in a hypercube-based system rises as $O(n \log n)$, where n is the number of processor/memory pairs.

Another concern is algorithms [QUIN87]. A problem must be highly decomposable to be amenable to parallel or distributed computation. Furthermore, algorithms for non–Von Neumann machines must be tailored to individual architectures to a much greater extent than their Von Neumann counterparts, adding to the cost of software development.

Because of inherent trade-offs between performance and cost, there may be a divergence between the computing power attainable in theory and that which is practical (and in the event of commercial machines, marketable). Within 10 years it is conceivable that machines executing 10^{12} operations per second could be built with refinements of existing technology; the real question seems to be financing.

An alternative to the single-machine approach is the use of networks for completely distributed computing (e.g., [LENS89]). The class of problems amenable to solution via networks (and wide area networks in particular) is restricted, since the nodes must communicate infrequently (typically only at the beginning and end of a computation). Nonetheless, in Section 4 it was noted that some of the strongest cryptanalytically related results have been obtained by networks of computers. This is possible because many of the relevant cryptanalytic algorithms are fully decomposable into independent portions that do not need to interact. An example is the quadratic sieve.

It may be possible to assemble more computing power in such a network than is present in any single computer. Once again, the constraints are largely pragmatic. Designers of cryptosystems must therefore attempt to anticipate not only advances in algorithms and architectures, but also the greatest amount of computing power that might realistically be brought to bear against a given task.

B.3 SOME RELEVANT ALGORITHMS AND IMPLEMENTATION

Most of the powerful algorithms for factoring and discrete logarithm are fairly new. As we have noted, most of them have not been fully analyzed for runtimes. Nonetheless, some standards have emerged in this regard. The best guess for the future can only amount to anticipation of improvements in the present algorithms.

B.3.1 Quadratic Sieve Factoring Algorithm

The quadratic sieve [POME84] provides an alternative to the earlier continued fraction factorization algorithm [MORR75].

As noted in [DAVI83b], the continued fraction approach uses considerable multiple precision division. The quadratic sieve works with larger residues but involves mainly single-precision subtraction. Both have runtimes of exp(sqrt(c ln n ln ln n)) but $c = 2$ for continued fraction, $c = \frac{9}{8}$ for the quadratic sieve.

In [DAVI84] the results of implementing the quadratic sieve on a Cray X-MP [CRAY85] are reported. Factorization of 70-digit numbers takes about an hour; factorization of 100-digit numbers should take about a year. The key to this match of algorithm and architecture is the use of the vector capability of the Cray architecture. In particular, a steady stream of operands is necessary to take advantage of a pipelined, register-to-register machine. The loops in the sieve are amenable to streaming of operands, and about 75% efficiency was obtained. However, the multitasking capability of the X-MP was not utilized. Since Crays with up to 16 processors are expected in the near future, with even greater parallelism to be anticipated, it follows that the results of such experiments are probably too conservative. On the other hand, using both vector and parallel capabilities of a system is nontrivial, partially as a result of memory bank contention. Furthermore, adaption of the algorithm to exploit both capabilities may be nontrivial. Hence it is not clear to what extent the preceding efficiency can be maintained as the degree of parallelism grows.

An alternative is distributed computing: Silverman [SILV87] has used the quadratic sieve implemented via a network of nine SUN 3 workstations to factor numbers up to 80 digits in 8 weeks. Recently another network was used to factor 106-digit numbers [LENS89].

It should be noted that the results obtained via the quadratic sieve are directly relevant cryptanalytically, since the integers factored are general. Integers of up to 155 digits, but with special forms, have been factored, also by use of networks [SIAM90]. However, these results are only indirectly relevant to systems such as RSA, assuming moduli are properly chosen.

B.3.2 Computations in Finite Fields

Computations in finite fields is a subject that has been explored fairly thoroughly (e.g., [BART63]) and we will not review it here.

Multiplication of elements in GF(m) has classically been implemented at the circuit level using linear feedback shift registers. However, Laws [LAWS71] has noted the potential for using cellular arrays. These are highly amenable to VLSI implementation. A pipeline architecture for multiplication and inverses is proposed in [WANG85].

One class of algorithms of particular interest is for discrete logarithms in GF(p^n). Still more particularly, the case $n = 2$ has been explored considerably ([BLAK84], [BLAK84b], [COPP84]). Since finite logarithms are easy to compute in GF(m) if m has only small factors [POHL78], n should be chosen so that $2^n - 1$ is (a Mersenne) prime, for example, $n = 521$. However, Odlyzko [ODLY84b] notes that a next-generation supercomputer might break $n = 521$ in a year. A special-purpose computer might attack n on the order of 700 in a year.

Odlyzko also notes that with similar bounds on key size, GF(p) may be preferable to GF(2^n); for example, $n = 2000$ is roughly equivalent to p of about 750 bits. As noted above, this advantage may be counterbalanced by implementation efficiency.

B.3.3 Other Algorithms

Many of the most fundamental operations in this setting seem to be characterized by inherent sequentiality. An example is exponentiation; computing the nth power seems to take log n steps regardless of the number of processors available.

Another classic example is GCD. Although some partial results are noted in [ADLE86], major improvement over Euclid does not seem forthcoming regardless of advances in architecture.

Many of the powerful factoring algorithms are highly parallelizable (e.g., [SCHN84]). The main requirement is existence of appropriate architectures.

B.4 APPLICATION-SPECIFIC ARCHITECTURES

General-purpose architectures such as that of the Cray supercomputer are of interest in this setting because of their immediate availability. At the other extreme, architectures closely matching the algorithms of interest could be constructed, but their over-specialized nature would virtually preclude their commercial distribution. In between are classes of architectures that are more versatile but not truly general-purpose. We note several examples of partly specific architectures and a proposed highly-specific machine.

B.4.1 Systolic and Wavefront Arrays

The notion of systolic arrays ([KUNG82], [KUNG78]) is an extension of pipelining. The idea is to have a collection of processors operate on data that are pulsed rhythmically through the array. If the original requirement of a synchronous mode of operation is relaxed, the result is a wavefront array [KUNG82b]. Both types were originally targeted at applications such as signal processing that are compute-intensive and involve repetitions of operations on many operands.

A systolic array for the computation of GCDs is proposed in [BREN83]. It is very amenable to VLSI implementation because of its regular topology. Furthermore it is linear, limiting I/O requirements which can constitute a bottleneck in VLSI implementations. Supporting algorithms are noted in [BREN83b].

It would be of interest to know what a more general-purpose linear systolic array such as the Warp [ANNA87] (which curiously more closely resembles a wavefront array) could accomplish on some of the problems discussed in this chapter, and factoring in particular. The Warp is already in production.

B.4.2 Proposal for a Quadratic Sieve Machine

Pomerance, Smith, and Tuler [POME88] describe a pipeline architecture that would efficiently execute the quadratic sieve. This notion is of considerable interest to anyone

implementing an algorithm such as RSA, since such a machine could presumably factor numbers beyond the reach of present-day supercomputers.

Cost-effectiveness was carefully analyzed by these authors, as is critical for an application-specific architecture. Pomerance et al. speculate that a $50,000 version of the architecture should factor 100-digit numbers in about 2 weeks, or 140-digit numbers in a year on a $10 million version, or 200-digit numbers in 1 year on a $100 billion version. It should be noted that the last example is clearly predicated on the notion that the architecture and algorithm will scale linearly with the number of processing units; however as we noted in Section B.2 various bottlenecks such as interprocessor communication and memory access make this assumption suspect. Nonetheless the possibility arises that moduli approaching 200 digits may be necessary in RSA because of the potential existence of machines such as these.

The crux of the architecture are the pipe and pipe I/O units. These should be custom-made and should be able to handle variable strides without the considerable loss of efficiency that usually accompanies nonconstant stride. The two units should be chained together. The pipe should consist of stages, all of which are bus-connected to the pipe I/O unit. The cycle time of memory should be the same as that of the processing elements.

It is interesting to note that this architecture bears a close resemblance to a linear systolic array.

B.4.3 Massively Parallel Machines

An alternative to pipelined machines is to configure a large number of primitive processing elements in an array and have them execute the same instruction stream synchronously, processing a large quantity of data in parallel [single instruction multiple data (SIMD) mode]. Such machines are generally applied to real-time image or signal processing, or to large-scale matrix operations.

The massively parallel processor (MPP) [BATC80] is an example. It consists of 16,384 processors in a square array. It was intended mainly for image processing of data from satellites; but in [WUND83] and [WILL87] it is used for an implementation of the continued-fraction factoring algorithm. Wunderlich [WUND85] also implements this algorithm on the distributed array processor (DAP) (e.g., [HOCK81]), another massively parallel machine with 4096 processors. Unlike the MPP, the DAP has been marketed.

APPENDIX C
THE CLASSIC THEORY OF COMPUTATION

In Section 7 a number of topics from the classic theory of computation were mentioned. Here we give a more precise treatment of some notions from the classic theory (e.g., [LEWI81]).

An *alphabet* is a finite set of symbols. A *string* is a sequence of symbols from an alphabet. A *language* on an alphabet is a set of strings from the alphabet. If S is an alphabet, S^* denotes the set of all strings from S, including the empty string. Concatenation of strings a and b is written ab. If A and B are subsets of S^*, AB is the set of strings formed by concatenating elements from A and B.

C.1 TURING MACHINES

A (one-tape, deterministic) Turing machine is a quadruple (K, S, D, s) where:

S is an alphabet that contains a blank = #, but not the symbols L or R which are reserved for tape movement to the left or right

K is a set of states that does not include the halt state = h, which signals an end to computation

s = initial state; it is in K

$D : K \times S \rightarrow (K + \{h\}) \times (S + \{\text{L, R}\})$ where + denotes union

A Turing machine = M may be interpreted semiphysically as consisting of a control unit and a tape. At any time M is in some state, and the read/write head of the tape is over some symbol on the tape. If $D(q, a) = (p, b)$ then initially the head is scanning a and M is in state q; its next move will be to enter state p. If b is in S the head will set $a := b$ without moving; otherwise b = L or R in which case the head will move to the left or right. M halts when state h is entered, or M hangs (the left end of the tape is surpassed). The tape has no right end.

Input to M consists of a string. The string is padded by a # on each side and placed on the leftmost squares of the tape. The rest of the tape consists of #'s. Initially the head is positioned over the # that marks the right end of the input. Initially M is in state s; its first move is thus $D(s, \#)$.

At a given time M is in configuration (q, w, a, u),

where:

q = state (element of $K + \{h\}$)

w = portion of the tape to the left of the head (element of S^*)

a = symbol under the head (element of S)

u = portion of the tape to the right of the head

If e denotes the empty string, u in the list above is required to be an element of $(S^*)(S - \{\#\}) + \{e\}$; that is, either $u = e$, meaning the tape has all #'s to the right of the head, or the last symbol of u is the last nonblank symbol to the right of the head position. This gives the configuration a unique description. In particular, if the input to M is w then the initial configuration of M is $(s, \#w, \#, e)$.

M is said to halt on input w if some halted configuration (state = h) is reached from $(s, \#w, \#, e)$ in a finite number of steps; then it is said that $(s, \#w, \#, e)$ yields a halted configuration. If M halts on input w, M is said to accept w. The language accepted by M is the set of strings w accepted by M. Conversely, a language is said to be Turing-acceptable if it is accepted by some Turing machine.

Suppose S is an alphabet, T and W are subsets of S, and $f : W \rightarrow T$. Suppose there exists a Turing machine $M = (K, S, D, s)$ such that for any w in W, the configuration $(s, \#w, \#, e)$ yields $(h, \#u, \#, e)$, that is, u is the output of M on input w, and furthermore $u = f(w)$. Then f is said to be computed by M.

Suppose alphabet A does not contain #, Y, or N. Suppose L is a language in A^* and X is its characteristic function, that is, for w in A^*, $X(w) = Y$ if w is in L, $X(w) = N$ otherwise. If X is computed by Turing machine M then M is said to decide

(or recognize) L. Conversely, if X is the characteristic function of a language L and X is Turing-computable, that is, there exists a Turing machine that computes X, then L is said to be Turing-decidable.

An extension of the basic model is to permit the control unit to control (a finite number of) multiple tapes. However, a language accepted by a multitape Turing machine is also accepted by a one-tape Turing machine; that is, additional tapes do not increase the computing power of Turing machines in terms of expanding the class of languages accepted.

C.2 NONDETERMINISTIC TURING MACHINES

The preceding Turing machines were deterministic; that is, D was a function: if $D(q, a) = (p, b)$ then the next state p and scanned symbol b were uniquely determined by the present state q and symbol a. A nondeterministic Turing machine is defined similarly except that D is now a relation on

$$(K \times S) \times ((K + \{h\}) \times (S + \{\mathrm{L}, \mathrm{R}\}))$$

That is, D is now multiple-valued, so that the next configuration is no longer uniquely determined by the present. Instead, in a given number of steps a configuration may yield a number of configurations. Consequently an input may yield many outputs. A sequence of steps starting from a given input defines a computation; in general, many computations are possible on one input. A nondeterministic machine is said to accept an input if there exists a halting computation on it. Again the language accepted by a machine is the set of strings accepted by it. Trivially any language accepted by nondeterministic machines is also accepted by deterministic machines, which are merely special cases of the more general nondeterministic case.

It is also true (but not as trivial) that any language accepted by a nondeterministic Turing machine is also accepted by a deterministic Turing machine. That is, nondeterminism does not increase the power of Turing machines insofar as the class of languages is concerned.

Similar extensions hold for decidable languages.

C.3 COMPUTATIONAL COMPLEXITY

The time complexity of Turing machines is measured by the number of steps taken by a computation. If T is a function on the nonnegative integers, a deterministic Turing machine M is said to decide language L in time T if it decides in time $T(n)$ or less whether w is or is not in L, where w has length n. If T is a polynomial then L is said to be decided in polynomial time. If a language is decidable in polynomial time on a multitape deterministic Turing machine then it is decidable in polynomial time on a one-tape Turing machine.

A nondeterministic Turing machine M is said to accept w in time T if there is halting computation on w of $T(n)$ or fewer steps, where w has length n. M is said to accept language L in time T if it accepts each string in L in time T.

The class of languages decidable in polynomial time on some deterministic Turing machine is denoted by P. The class of languages acceptable in polynomial time on some nondeterministic Turing machine is denoted by NP.

It is not known whether P = NP.

APPENDIX D
THE THEORY OF PROBABILISTIC COMPUTING

Probabilistic algorithms are employed as adjuncts in cryptosystems for purposes such as finding primes. They have also produced virtually all major practical cryptanalytic algorithms for factoring, discrete logarithms, etc. Here we review an extension of the classic theory of computation which incorporates probabilistic computing. This extension has proven particularly valuable in the study of probabilistic cryptosystems.

An ordinary deterministic multitape Turing machine may be considered to have an input tape, an output tape, and read-write worktapes. A modification of the ordinary model (e.g., [GILL77]) is the probabilistic Turing machine. It has a distinguished state called the coin-tossing state, which permits the machine to make random decisions. In terms of languages recognized, these have the same power as deterministic machines. However, time considerations are more subtle. In particular, a notion of probabilistic runtime is needed, rather than measures such as maximum runtime used for ordinary machines.

A probabilistic Turing machine operates deterministically, except when it is in a special coin-tossing state (or states). In such a state the machine may enter either of two possible next states. The choice between these is made via the toss of an unbiased coin. The sequence of coin tosses may be considered to constitute the contents of an auxiliary read-only input tape, the random tape, which contains a binary string. Thus a computation by a probabilistic machine is a function of two variables, the ordinary input tape and the random tape.

If the random tape is unspecified, the output of the computation of probabilistic machine M, $M(x)$, is a random variable (e.g., [MCEL78]): M produces output y with probability $Pr\{M(x) = y\}$. For a given input x, there may exist a y such that $Pr\{M(x) = y\} > \frac{1}{2}$. Such a y is clearly unique if it exists, in which case we can write $q(x) = y$. This defines a partial function: q is undefined if no such y exists. The partial function q is said to be computed by M. The set accepted by M is the domain of q.

If x is accepted by M let $e(x) = Pr\{M(x) \; ! = y\}$. It is direct from definition that $e(x) < \frac{1}{2}$. Suppose there exists a constant $c < \frac{1}{2}$ such that $e(x) \leq c$ for all x in the domain of e. Then it may be said that M has bounded error probability (the error probability e is bounded away from $\frac{1}{2}$), another concept important in zero-knowledge frameworks.

Again leaving the random tape unspecified, $Pr\{M(x) = y \text{ in time } n\}$ is the probability that probabilistic Turing machine M with input x gives output y in some computation of at most n steps. The probabilistic runtime $T(x)$ of M is defined to be infinity if x is not accepted by M, that is, if x is not in the domain of the partial function q computed by M. If x is in the domain of q, $T(x)$ is the smallest n such that $Pr\{M(x) = q(x) \text{ in time } n\} > \frac{1}{2}$. This is somewhat analogous to defining the runtime of a nondeterministic Turing machine to be the length of the shortest accepting computation.

A function is probabilistically computable if it is computed by some probabilistic Turing machine. There are many important examples of functions that are probabilistically computable in polynomial probabilistic runtime and bounded error probability, but are not known to be computable in polynomial deterministic time, that is, in P. An example is the characteristic function of the set of primes, which is probabilistically computable in polynomial time (e.g., [SOLO77]), but for which no deterministic algorithm is known.

Let BPP be the class of languages recognized by polynomial-bounded probabilistic Turing machines with bounded error probability. Letting $<$ denote inclusion, it follows easily that $P < BPP < NP$. An important question, with implications for schemes such as RSA as well as zero-knowledge schemes, probabilistic encryption, and so on is whether either of these inclusions is proper.

APPENDIX E
BREAKING KNAPSACKS

We give a brief account of the demise of the Merkle–Hellman trapdoor knapsack public key system and some of its variants. A much more complete discussion is given in [BRIC88].

We recall from Section 4.2.1 that the security of this approach rests on the difficulty of solving knapsack problems of the form

$$C = b_1 * M_1 + \ldots + b_n * M_n$$

where the $\{b_i\}$ are obtained from superincreasing $\{a_i\}$ by modular "disguising":

$$b_i = w * a_i \mod u$$

Around 1982, several authors (e.g., [DESM83]) made the observation that if $W * w \equiv 1 \pmod{u}$, where W may be found as in Appendix H, then for some $\{k_i\}$,

$$a_i = W * b_i - u * k_i$$

In particular

$$a_1 = W * b_1 - u * k_1$$

and hence

$$b_i * k_1 - b_1 * k_i = (b_1 * a_i - b_i * a_1)/u$$

Since $u > a_1 + \ldots + a_n$, $b_i < u$, and $a_1 < a_i$,

$$|b_i * k_1 - b_1 * k_i| < ua_i/(a_1 + \ldots + a_n)$$

Now it is easily shown that

$$a_{i+j} \geq 2^{j-1} * (a_i + 1)$$

and hence

$$|b_i * k_1 - b_1 * k_i| < 2^{i+1-n} * u$$

Thus

$$|k_1/k_i - b_1/b_i| < 2^{i+1-n} * u/k_i b_i < 2^{i+1-n} * u/b_i$$

This shows that the $\{k_i\}$ are in fact not random; they are determinable via the inequalities above if u is known. Shamir [SHAM84b] observed that an intruder merely

seeks any trapdoor, as represented by any u, w, and $\{a_i\}$ that produce an easy knapsack. Thus the last inequality may be regarded as an integer programming problem. In this particular instance Shamir noted that the algorithm of Lenstra [LENS83] is applicable, together with classic methods of Diophantine approximation. This yields the $\{k_i\}$, and the system is then broken easily.

Lenstra's algorithm made use of the Lovasz lattice basis reduction algorithm [LENS82], one of the basic tools in "unusually good" simultaneous Diophantine approximation [LAGA84]. This approach was utilized by Adleman [ADLE82] to break the Shamir–Graham knapsack, and by Brickell [BRIC84] to break the iterated Merkle–Hellman knapsack. The multiplicative version was broken by Odlyzko [ODLY84]. In fact, all major proposed knapsacks based on modular disguises have been broken using this approach.

It should be noted that the low-density attacks ([BRIC83], [LAGA83]) are successful where finding trapdoors fails. These use a measure of density for a knapsack with coefficients $b_1, \ldots, b_n$ defined by density $= n/(\log \max \{b_i\})$. This type of attack is independent of whether the knapsack is a disguised version of another. Trapdoors, in contrast, are easiest to find for high-density knapsacks.

The concept of density is related to two important parameters of cryptosystems, namely, information rate and expansion factor d. In [LAGA84] the information rate is defined to be the ratio of the size of plaintext to the maximum size of the ciphertext. This is the reciprocal of d, that is, ciphertext/plaintext. Information rate is essentially the same as density, although for the above knapsack and modulus u it is defined slightly differently, namely as $n/(\log n * v)$. Both definitions are derived from approximations to the actual ciphertext size, which is $\log(b_1 * M_1 + \ldots + b_n * M_n)$. Lagarias [LAGA84] notes that the attack in [SHAM84b] runs in time $O(P(n) * n^d)$ for a polynomial P. Hence the attack is feasible if d is fixed but not if the expansion factor d is large. This illustrates the interrelation between security and practicality.

APPENDIX F
BIRTHDAY ATTACKS

In Section 4 we noted several uses of birthday attacks against hash functions. Here we give a brief summary of the relevant mathematics.

Suppose H is a function that has m possible outputs. Whenever H outputs a value it is totally random and independent of any previous values which have been output.

If H is evaluated k times, each output is akin to an object placed in one of m cells corresponding to the range of H. Since the k values of H are independent, any object could be placed in any of m cells; the total number of ways of distributing the k objects is m^k. If no two objects are to be placed in any cell (i.e., if there are to be no collisions in applying H k times), the first object can be placed anywhere; the second can go in any of the $m - 1$ remaining cells; the third in any of $m - 2$ cells, etc. The total number of ways of distributing the objects is $m(m - 1) \ldots (m - k + 1)$ (sometimes called a falling factorial). The probability of no collisions is $m(m - 1) \ldots (m - k + 1)/m^k$. Hence the probability of at least one collision is

$$\begin{aligned} P(m, k) &= 1 - (m - 1) \ldots (m - k + 1)/m^{k-1} \\ &= 1 - (1 - 1/m) \ldots (1 - (k - 1)/m) \end{aligned}$$

This yields:

Lemma F.1: Suppose the function H, with m possible outputs, is evaluated k times, where $m > k > (2cm)^{1/2}$ for some constant c. Then the probability of at least one collision (i.e., x and y with $H(x) = H(y)$ for some x, y) is at least $1 - e^{-c}$, $e = 2.718 \ldots$

Proof: for $0 < x < 1$,

$$1 - x < 1 - x + x^2(1 - x/3)/2 + x^4(1 - x/5)/24 + \ldots = e^{-x}$$

For $k < m$ this gives

$$(1 - 1/m) \ldots (1 - (k - 1)/m) < e^{-1/m \ldots -(k-1)/m} = e^{-k(k-1)/2m}$$

Thus

$$P(m, k) > 1 - e^{-k(k-1)/2m}$$

The lemma follows. ■

Example F.1: Suppose the H of Lemma F.1 is evaluated k times where $k > (2(\ln 2)m)^{1/2} = 1.17 * m^{1/2}$. Then the probability of at least one collision is $> \frac{1}{2}$.

This suggests an attack on hash functions. Its name derives from the classic problem of computing the probability that two members of a group of people have the same birthday.

APPENDIX G
MODULAR ARITHMETIC AND GALOIS FIELDS

We give a brief introduction to arithmetic modulo n where n is a positive integer. A ring structure may be imposed on $Z_n = \{0, 1, \ldots, n - 1\}$ by doing addition, subtraction, and multiplication mod n (e.g., [DENN83], [HERS64] or any book on elementary number theory). We use GCD for greatest common divisor; for example, GCD(x, y) = n if n is the largest positive integer dividing x and y. For $n > 1$ let

$$Z_n^* = \{x \in Z_n : x > 0 \quad \text{and} \quad \text{GCD}(x, n) = 1\}$$

For example, $Z_{10}^* = \{1, 3, 7, 9\}$. That is, Z_n^* consists of the nonzero elements of Z_n that are relatively prime to n.

If $a \in Z_n$ let

$$a * Z_n^* = \{a * x \mod n : x \in Z_n^*\}$$

For example, $2 * Z_{10}^* = \{2, 6, 14, 18\} \mod 10 = \{2, 6, 4, 8\}$, $3 * Z_{10}^* = \{3, 9, 21, 27\} \mod 10 = Z_{10}^*$. We have:

Lemma G.1: Suppose $n > 1$ and $a \in Z_n^*$. Then

1. For any x and y: $a * x \equiv a * y \pmod{n}$ iff $x \equiv y \pmod{n}$.

2. $a * Z_n^* = Z_n^*$.
3. a has a multiplicative inverse modulo n, that is, there exists $b \in Z_n^*$ such that $b * a \equiv 1 \pmod{n}$.

Proof: If $x \equiv y \pmod{n}$ then trivially $a * x \equiv a * y \pmod{n}$. Conversely, suppose $a * x \equiv a * y \pmod{n}$. Then $n \mid (a * (x - y))$. But $\text{GCD}(a, n) = 1$, so $n \mid (x - y)$, that is, $x \equiv y \pmod{n}$. Thus (1) holds.

For (2): if $x \in Z_n^*$ then $\text{GCD}(x, n) = 1$, so $\text{GCD}(a * x, n) = 1$. Thus $a * x \bmod n \in Z_n^*$. Hence $a * Z_n^*$ is contained in Z_n^*. By (1), if $a * x \equiv a * y \pmod{n}$, then $x \equiv y \pmod{n}$. Thus $a * Z_n^*$ consists of n distinct elements, and cannot be a proper subset of Z_n^*; (2) follows. In particular, since $1 \in Z_n^*$, there exists some $b \in Z_n^*$ such that $a * b \bmod n = 1$; (3) follows. ■

G.1 THE EULER PHI FUNCTION

The cardinality of a set S is denoted by $|S|$. In particular, $|Z_n^*|$ is denoted by $\phi(n)$, the Euler totient function.

Lemma G.1.1:

1. If p is prime then $\phi(p) = p - 1$.
2. If $n = p * q$, p and q prime, then $\phi(n) = (p - 1)(q - 1)$.

Proof: (1) is trivial, since $Z_p^* = \{1, \ldots, p - 1\}$.

For (2): let $Y_p = \{p, \ldots, (q - 1)p\}$, that is, the nonzero elements of Z_n divisible by p. Let $Y_q = \{q, \ldots, (p - 1)q\}$, that is, the nonzero elements of Z_n divisible by q. If $a * p = b * q$, then $p \mid b$ and $q \mid a$; hence $a * p$ and $b * q$ cannot be in Y_p and Y_q, respectively. Thus Y_p and Y_q are disjoint. Letting + denote disjoint union,

$$Z_n = \{0\} + Y_p + Y_q + Z_n^*$$

Taking cardinalities,

$$p * q = 1 + q - 1 + p - 1 + |Z_n^*|$$

Then (2) follows. ■

G.2 THE EULER–FERMAT THEOREM

Lemma G.2.1 (Euler's theorem): Suppose $n > 0$ and $a \in Z_n^*$. Then

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

Proof: If $Z_n^* = \{r_1, \ldots, r_m\}$ then a restatement of (2) of Lemma G.1 is

$$\{a * r_1 \bmod n, \ldots, a * r_m \bmod n\} = Z_n^*$$

Hence

$$a * r_1 * \ldots * a * r_m \equiv r_1 * \ldots * r_m \pmod{n}$$

Thus

$$a^m * r_1 *. \ldots * r_m \equiv r_1 * \ldots * r_m \pmod{n}$$

By (1) of Lemma G.1, each r_i above may be canceled, leaving

$$a^m \equiv 1 \pmod{n}$$

Noting that by definition $m = \phi(n)$ gives Euler's theorem. ■

Corollary G.2.1 (Fermat's theorem): Suppose p is prime and $a \in Z_p^*$. Then

$$a^{p-1} \equiv 1 \pmod{p}$$

Proof: Use (1) of Lemma G.1.1 in Lemma G.2.1. ■

Corollary G.2.2: Suppose $n > 0$, $a \in Z_n^*$, and $x \equiv 1 \pmod{m}$ where $m = \phi(n)$. Then $a^x \equiv a \pmod{n}$.

Proof: We have $x = m * y + 1$ for some y. Now by Lemma G.2.1,

$$a^x \equiv (a^m)^y * a \equiv 1^y * a \equiv a \pmod{n}$$ ■

Corollary G.2.3: Suppose $n > 0$. Let $m = \phi(n)$. Suppose e and d are in Z_m^* and $e * d \equiv 1 \pmod{m}$. Let $E(M) = M^e \bmod n$ and $D(C) = C^d \bmod n$. Then $D(E(M)) = E(D(M)) = M$ for any M in $[0, n)$.

Proof:

$$D(E(M)) \equiv (M^e \bmod n)^d \bmod n \equiv M^{e*d} \bmod n \equiv M \pmod{n}$$

The last step uses Corollary G.2.2. Also $0 \leq D(E(M)) < n$ and $0 \leq M < n$, so $D(E(M)) = M$. Similarly $E(D(M)) = M$. ■

G.3. GALOIS FIELDS

Lemma G.1 shows that Z_n^* is an Abelian group (e.g., [HERS64], p. 27) under the operation of multiplication modulo n; Z_n^* is called the multiplicative group of units of Z_n.

In particular, if p is prime then $Z_p^* = \{1, \ldots, p-1\}$ is a group; that is, each nonzero element of Z_p has a multiplicative inverse modulo p. Thus the ring $Z_p = \{0, 1, \ldots, p-1\}$ is in fact a finite field (e.g., [HERS64], p. 84) with p elements.

It can be shown (e.g., [HERS64], p. 314) that every finite field has p^n elements for some prime p. These are called the Galois fields, and denoted GF(p^n). We have already noted that GF(p) = Z_p is defined by doing arithmetic modulo p. The elements of Z_p are called the residues modulo p; that is, each integer x has a unique representation $x = q * p + r$ where $r \in Z_p$. To define GF(p^n) we choose an irreducible polynomial $f(x)$ of degree n in the ring of polynomials modulo p (e.g., [DENN83], p. 49).

Now arithmetic may be defined on this ring of polynomials, modulo $f(x)$; that is, write $g(x) \equiv h(x)$ iff $f(x)$ is a divisor of $g(x) - h(x)$. Each polynomial $g(x)$ has a unique representation $g(x) = q(x)f(x) + r(x)$ for some polynomial $r(x)$ of degree at most $n - 1$. The residues modulo $f(x)$ are the elements of GF(p^n). These consist of polynomials of degree $\leq n - 1$ with coefficients from Z_p. Each of the n coefficients of a residue has p possible values, accounting for the p^n element count for GF(p^n).

APPENDIX H
EUCLID'S ALGORITHM

On a number of occasions we referred to Euclid's algorithm, which can be used to find GCDs and multiplicative inverses. We give some versions of it here. These versions are recursive, and minimize storage requirements.

Suppose x and y are arbitrary positive integers. Their greatest common divisor GCD(x, y) can be computed in O(log max{x, y}) steps by recursively employing GCD(s, t) = GCD(s, t mod s). This is Euclid's algorithm:

```
function GCD(x,y) returns integer;
  /* return GCD of x > 0 and y > 0 */
  s := x; t := y;
  while (s > 0) do
    div := s; s := t mod s; t := div;
  end while;
  return(div);
end GCD;
```

This is readily generalized to

```
function Multi_GCD(m; x : array[0..m]);
  /* for m > 0 return GCD of
     x_1, ..., x_m > 0 */
  if (m = 1) return(x_1)
  else return(GCD(x_m, Multi_GCD(m - 1, x)));
end Multi_GCD;
```

The above runs in O(m * log max{x_i}) time.

For $x \in Z_n^*$, with the latter as in Appendix G, a simple extension of GCD yields the multiplicative inverse of x modulo n, that is, u with $x * u \equiv 1 \pmod{n}$. With $[y]$ denoting the largest integer $\leq y$, the extended Euclid algorithm to find u is:

```
function INVERSE(n, x) returns integer;
  /* for n > 0 and x ∈ Z_n^* return u ∈ Z_n^*
     with u * x ≡ 1 (mod n) */
  procedure Update(a, b);
    temp := b; b := a - y* temp; a := temp;
  end Update;
  g := n; h := x; w := 1; z := 0; v := 0; r := 1;
  while (h > 0) do
```

```
    y := [g/h]; Update(g, h); Update(w, z); Update(v, r);
  end while;
  return(v mod n);
end INVERSE;
```

For example, for $n = 18$, $x = 7$, this algorithm gives the values

g:	18	7	4	3	1
h:	7	4	3	1	0
w:	1	0	1	−1	2
z:	0	1	−1	2	−7
v:	0	1	−2	3	−5
r:	1	−2	3	−5	18
y:		2	1	1	3

Finally $u = -5 \mod 18 = 13$ is returned. This is equivalent to finding the sequence $\frac{7}{18}, \frac{2}{5}, \frac{1}{3}, \frac{1}{2}, \frac{0}{1}$ in which each neighboring pair is a pair of Farey fractions; $\frac{a}{b}$ and $\frac{c}{d}$ are Farey fractions (e.g., [RADE64]) if $a * d - b * c = \pm 1$. We have found $2 * 18 - 5 * 7 = 1$, so that $-5 * 7 \equiv 1 \pmod{18}$, or equivalently $13 * 7 \equiv 1 \pmod{18}$. Thus finding multiplicative inverses modulo n can also be done in $0(\log n)$ steps.

This will also find s with $u * x + s * n = 1$: take $s = (1 - u * x)/n$. More generally we have (with GCD(x) = x):

```
procedure Coeffs(m; x : in array[1..m]; u : out array[1..m]);
  /* given m > 0 and x_1, ..., x_m > 0 with GCD(x_1, ..., x_m) = 1,
    find u_1, ..., u_m with u_1x_1 + ... + u_mx_m = 1 */
  if (m = 1) return(x_1)
  else
    g = Multi_GCD(m − 1, x);
    for i := 1 to m − 1 do y_i := x_i/g;
    coeffs(m − 1, y, u');
    u_m := INVERSE(g, x_m);
    b := (1 − u_m * x_m)/g;
    for i := 1 to m − 1 do u_i := b * u_i';
  end else;
  end Coeffs;
```

This runs in $O(m * \log \max\{x_i\})$ time.

APPENDIX I
THE CHINESE REMAINDER THEOREM

The Chinese remainder theorem is useful for purposes such as simplifying modular arithmetic. In particular, we note in Appendix M that its use increases the efficiency of decryption in RSA.

We begin with a special case: with Z_n as in Appendix G, we have

Lemma I.1: Suppose p and q are primes and $p < q$. Then:

1. For arbitrary $a \in Z_p$ and $b \in Z_q$, there exists a unique $x \in Z_{pq}$ with

$$x \equiv a \pmod{p}$$
$$x \equiv b \pmod{q}$$

2. If $u * q \equiv 1 \pmod{p}$, the x in (1) is given by

$$x = (((a - b) * u) \bmod p) * q + b$$

Proof: We note that for any y and s, $0 \leq y \bmod s \leq s - 1$. Thus if x and u are as in (2), $0 \leq x \leq (p - 1) * q + q - 1 = p * q - 1$. Hence $x \in Z_{pq}$. Trivially $x \equiv b \pmod{q}$. Also

$$\begin{aligned} x &\equiv (((a - b) * u) \bmod p) * q + b \\ &\equiv (a - b) * u * q + b \\ &\equiv (a - b) * 1 + b \\ &\equiv a \pmod{p} \end{aligned}$$

Thus x is a solution to the simultaneous linear system in (1). To show that it is unique, suppose $x' \equiv a \pmod{p}$ and $x' \equiv b \pmod{q}$. Then for some k and m, $x - x' = k * p = m * q$. Thus $k = q * k'$ for some k'; hence $x - x' = p * q * k'$, that is, $x' = x - p * q * k'$. Since $0 \leq x \leq p * q - 1$, if $k' > 0$ then $x' < 0$; if $k' < 0$ then $x' \geq p * q$. Hence $x' \in Z_{pq}$ iff $k' = 0$, that is, $x' = x$. ■

We note that in (2) above, u can be found via the INVERSE function of Appendix H.

The condition $p < q$ is arbitrary, but useful in noting

Corollary I.1: Suppose p and q are primes, $p < q$, $0 \leq x < p * q$, and $a = x \bmod p$, $b = x \bmod q$. Then

1. $x = (((a - (b \bmod p)) * u) \bmod p) * q + b$
$(a \geq b \bmod p)$

2. $x = (((a + p - (b \bmod p)) * u) \bmod p) * q + b$
$(a < b \bmod p)$

Proof: Immediate from Lemma I.1. ■

Corollary I.1 provides an optimal framework for representing x via a and b, that is, the residues of x modulo p and q, respectively.

Although the most general case of the Chinese remainder theorem is not used in this exposition, we remark that the above can be extended:

Theorem I.1: Given pairwise relatively prime moduli $\{p_1, \ldots, p_n\}$ and arbitrary $\{a_1, \ldots, a_n\}$, there exists a unique $x \in [0, p_1 * \ldots * p_n)$ satisfying $x \equiv a_i \pmod{p_i}$ for each i.

Proof: A straightforward generalization of the above (e.g., [DENN83], p. 47).

APPENDIX J
QUADRATIC RESIDUES AND THE JACOBI SYMBOL

Quadratic residues play a role in primality testing as we note in Appendix N. In Sections 8 and 9 we also noted their use in encryption.

Let Z_n^* be as in Appendix G. For a positive integer n and $a \in Z_n^*$, a is called a quadratic residue modulo n if $x^2 \equiv a \pmod{n}$ for some x; otherwise a is a quadratic nonresidue.

Lemma J.1: Suppose $n > 0$ and a is a quadratic residue modulo n. Then every y with $y^2 \equiv a \pmod{n}$ has the form $y = x + k * n$ where k is an integer and $x \in Z_n^*$.

Proof: Suppose $y^2 \equiv a \pmod{n}$. Let $x = y \bmod n$. Then $x \in [0, n)$. Also, $x^2 \equiv a \pmod{n}$. Now $x^2 = a + j * n$ for some j. Suppose for some m we have $m > 0$, $m \mid x$ and $m \mid n$. Then $m \mid a$, so $m = 1$ since $\text{GCD}(a, n) = 1$. Hence $x \in Z_n^*$; also $y = x + k * n$ for some k. ■

Thus the square roots of a quadratic residue modulo n can be taken to be elements of Z_n^*; all other square roots are obtained by adding multiples of n to these.

J.1 QUADRATIC RESIDUES MODULO A PRIME

If p is prime and $0 < a < p$, a is a quadratic residue modulo p iff $x^2 \equiv a \pmod{p}$ for some x with $0 < x < p$. For example, the quadratic residues modulo 7 are

$$\{1^2, 2^2, 3^2, 4^2, 5^2, 6^2\} \bmod 7 = \{1, 2, 4\}$$

The quadratic nonresidues modulo 7 are $\{3, 5, 6\}$.
Given a prime p, let $s = (p - 1)/2$ and

$$S_p = \{x^2 \bmod p : 0 < x \leq s\}$$

Then S_p is a set of quadratic residues modulo p. In fact we have

Lemma J.1.1: Suppose $p > 2$ is prime. Let $s = (p - 1)/2$. Then

1. The elements of S_p are precisely the quadratic residues modulo p.
2. There are s quadratic residues modulo p.
3. There are s quadratic nonresidues modulo p.

Proof: As noted above, S_p is a subset of the set of quadratic residues modulo p. Furthermore, if $x^2 \equiv y^2 \pmod{p}$ then $p \mid x^2 - y^2$; hence $p \mid x - y$ or $p \mid x + y$. If x and y are in $(0, s]$ then $1 < x + y < p$; thus $p \mid x - y$; hence $x = y$. It follows that S_p contains distinct elements.

Now suppose a is a quadratic residue, $x^2 \equiv a \pmod{p}$, and $x \in Z_p^*$. We note

$$(p - x)^2 \equiv p^2 - 2 * p * x + x^2 \equiv a \pmod{p}$$

Since $0 < x < p$, either x or $p - x$ is in $(0, s]$. It follows that $a \in S_p$. Thus the set of quadratic residues modulo p is contained in S. Hence the two sets are identical, establishing (1). Since $|S_p| = s$, (2) follows. Also, the complement of S_p in Z_p^* is the set of quadratic nonresidues modulo p. Since $|Z_p^*| = 2s$, the complement of S_p also has cardinality s; (3) follows. ■

J.2 THE JACOBI SYMBOL

If p is a prime > 2 and $0 < a < p$, the Legendre symbol (a/p) is a characteristic function of the set of quadratic residues modulo p (e.g., [RADE64]):

1. $(a/p) = 1$ if a is a quadratic residue mod p.
2. $(a/p) = -1$ if a is a quadratic nonresidue mod p.

More generally, if $k > 1$ is odd and h is in Z_k^*, the Jacobi symbol (h/k) may be defined as follows:

1. The Jacobi symbol (h/p) coincides with the Legendre symbol if p is prime.
2. If $k = p_1 * \ldots * p_m$ with p_i prime, $(h/k) = (h/p_1) * \ldots * (h/p_m)$.

An efficient mode for computing the Jacobi symbol is via the recursion:

1. $(1/k) = 1$
2. $(a * b/k) = (a/k)(b/k)$
3. $(2/k) = 1$ if $(k^2 - 1)/8$ is even, -1 otherwise
4. $(b/a) = ((b \mod a)/a)$
5. If GCD$(a, b) = 1$:
   a. $(a/b)(b/a) = 1$ if $(a - 1)(b - 1)/4$ is even
   b. $(a/b)(b/a) = -1$ if $(a - 1)(b - 1)/4$ is odd

The key step in the recursion is (5), which is Gauss's law of quadratic reciprocity (e.g., [RADE64]). The above list shows that the Jacobi symbol (a/n) can be computed in O(log n) steps.

J.3 SQUARE ROOTS MODULO A PRIME

Regarding solutions of $x^2 \equiv a \pmod p$, that is, the square roots of a modulo p, we have:

Lemma J.3.1: Suppose $p > 2$ is prime. Let $s = (p - 1)/2$. Suppose a is a quadratic residue modulo p. Then

1. a has exactly two square roots modulo p in Z_p^*. One of these lies in $(0, s]$ and the other in $(s, p - 1]$.

2. the square roots of a modulo p can be found in probabilistic polynomial time.

Proof: By Lemma J.1.1 we know that $a \in S_p$ as defined in Section J.1; hence there exists a unique x in $(0, s]$ with $x^2 \equiv a \pmod{p}$. Then $y = p - x$ satisfies $y^2 \equiv a \pmod{p}$, and y is in $(s, p - 1]$. Conversely, suppose y is in $(s, p - 1]$ and $y^2 \equiv a \pmod{p}$. Then $x = p - y$ is in $(0, s]$, and $x^2 \equiv a \pmod{p}$. Hence y is unique. Thus we have (1).

For (2) we invoke a probabilistic polynomial-time algorithm for finding square roots modulo a prime (e.g., [PERA86] or p. 22 of [KRAN86]). ■

J.4 QUADRATIC RESIDUOSITY MODULO A PRIME

Deciding quadratic residuosity plays a role in both primality testing and probabilistic encryption. For a prime $p > 2$, to decide whether a given element $a \in Z_p^*$ is a quadratic residue modulo p, define

$$e_p(a) = a^s \mod p \qquad (s = (p - 1)/2)$$

Then we have

Lemma J.4.1: If $p > 2$ is a prime and $a \in Z_p^*$, $e_p(a) = 1$ iff a is a quadratic residue modulo p; $e_p(a) = p - 1$ iff a is a quadratic nonresidue modulo p.

Proof: By the Euler–Fermat theorem (Corollary G.2.1), $e_p(a)^2 \equiv 1 \pmod{p}$. By Lemma J.3.1, 1 has exactly two square roots modulo p in Z_p^*, namely, 1 and $p - 1$. Hence $e_p(a) = 1$ or $p - 1$. If a is a quadratic residue modulo p, then $a = x^2 \mod p$ for some x with $0 < x < p$. Then $e_p(a) = x^{p-1} \mod p$. Again by the Euler–Fermat theorem, $e_p(a) = 1$. Thus all s quadratic residues a satisfy $a^s - 1 = 0 \pmod{p}$. An sth degree congruence modulo p has at most s solutions (e.g., [RADE64], p. 21). Thus all quadratic nonresidues b must have $e_p(b) = p - 1$. ■

Also, e_p can be evaluated in $O(\log p)$ time. Thus quadratic residuosity modulo p can be decided in $O(\log p)$ time.

APPENDIX K
PRIMITIVE ROOTS AND DISCRETE LOGARITHMS

The use of primitive roots was noted in Sections 2.2 and 4.2.2. We also observed that the security of exponentiation-based cryptosystems depends in part on the difficulty of computing discrete logarithms. Here we briefly explore these topics.

Let Z_p^* be as in Appendix G. Suppose p is a prime and $a \in Z_p^*$. Suppose $m > 0$, $a^m \equiv 1 \pmod{p}$, but $a^r ! \equiv 1 \pmod{p}$ for any r with $0 < r < m$. Then we say that a belongs to the exponent m modulo p. The existence of m is guaranteed by the Euler–Fermat theorem (Corollary G.2.1), which shows $m < p$. Let

$$C_p(a) = \{a^x \mod p : 0 \le x < m\}$$

Then we have

Lemma K.1: Suppose p is prime, $a \in Z_p^*$, and a belongs to the exponent m modulo p. Then:

1. $C_p(a)$ contains distinct elements, that is, $|C_p(a)| = m$.
2. $C_p(a)$ is a cyclic subgroup of Z_p^*.
3. $m \mid p - 1$.

Proof: Suppose $C_p(a)$ does not contain distinct elements. Then for some $\{k, j\}$ in $[0, m)$ with $k < j$ we have $a^k \equiv a^j \pmod p$. Thus $a^{j-k} \equiv 1 \pmod p$, which is a contradiction. This gives (1). Now suppose x and y are in $[0, m)$ and $x + y = k * m + r$ where r is in $[0, m)$. Since $a^m \equiv 1 \pmod p$, $a^x * a^y \bmod p = a^r \bmod p$. Thus $C_p(a)$ is closed under multiplication, and forms a subgroup of Z_p^*; this is called the cyclic subgroup generated by a (e.g., [HERS64]). This gives (2). The order of a subgroup divides the order ($= p - 1$) of the whole group Z_p^* (ibid.). This gives (3). ■

If g belongs to the exponent $p - 1$ modulo prime p, g is called a primitive root modulo p.

Lemma K.2: Suppose p is a prime and g is a primitive root modulo p. Then $C_p(g) = Z_p^*$; that is, each $y \in [1, p)$ has a unique representation $y = g^x \bmod p$ for some $x \in [0, p - 1)$.

Proof: $C_p(g)$ is a subset of Z_p^*. The result follows from Lemma K.1, which shows $|C_p(g)| = p - 1 = |Z_p^*|$. ■

The x in Lemma K.2 is the discrete logarithm, or index, of y modulo p with base $= g$. However, the range of the logarithm can be any interval of length $p - 1$. We have used $[0, p - 2]$, but, for example, we could also take x to lie in $[1, p - 1]$.

A restatement of Lemma K.2 is that the cyclic subgroup generated by the primitive root g modulo p is the whole group Z_p^*. Thus g is a generator of Z_p^*.

Lemma K.3: Suppose $p > 2$ is prime and $p - 1$ has k distinct prime factors which are known. Then:

1. The number of primitive roots modulo p is $\phi(p - 1)$, where ϕ is the phi function (Appendix G.1).
2. Testing a given a to determine if it is a primitive root modulo p can be done in time $O(k * \log p) = O((\log p)^2)$.

Proof: For (1) see, for example, [RADE64], p. 49. For (2), if the exponent of a modulo p is m, then $m \mid p - 1$ by Lemma K.1. Now $m < p - 1$ iff $m \mid (p - 1)/q$ for some prime factor q of p, whence $a^{(p-1)/q} \equiv 1 \pmod p$. Thus a is a primitive root modulo p iff $a^{(p-1)/q} \not\equiv 1 \pmod p$ for each prime factor q of p. This condition can be tested for each of the k factors in $O(\log p)$ time. Clearly $k = O(\log p)$. ■

In particular, if p has the form $2q + 1$ for prime q, it is easy to find the exponent m modulo p for a given a, since $m = 2$, q, or $2q$. It is also easy to find primitive roots modulo p : $\phi(p - 1) = q - 1 = (p - 3)/2$; thus for large p, a random element of Z_p^* has about a $\frac{1}{2}$ probability of being a primitive root.

Lemma K.4: Suppose $p > 2$ is prime and g is a primitive root modulo p. Let $s = (p - 1)/2$. Then:

1. $g^s \equiv -1 \pmod{p}$.
2. g is a quadratic nonresidue modulo p.

Proof: For (1): We note $g^0 \equiv 1 \pmod{p}$. Since g is a generator of $\{1, g, \ldots, g^{p-2}\}$ and $0 < s < p - 1$ we thus know $g^s \not\equiv 1 \pmod{p}$. Also, $g^{2s} \equiv 1 \pmod{p}$, so that g^s is a square root of 1 modulo p. By Lemma J.3.1, 1 has exactly two square roots modulo p, namely, 1 and -1. Thus $g^s \equiv -1 \pmod{p}$.

For (2): Suppose $x^2 \equiv g \pmod{p}$. Now $x \equiv g^r \pmod{p}$ for some r; hence $g^{2r} \equiv g \pmod{p}$, and $g^{2r-1} \equiv 1 \pmod{p}$. Thus $2r - 1 \equiv 0 \pmod{p - 1}$, which is impossible since $p - 1$ is even. ■

APPENDIX L
PRIMALITY TESTING

Testing large integers to determine whether they are prime plays a major role in key generation in RSA and other public key systems. We give a few examples of algorithms to effect this.

It is well known (e.g., [KNUT81], p. 366) that if the number of primes between 1 and x is $f(x)$, then with ln denoting natural logarithm (to base $e = 2.718 \ldots$),

$$f(x) = x/(\ln x) + O(x/(\ln x)^2)$$

The number of primes between x and $x + h$ is $f(x + h) - f(x)$, and hence the probability that a number between x and $x + h$ is prime is roughly

$$(f(x + h) - f(x))/h = f'(x) = 1/(\ln x) - 1/(\ln x)^2$$

That is, roughly ln x numbers must be tested before a prime is found near x; but the even numbers may be ignored, so that roughly $(\ln x)/2$ odd numbers need be tested. For example, about $(\ln 10^{100})/2 = 115$ odd numbers must be tested to find a prime of about 100 digits. If multiples of small primes > 2 are eliminated, even fewer numbers need be tested before a prime is found (e.g., [GORD84, GORD84b]).

A number of tests have been given to establish the primality of a candidate. Proving primality deterministically (i.e., with certainty) is less difficult than factoring or computing discrete logarithms, but is nonetheless nontrivial. For example, the algorithms of [ADLE83] or [COHE84] could be used; but they have a runtime on the order of

$$(\log n)^{c \log \log \log n}$$

Such deterministic algorithms are computationally infeasible or inadvisable unless high-performance computers are used for key generation. This may be undesirable even

if a high-performance computer is available, since it may not constitute a cryptographically secure environment. For key generation on, for example, personal computers or smart cards, which is preferable from a security standpoint, more efficient algorithms are desirable. Thus, probabilistic tests may be used in practice to make an "educated guess" as to whether a candidate is prime.

Often a Monte Carlo approach is employed. In this event there is a slight possibility of an erroneous conclusion; however, the error probability can be made arbitrarily small. Typically, a sequence of "witnesses" attest to the primality or compositeness of a number. Agreement among a group of about 100 witnesses is generally sufficient to reach a conclusion beyond any reasonable doubt, although evidently the legality of this procedure has not been tested in the courts.

L.1 THE SOLOVAY–STRASSEN TEST

One example of a probabilistic polynomial-time primality test is the Solovay–Strassen test [SOLO77] mentioned in Section 4.1.1 (although this approach is commonly attributed to Solovay and Strassen, it was noted implicitly by Lehmer [LEHM76]; see [LENS86]). If n is an odd positive integer and $a \in Z_n^*$, with the latter as in Appendix G, let

$$e(a, n) \equiv a^{(n-1)/2} \pmod{n}, \quad -1 \leq e(a, n) \leq n - 2$$

For convenience we have $e(a, n)$ take on the value -1 instead of the usual $n - 1$.

If p is prime we have $e(a, p) \equiv e_p(a) \pmod{p}$ with e_p as in Appendix J.4. Lemma J.4.1 shows that $e(a, p) = (a/p)$, where the latter is the Jacobi symbol (Appendix J.3). The latter equality thus provides evidence of primality. That is, if p is prime and $a \in Z_n^*$ then $e(a, p) = (a/p)$. Per se this is useless, but the contrapositive provides a proof of compositeness: if $(a/n) \neq e(a, n)$ then n is composite. Also, both (a/n) and $e(a, n)$ can be computed in O(log n) steps, so this proof of compositeness runs in deterministic polynomial time. The converse is more subtle. Suppose n is an odd positive integer and $0 < a < n$. If GCD$(a, n) > 1$ then n is certainly composite. Let

$$G = \{a \in Z_n^* : (a/n) = e(a, n)\}$$

Now the Jacobi symbol is multiplicative; that is,

$$(a * b/n) = (a/n) * (b/n)$$

Also

$$e(a * b, n) \equiv e(a, n) * e(b, n) \pmod{n}$$

That is, e is also multiplicative. It follows that G is a subgroup of Z_n^*. The order of G must divide the order of Z_n^* (e.g., [HERS64], p. 35). Thus if $G \neq Z_n^*$, G has cardinality at most $(n - 1)/2$. Solovay and Strassen showed that indeed $G \neq Z_n^*$ if n is composite [SOLO77]. Hence if n is composite and a is chosen at random, the probability that a is in G is at most $\frac{1}{2}$. We thus test as follows:

```
function Solovay_Strassen (a, n) returns charstring;
  /* for n > 2, n odd, a ∈ Z_n, decides probabilistically
```

```
        whether n is prime */
      if (GCD(a, n) > 1) return("composite")
      else
        if ((a/n) = e(a, n)) return("prime")
        else return("composite");
    end Solovay_Strassen;
```

We will certainly reach a correct conclusion in any of the following cases:

1. n is prime.
2. n is composite and GCD(a, n) > 1.
3. n is composite, GCD(a, n) $= 1$, and $(a/n) \neq e(a, n)$.

In the remaining case, that is, n is composite, GCD(a, n) $= 1$, and $(a/n) = e(a, n)$, n "masquerades" as a prime due to the perjury of a. But such an a is in G above; hence the probability of false testimony from an a is at most $\frac{1}{2}$ in this case. If one hundred random a's are used as witnesses, we conclude n is prime or composite with probability of error zero if n is prime, and at most 2^{-100} otherwise. At most $6 * \log n$ operations are needed ([SOLO77]).

L.2 LEHMAN'S TEST

Lehman [LEHM82] noted that the Jacobi function is not needed in Monte Carlo testing for primality. He defines

$$e'(a, n) = a^{(n-1)/2} \mod n$$
$$G = \{e'(a, n) : a \in Z_n^*\}$$

We note that e' differs only slightly from e, taking on the value $p - 1$ instead of -1.

Lehman shows that if n is odd, $G = \{1, p - 1\}$ iff n is prime. Again one hundred a's are tested, but only $a^{(n-1)/2} \mod n$ is computed. If anything except 1 or -1 is found, n is composite. If only 1's and -1's are found, we conclude n is prime if any -1's are found; otherwise we conclude n is composite. Again the probability of error is at most 2^{-100}.

L.3 THE MILLER–RABIN TEST

Another Monte Carlo test was noted by Miller [MILL76] and Rabin [RABI80]. If $n - 1 = u * 2^k$ where u is odd and $k > 0$, let $\exp(i) = 2^i$, $0 \leq i \leq k - 1$. If $a \in Z_n^*$ then a is a witness to the compositeness of n if $a^u \not\equiv 1 \pmod n$ and $a^{u*\exp(i)} \not\equiv -1 \pmod n$, $0 \leq i \leq k - 1$. Again, for example, a hundred witnesses may be tested; if none attest to compositeness of n then n may be assumed prime.

When $n \equiv 3 \pmod 4$ this test is similar to Lehman's.

APPENDIX M
MATHEMATICS OF RSA AND OTHER EXPONENTIAL SYSTEMS

In Section 1.5 the basic exponential cipher was introduced; that is, $E(M) = M^k \bmod p$, $D(C) = C^I \bmod p$, where $K * I \equiv 1$ (modulo $p - 1$). We recall (Appendix H) that I can be computed from K using Euclid's algorithm in O(log p) time. Also, by Lemma G.1.1 we have $\phi(p) = p - 1$. Thus, by Corollary G.2.3 we have $D(E(M)) = M$ and $E(D(C)) = C$.

The RSA system is analogous: we have $E(M) = M^e \bmod n$, $D(C) = C^d \bmod n$, $n = p * q$, $e * d \equiv 1 \pmod m$, $m = (p - 1)(q - 1)$. By Lemma G.1.1 we have $\phi(n) = m$, so once again Corollary G.2.3 gives $D(E(M)) = M$ and $E(D(C)) = C$.

We have noted in Appendix L how the Solovay–Strassen or Lehman tests can be used to choose p and q efficiently, that is, in O(log n) steps. However, these involve multiple-precision arithmetic [KNUT81]. Hence the total time can be significant, especially in software.

Once p and q have been chosen, e is chosen easily; then d is computed via $e * d \equiv 1 \pmod m$. This can be done in O(log n) steps using Euclid's algorithm. Thus key material can be generated in linear time.

Encryption is easy since e can be chosen small. However, efficient decryption requires the Chinese remainder theorem. In general we have

$$(y \bmod n) \bmod p = y \bmod p$$
$$(y \bmod n) \bmod q = y \bmod q$$

Suppose we are given ciphertext C; we wish to efficiently compute

$$M = C^d \bmod n$$

To use the machinery of Appendix I, let $y = C^d$ and $x = M = y \bmod n$. Then

$$a = C^d \bmod p$$
$$b = C^d \bmod q$$

Also, suppose $d = k * (p - 1) + r$. Then by the Euler–Fermat theorem,

$$a = (C^{p-1})^k C^r \bmod p = 1^k C^r \bmod p = (C \bmod p)^r \bmod p$$

Similarly if $d = j * (q - 1) + s$,

$$b = (C \bmod q)^s \bmod q$$

Also $r = d \bmod p - 1$ and $s = d \bmod q - 1$. Thus by Corollary I.1, an algorithm for decryption [QUIS82] is as follows:

1. Compute
   a. $a = (C \bmod p)^{d \bmod (p-1)} \bmod p$
   b. $b = (C \bmod q)^{d \bmod (q-1)} \bmod q$
2. Find u with $0 < u < p$ and

$$u * q \equiv 1 \pmod p$$

3. Use one of
   a. $M = (((a - (b \mod p)) * u) \mod p) * q + b$
   $(a \geq b \mod p)$
   b. $M = (((a + p - (b \mod p)) * u) \mod p) * q + b$
   $(a < b \mod p)$

These represent roughly optimal formulae for deciphering. Again u is found easily using Euclid's algorithm, and M is easy to compute once a and b have been found. Finding a and b requires at most 2 log p and 2 log q multiplications, respectively ([KNUT81], p. 442). Thus both encryption and decryption can be done in O(log n) operations, that is, linear time. Nonetheless, these operations are relatively time-consuming (though elementary), since they involve multiplications and divisions of numbers of about 100 digits. For efficiency this requires hardware support.

APPENDIX N
QUADRATIC RESIDUOSITY MODULO A COMPOSITE

Deciding quadratic residuosity modulo a composite played a major role in Sections 8 and 9. In particular, suppose $N = p * q$ where p and q are large primes. Deciding quadratic residuosity modulo such N when p and q are unknown is regarded as computationally intractable. This forms the basis of many probabilistic encryption and zero-knowledge schemes.

Also, it was noted by Rabin [RABI79] that finding square roots modulo N in this event has essentially the same complexity as factoring N, the intractability of which forms the basis for RSA. This was exploited by Rabin (see Section 4.1.4) to obtain a modification of RSA which is provably equivalent to factoring. Essentially his scheme was to encrypt via

$$E_N(x) = x^2 \mod N.$$

Let Z_N^* be as in Appendix G.

N.1 CHARACTERIZING QUADRATIC RESIDUES

The basic extension of quadratic residuosity modulo a prime to the composite modulus case is:

Lemma N.1.1: If $N = p * q$, p and q primes > 2, then:

1. Suppose $z \in Z_N^*$. Then z is a quadratic residue modulo N iff z mod p is a quadratic residue modulo p and z mod q is a quadratic residue modulo q.
2. A quadratic residue z modulo N has exactly four square roots of the form $\{x, N - x, y, N - y\}$ in Z_N^*.
3. If z is a quadratic residue modulo N, and p and q are known, then the square roots of z modulo N can be found in probabilistic polynomial time.

Proof: Suppose $z \in Z_N^*$, z mod p is a quadratic residue modulo p, and z mod q is a quadratic residue modulo q. Then for some r and t in Z_p^* and Z_q^*, respectively, we have $r^2 \equiv z \pmod p$ and $t^2 \equiv z \pmod q$. By the Chinese remainder theorem (Appendix

I) there exists w in Z_N with $w \equiv r \pmod{p}$ and $w \equiv t \pmod{q}$. Then $w^2 \equiv z \pmod{p \text{ or } q}$, so $w^2 \equiv z \pmod{N}$. Thus z is a quadratic residue modulo N. This proves one direction of (1).

Now suppose z is a quadratic residue modulo N. Let $z_p = z \bmod p$ and $z_q = z \bmod q$. Then $z \in Z_N^*$, and hence $z_p \in Z_p^*$ and $z_q \in Z_q^*$. Also, $w^2 \equiv z \pmod{N}$ for some w in Z_N^*. Thus $w^2 \equiv z \pmod{p \text{ or } q}$ and hence $w^2 \equiv z_p \pmod{p}$ and $w^2 \equiv z_q \pmod{q}$. Thus z_p and z_q are quadratic residues modulo p and q, respectively, proving (1) in the other direction.

Furthermore, by Lemma J.3.1, z_p has exactly two square roots $\{x_1, x_2\}$ in Z_p^*, and z_q has exactly two square roots $\{y_1, y_2\}$ in Z_q^*. Hence $w \equiv x_i \pmod{p}$ and $w \equiv y_j \pmod{q}$ for some i and j. There are four possible pairs ($i = 1, 2$ and $j = 1, 2$). The Chinese remainder theorem shows that w is uniquely determined in Z_N by a given i and j; hence z has at most four square roots modulo N in Z_N^*. Conversely, by the Chinese remainder theorem once again, w can be found for each (i, j) pair. Thus z has exactly four square roots modulo N in Z_N^*. Let x denote one root. Then $N - x$ is another. If y is a third, then $N - y$ is the fourth. This proves (2).

By Lemma J.3.1, $\{x_i\}$ and $\{y_j\}$ can be found in probabilistic polynomial time; the corresponding w's can then be found in polynomial time. This proves (3). ■

Corollary N.1.1: Suppose $N = p * q$, p and q primes > 2. Then:

1. E_N is a four-to-one function.
2. If p and q are known and z is a quadratic residue modulo N, the four values of x for which $E_N(x) = z$ can be found in probabilistic polynomial time.

Proof: Immediate from the previous lemma. ■

N.2 THE JACOBI SYMBOL ONCE MORE

Suppose $N = p * q$ where p and q are primes > 2. Then for x in Z_N^*, the Jacobi symbol (x/N) is given by

$$(x/N) = (x/p)(x/q)$$

If the Jacobi symbol $(x/N) = -1$, then (x/p) or $(x/q) = -1$. Hence $x \bmod p$ and $x \bmod q$ are quadratic nonresidues modulo p and q, respectively. By Lemma N.1.1, x is a quadratic nonresidue modulo N. Since (x/N) can be evaluated in $O(\log N)$ time, $(x/N) = -1$ is a deterministic polynomial-time test for quadratic nonresiduosity of x modulo N, $N = p * q$. The interesting case is $(x/N) = 1$, whence no conclusion can be drawn regarding quadratic residuosity of x modulo N. To study this further, Z_N^* may be partitioned into four subclasses, according to the Jacobi symbols (x/p) and (x/q):

(x/p)	(x/q)	Class	(x/N)
1	1	$Q_{00}(N)$	1
−1	−1	$Q_{11}(N)$	1
1	−1	$Q_{01}(N)$	−1
−1	1	$Q_{10}(N)$	−1

Lemma N.2.1: Suppose $N = p * q$, p and q primes > 2. Then:

1. The $\{Q_{ij}(N)\}$ partition Z_N^* into disjoint classes.
2. $Q_{00}(N)$ is the set of quadratic residues modulo N; the other Q's contain nonresidues.
3. For $i = 0$ or 1 and $j = 0$ or 1, $|Q_{ij}(N)| = (p - 1)(q - 1)/4$.

Proof: (1) Is trivial, and (2) is immediate from Lemma N.1.1. For (3), let g and h be generators for Z_p^* and Z_q^*, respectively. For $w \in Z_N^*$, suppose $w \equiv g^r \pmod p$ and $w \equiv h^s \pmod q$, where $0 \leq r < p - 1$ and $0 \leq s < q - 1$. Then r and s are unique. Conversely, any such pair (r, s) uniquely determines w, by the Chinese remainder theorem. Let $f(w) = (r, s)$. Then f is a bijection from Z_N^* to $Z_p \times Z_q$. Also, if $f(w) = (r, s)$ then

$$(w/p) = (g^r/p) = (g/p)^r$$
$$(w/q) = (h^s/q) = (h/q)^s$$

By Lemma K.4, g and h are quadratic nonresidues modulo p and q, respectively. Thus $(w/p) = (-1)^r$ and $(w/q) = (-1)^s$. Let $i = r \bmod 2$ and $j = s \bmod 2$. Then $(w/p) = (-1)^i$ and $(w/q) = (-1)^j$. Hence w is in $Q_{ij}(N)$. For $k = 0, 1$ let Z_p^k and Z_q^k be the subsets of Z_p^* and Z_q^*, respectively, which contain elements $= k \pmod 2$. Then for $i = 0, 1$ and $j = 0, 1$, f is a bijection from $Q_{ij}(N)$ to $Z_p^i \times Z_q^j$ The latter has cardinality $(p - 1)(q - 1)/2$; this gives (3). ■

Thus exactly half of the elements of Z_N^* have $(x/N) = -1$; these are nonresidues modulo N. The other half have $(x/N) = 1$. Of the latter, half (i.e., $Q_{00}(N)$) are quadratic residues and half (i.e., $Q_{11}(N)$) are nonresidues. The quadratic residuosity conjecture is that determining which of the elements of Z_N^* with $(x/N) = 1$ are quadratic residues modulo N is not solvable in probabilistic polynomial time, for N a product of two primes. This is in contradistinction to the case of one prime p, for which we have noted that quadratic residuosity is decidable in deterministic polynomial time.

Lemma N.2.2: Suppose $N = p * q$, p and q primes > 2. Then:

1. For $x, y \in Z_N^*$, $x * y$ is a quadratic residue modulo N iff x and y are in the same $Q_{ij}(N)$.
2. The product of quadratic residues is a quadratic residue.
3. The product of a quadratic residue and a nonresidue is a nonresidue.

Proof: Suppose $x \in Q_{ij}(N)$ and $y \in Q_{rt}(N)$. Then $(x/p) = (-1)^i$, $(x/q) = (-1)^j$, $(y/p) = (-1)^r$, $(y/q) = (-1)^t$. Hence $(x * y/p) = (-1)^{i+r}$ and $(x * y/q) = (-1)^{j+t}$. Now $x * y \bmod N$ will be a residue iff $i = r$ and $j = t$. This gives (1). In particular, $x * y \bmod N$ is a residue if both are in $Q_{00}(N)$, yielding (2). If x is a residue and y a nonresidue, x is in $Q_{00}(N)$ but y is not; (3) follows. ■

Thus the quadratic residues modulo N form a subgroup of Z_N^*.

N.3 QUADRATIC RESIDUOSITY AND FACTORING

We note the connection between quadratic residuosity and factoring observed by Rabin [RABI79].

Lemma N.3.1: Suppose $N = p * q$, p and q prime, x and y are in Z_N^*, $x^2 \equiv y^2 \pmod N$, and $y \not\equiv x$ or $-x \pmod N$. Then possession of x and y permits factoring N in deterministic polynomial time.

Proof: We have $x - y \not\equiv 0 \pmod N$ and $x + y \not\equiv 0 \pmod N$, so $\text{GCD}(N, y + x) < N$ and $\text{GCD}(N, y - x) < N$. Now $x^2 - y^2 = (x - y)(x + y) = 0 \pmod N$. Hence $(x - y)(x + y) \equiv 0 \pmod p$. Thus $p \mid y - x$ or $p \mid y + x$; also $p \mid N$. If $p \mid y - x$, $p \mid \text{GCD}(N, y - x) < N$, so $p = \text{GCD}(N, y - x)$. The latter can be found via Euclid's algorithm (Appendix H). If $p \mid y + x$, $p \mid \text{GCD}(N, y + x) < N$, so $p = \text{GCD}(N, y + x)$. ■

Lemma N.3.2: Suppose $N = p * q$, p and q prime. Suppose there exists an algorithm to find in probabilistic polynomial time a square root of a quadratic residue modulo N, which works for at least a fraction $1/\log^c N$ of quadratic residues, c a constant positive integer. Then N can be factored in probabilistic polynomial time.

Proof: Let $m = \log^c N$. Choose $x_1, \ldots, x_m$ randomly in Z_N^* and compute $z_i = x_i^2 \bmod N$, $i = 1, \ldots, m$. If any z_i has a square root w computable in probabilistic polynomial time via the hypothetical algorithm, find w. The probability that $w \not\equiv x_i$ or $-x_i \pmod N$ is $\frac{1}{2}$. In this event possession of x_i and w factors N by Lemma N.3.1. The probability that some z_i has a computable square root is at least $\frac{1}{2}$. Thus the probability of factoring N is at least $\frac{1}{4}$. If this procedure is repeated m times, the probability of success is at least $1 - \left(\frac{3}{4}\right)^m$. Also, the problem size is $\log N$, so m is a polynomial in the problem size. ■

Corollary N.3.2: If $N = p * q$, p and q prime, and the factorization of N is computationally intractable, then the Blum encryption function E_N above is a trapdoor one-way function, with (p, q) forming the trapdoor.

Proof: By the previous theorem, an algorithm to invert E_N would yield an algorithm to factor N. ■

N.4 QUADRATIC RESIDUOSITY AND BLUM INTEGERS

Suppose $N = p * q$ where $p \equiv 3 \pmod 4$ and $q \equiv 3 \pmod 4$. Then N is called a Blum integer (normally p and q are specified to have roughly the same size). For example, $N = 77$ was used in Section 8.

Lemma N.4.1: If $p, q \equiv 3 \pmod 4$ then $(-1/p) = (-1/q) = -1$.

Proof: In Lemma J.4.1 take $a = -1$. We recall that $(-1/p) = 1$ iff -1 is a quadratic residue modulo p. Thus $(-1/p) = (-1)^{(p-1)/2} \pmod p$ and similarly $(-1/q) = (-1)^{(q-1)/2} \pmod q$. The result follows. ■

Lemma N.4.2: If $N = p * q$ is a Blum integer then $(-x/p) = -(x/p)$, $(-x/q) = -(x/q)$, $(-1/N) = 1$, and $(-x/N) = (x/N)$.

Proof: The first two are immediate from the previous lemma; also $(-1/N) = (-1/p)(-1/q)$. ■

Lemma N.4.3: Suppose $N = p * q$ is a Blum integer, x and y are in Z_N^*, $x^2 \equiv y^2 \pmod N$, and $x \not\equiv y$ or $-y \pmod N$. Then $(x/N) = -(y/N)$.

Proof: $x^2 \equiv y^2 \pmod p$, so $(y - x)(y + x) \equiv 0 \pmod p$, so $y \equiv d * x \pmod p$ where $d = 1$ or -1. Similarly $y \equiv e * x \pmod q$ where $e = 1$ or -1. Now $(x/N) = (x/p)(x/q)$ and $(y/N) = (y/p)(y/q) = (d * x/p)(e * x/q) = (d/p)(e/q)(x/N)$. Since $(1/p) = (1/q) = 1$, by Lemma N.4.1, $(y/N) = e * d * (x/N)$. If $e = d$ then $y \equiv d * x \pmod{q \text{ or } p}$, so $y \equiv d * x \pmod N$. But $y \not\equiv x$ or $-x \pmod N$, which is a contradiction. Thus $e \neq d$ and $e * d = -1$. ■

Lemma N.4.4: If $N = p * q$ is a Blum integer and z is a quadratic residue modulo N, then z has a square root in each $Q_{ij}(N)$, $i = 0, 1$, $j = 0, 1$.

Proof: Let $\{x, N - x, y, N - y\}$ be the square roots of z in Z_N^*. Since $N - x \equiv -x \pmod p$, by Lemma N.4.2, $(N - x/p) = -(x/p)$. Similarly $(N - x/q) = -(x/q)$, so if $x \in Q_{ij}(N)$, $N - x \in Q_{1-i,\,1-j}(N)$; similarly for y and $N - y$. Now $x^2 \equiv y^2 \pmod N$ and $y \not\equiv x$ or $-x \pmod N$. Since by Lemma N.4.3, $(y/N) = -(x/N)$, y cannot be in the same Q as x. By Lemma N.4.2, $((N - y)/N) = (y/N)$, so $N - y$ cannot be in the same Q as x. Thus y and $N - y$ must be in $Q_{i,\,1-j}(N)$ and $Q_{1-i,\,j}(N)$ in some order. ■

For Blum integers we can restrict the function E_N to $Q_{00}(N)$; that is, let $B_N : Q_{00}(N) \to Q_{00}(N)$ by $B_N(x) = x^2 \bmod N$. Then we have

Lemma N.4.5: If $N = p * q$ is a Blum integer then:

1. B_N is a permutation on $Q_{00}(N)$, that is, on the set of quadratic residues modulo N.
2. If the factorization of N is computationally intractable then B_N is a trapdoor one-way permutation, with (p, q) constituting the trapdoor.

Proof: By Corollary N.1.1 we know that if p and q are known and y is in $Q_{00}(N)$, the equation $E_N(x) = y$ has exactly four solutions which can be found in probabilistic polynomial time. By Lemma N.4.4, exactly one of these, x_{00}, lies in $Q_{00}(N)$; x_{00} is easily extracted from the set of four since only it satisfies $(x/p) = (x/q) = 1$. Hence x_{00} is the unique solution of $B_N(x) = y$. Thus B_N is a permutation on $Q_{00}(N)$ whose inverse may be computed in probabilistic polynomial time with knowledge of the trapdoor (p, q). On the other hand, Lemma N.3.2 shows that an algorithm to invert B_N can be converted to an algorithm to factor N. ■

Lemma N.4.6: If $N = p * q$, p and q prime, then it is possible to find x in $Q_{11}(N)$, that is, $x \in Z_N^*$ such that x is a quadratic nonresidue modulo N and $(x/N) = 1$, in probabilistic polynomial time.

Proof: Choose a in Z_p^* and evaluate $a^{(p-1)/2}$; if the latter is not 1 then choose a new a, etc. The probability of failure each time is $\frac{1}{2}$, so that the probability of not finding an a with $a^{(p-1)/2} = 1 \pmod{p}$ in n tries is 2^{-n}. Hence a can be found in probabilistic polynomial time (in n = length of p). Now $(a/p) = 1$. Similarly, find b with $(b/q) = 1$, also in probabilistic polynomial time. Use the Chinese remainder theorem to find y in Z_N^* with $y \equiv a \pmod{p}$ and $y \equiv b \pmod{q}$, in polynomial time (in length of N). ■

References

[ADLE79] L. Adleman, "A subexponential algorithm for the discrete logarithm problem with applications to cryptography," in *20th Annu. Symp. Found. Comput. Sci.*, San Juan, Puerto Rico, October 29–31, 1979, pp. 55–60. Silver Spring, MD: IEEE Computer Society Press, 1979.

[ADLE82] L. M. Adleman, "On breaking the iterated Merkle–Hellman public-key cryptosystems" 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. 303–308. New York: Plenum Press, 1983.

[ADLE86] L. M. Adleman and K. S. McCurley, "Open problems in number theoretic complexity," in *Discrete Algorithms and Complexity, Proc. Japan–U.S. Joint Seminar*, D. S. Johnson, T. Nishizeki, A. Nozaki, and H. S. Wilf, Eds., June 4–6, 1986, pp. 237–262. Orlando, FL: Academic Press, 1987.

[ADLE83] L. M. Adleman, C. Pomerance, and R. S. Rumely, "On distinguishing prime numbers from composite numbers," *Ann. Math.*, vol. 117, pp. 173–206, 1983.

[AKL-83] S. G. Akl, "Digital signatures: A tutorial survey," *Computer*, vol. 16, no. 2, pp. 15–24, Feb. 1983.

[AKL-84] S. G. Akl and H. Meijer, "A fast pseudo random permutation generator with applications to cryptology," in *Lecture Notes in Computer Science 196; Advances in Cryptology: Proc. Crypto'84*, G. R. Blakley and D. Chaum, Eds., Santa Barbara, CA, Aug. 19–22, 1984, pp. 269–275. Berlin: Springer-Verlag, 1985.

[ANSI85] American National Standard X9.17-1985, Financial Institution Key Management (Wholesale), American Bankers Association, Washington, DC, 1985.

[ANNA87] M. Annaratone, E. Arnould, T. Gross, H. T. Kung, M. Lam, O. Menzilcioglu, and J. A. Webb, "The Warp computer: Architecture, implementation and performance," *IEEE Trans. Comput.*, vol. C-36, no. 12, pp. 1523–1538, Dec. 1987.

[BANE82] S. K. Banerjee, "High speed implementation of DES," *Computers and Security*, vol. 1, no. 3, pp. 261–267, Nov. 1982.

[BART63] T. C. Bartee and D. I. Schneider, "Computation with finite fields, *Inform. Contr.*, vol. 6, no. 2, pp. 79–98, June 1963.

[BATC80] K. E. Batcher, "Design of a massively parallel processor," *IEEE Trans. Comput.*, vol. C-29, no. 9, pp. 836–840, Sept. 1980.

[BLAK84] I. F. Blake, R. Fuji-Hara, R. C. Mullin, and S. A. Vanstone, "Computing logarithms in finite fields of characteristic two," *SIAM J. Algebraic Discrete Methods*, vol. 5, no. 2, pp. 276–285, June 1984.

[BLAK84b] I. F. Blake, R. C. Mullin, and S. A. Vanstone, "Computing logarithms in GF(2^n)," in *Lecture Notes in Computer Science 196; Advances in Cryptology: Proc. Crypto'84*, G. R. Blakley and D. Chaum, Eds., Santa Barbara, CA, Aug. 19–22, 1984, pp. 73–82. Berlin: Springer-Verlag, 1985.

[BLAK83] G. R. Blakley, "A computer algorithm for calculating the product AB modulo M," *IEEE Trans. Comput.*, vol. C-32, no. 5, pp. 497–500, May 1983.

[BLUM84] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," *SIAM J. Comput.*, vol. 13, no. 4, pp. 850–864, Nov. 1984.

[BOOT81] K. S. Booth, "Authentication of signatures using public key encryption," *Commun. ACM*, vol. 24, no. 11, pp. 772–774, Nov. 1981.

[BRAS79] G. Brassard, "A note on the complexity of cryptography," *IEEE Trans. Inform. Theory*, vol. IT-25, no. 2, pp. 232–233, March 1979.

[BRAS83] G. Brassard, "Relativized cryptography," *IEEE Trans. Inform. Theory*, vol. IT-29, no. 6, pp. 877–894, Nov. 1983.

[BRAS88] G. Brassard, *Modern Cryptology: a Tutorial: Lecture Notes in Computer Science Vol. 325*. Berlin/New York: Springer-Verlag, 1988.

[BREN83] R. P. Brent and H. T. Kung, "Systolic VLSI arrays for linear-time GCD computation," in *VLSI 83, Proceedings of the IFIP TC 10/WG 10.5 International Conference on VLSI*, F. Anceau and E. J. Aas, Eds., Trondheim, Norway, August 16–19, 1983, pp. 145–154. Amsterdam/New York: North-Holland, 1983.

[BREN83b] R. P. Brent, H. T. Kung, and F. T. Luk, "Some linear-time algorithms for systolic arrays," in *IFIP Congress Series Vol. 9: Information Processing 83, Proceedings of the IFIP 9th World Congress*, R. E. A. Mason, Ed., Paris, Sept. 19–23, 1983, pp. 865–876. Amsterdam/New York: North-Holland, 1983.

[BRIC82] E. F. Brickell, "A fast modular multiplication algorithm with application to two key cryptography," 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. 51–60. New York: Plenum Press, 1983.

[BRIC83] E. F. Brickell, "Solving low density knapsacks," in *Advances in Cryptology: Proc. Crypto'83*, D. Chaum, Ed., Santa Barbara, CA, Aug. 22–24, 1983, pp. 25–37. New York: Plenum Press, 1984.

[BRIC84] E. F. Brickell, "Breaking iterated knapsacks," in *Lecture Notes in Computer Science 196; Advances in Cryptology: Proc. Crypto'84*, G. R. Blakley and D. Chaum, Eds., Santa Barbara, CA, Aug. 19–22, 1984, pp. 342–358. Berlin: Springer-Verlag, 1985.

[BRIC89] E. F. Brickell, "A survey of hardware implementations of RSA," in *Lecture Notes in Computer Science 435; Advances in Cryptology: Proc. Crypto'89*, G. Brassard, Ed., Santa Barbara, CA, Aug. 20–24, 1989, pp. 368–370. Berlin: Springer-Verlag, 1990.

[BRIC88] E. F. Brickell and A. M. Odlyzko, "Cryptanalysis: A survey of recent results," *Proc. IEEE*, vol. 76, no. 5, pp. 578–593, May 1988.

[BRIG76] H. S. Bright and R. L. Enison, "Cryptography using modular software elements," in *AFIPS Conference Proceedings Vol. 45: National Computer Conference*, S. Winkler, Ed., New York, June 7–10, 1976, pp. 113–123. Montvale, NJ: AFIPS Press, 1976.

[CCIT87] CCIT, Draft Recommendation X.509: *The Directory—Authentication Framework*. Geneva: Consultation Committee, International Telephone and Telegraph, Nov. 1987.

[CARO87] T. R. Caron and R. D. Silverman, "Parallel implementation of the quadratic scheme," *J. Supercomput.*, vol. 1, no. 3, 1987.

[COHE87] H. Cohen and A. K. Lenstra, "Implementation of a new primality test," *Math. Comput.*, vol. 48, no. 177, pp. 103–121, Jan. 1987.

[COHE84] H. Cohen and H. W. Lenstra, Jr., "Primality testing and Jacobi sums," *Math. Comput.*, vol. 42, no. 165, pp. 297–330, Jan. 1984.

[CONT80] Control Data Corporation, *CDC CYBER 200 Model 205 Computer System*. Minneapolis, MN, 1980.

[COPP84] D. Coppersmith, "Fast evaluation of logarithms in fields of characteristic two," *IEEE Trans. Inform. Theory*, vol. IT-30, no. 4, pp. 587–594, July 1984.

[COPP85] 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.

[COPP87] D. Coppersmith, "Cryptography," *IBM J. Res. and Develop.*, vol. 31, no. 2, pp. 244–248, March 1987.

[COPP86] D. Coppersmith, A. M. Odlyzko, and R. Schroeppel, "Discrete logarithms in GF(p)," *Algorithmica*, vol. 1, no. 1, pp. 1–15, 1986.

[CRAY80] Cray Research, Inc., *Cray 1-S Series Hardware Reference Manual*. Mendota Heights, MN: Cray Research, Inc., June 1980.

[CRAY85] Cray Research, Inc., *Cray X-MP Series of Computer Systems*. Mendota Heights, MN: Cray Research, Inc., 1985.

[DAVI83] D. W. Davies, "Applying the RSA digital signature to electronic mail," *Computer*, vol. 16, no. 2, pp. 55–62, Feb. 1983.

[DAVI80] J. A. Davies and W. L. Price, "The application of digital signatures based on public key cryptosystems," *NPL Report DNACS 39/80*. Teddington, Middlesex, England: National Physics Laboratory, Dec. 1980.

[DAVI83b] J. A. Davis and D. B. Holdridge, "Factorization using the quadratic sieve algorithm," in *Advances in Cryptology: Proc. Crypto'83*, D. Chaum, Ed., Santa Barbara, CA, Aug. 22–24, 1983, pp. 103–113. New York: Plenum Press, 1984.

[DAVI84] J. A. Davis, D. B. Holdridge, and G. J. Simmons, "Status report on factoring," in *Lecture Notes in Computer Science 209; Advances in Cryptology: Proc. Eurocrypt'84*, T. Beth, N. Cot, and I. Ingemarsson, Eds., Paris, France, April 9–11, 1984, pp. 183–215. Berlin: Springer-Verlag, 1985.

[DEMI83] R. DeMillo and M. Merritt, "Protocols for data security," *Computer*, vol. 16, no. 2, pp. 39–51, Feb. 1983.

[DENN79] D. E. Denning, "Secure personal computing in an insecure network," *Commun. ACM*, vol. 22, no. 8, pp. 476–482, Aug. 1979.

[DENN83b] D. E. Denning, "Protecting public keys and signature keys," *Computer,* vol. 16, no. 2, pp. 27–35, Feb. 1983.

[DENN81] D. E. Denning and G. M. Sacco, "Timestamps in key distribution protocols," *Commun. ACM,* vol. 24, no. 8, pp. 533–536, Aug. 1981.

[DENN83] D. E. R. Denning, *Cryptography and Data Security.* Reading, MA: Addison-Wesley, 1983.

[DESM83] Y. Desmedt, J. Vandewalle, and R. J. M. Govaerts, "A critical analysis of the security of knapsack public key algorithms," *IEEE Trans. Inform. Theory,* vol. IT-30, no. 4, pp. 601–611, July 1984.

[DIFF82] W. Diffie, "Conventional versus public key cryptosystems," in *Secure Communications and Asymmetric Cryptosystems,* G. J. Simmons, Ed., pp. 41–72. Boulder, CO: Westview Press, 1982.

[DIFF84] W. Diffie, "Network security problems and approaches," *Proceedings of the National Electronics Conference,* vol. 38, pp. 292–314, 1984.

[DIFF86] W. Diffie, "Communication security and national security business, technology, and politics," *Proc. National Communications Forum,* vol. 40, pp. 734–751, 1986.

[DIFF88] W. Diffie, "The first ten years of public-key cryptography," *Proc. IEEE,* vol. 76, no. 5, pp. 560–577, May 1988.

[DIFF76] W. Diffie and M. E. Hellman, "Multiuser cryptographic techniques," in *AFIPS Conference Proceedings, vol. 45: National Computer Conference,* S. Winkler, Ed., New York, June 7–10, 1976, pp. 109–112. Montvale, NJ: AFIPS Press, 1976.

[DIFF76b] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Inform. Theory,* vol. IT-22, no. 6, pp. 644–654, Nov. 1976.

[DIFF79] W. Diffie and M. E. Hellman, "Privacy and authentication: An introduction to cryptography," *Proc. IEEE, vol. 67, no. 3, pp. 397–427, March 1979.*

[DIFF87] W. Diffie, B. O'Higgins, L. Strawczynski, and D. Steer, "An ISDN secure telephone unit," *Proc. National Communication Forum,* vol. 41, no. 1, pp. 473–477, 1987.

[DIXO81] J. D. Dixon, "Asymptotically fast factorization of integers," *Math. Comput.,* vol. 36, no. 153, pp. 255–260, Jan. 1981.

[DOLE81] D. Dolev and A. C. Yao, "On the security of public key protocols," in *22nd Annual Symposium on Foundations of Computer Science,* Nashville, TN, Oct. 28–30, 1981, pp. 350–357. Silver Spring, MD: IEEE Computer Society Press, 1981.

[DUSS90] S. R. Dusse and B. S. Kaliski, Jr., "A cryptographic library for the Motorola DSP 56000," in *Lecture Notes in Computer Science, 473; Advances in Cryptology: Proc. Eurocrypt'90,* I. Damgård, Ed., May 21–24, pp. 230–244. Berlin: Springer-Verlag 1990.

[EHRS78] W. F. Ehrsam, S. M. Matyas, C. H. Meyer, and W. L. Tuchman, "A cryptographic key management scheme for implementing the Data Encryption Standard," *IBM Syst. J.,* vol. 17, no. 2, pp. 106–125, 1978.

[ELGA85] T. El Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inform. Theory,* vol. IT-31, no. 4, pp. 469-472, July 1985.

[ELGA85b] T. El Gamal, "On computing logarithms over finite fields," 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. 396–402. Berlin: Springer-Verlag, 1986.

[EVAN74] A. Evans, Jr., and W. Kantrowitz, "A user authentication scheme not requiring secrecy in the computer," *Commun. ACM*, vol. 17, no. 8, pp. 437–442, Aug. 1974.

[FEIG87] U. Feige, A. Fiat, and A. Shamir, "Zero knowledge proofs of identity," in *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, New York, May 25–27, 1987, pp. 210–217. New York: ACM, 1987.

[FIAT86] 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.

[FLYN78] R. Flynn and A. S. Campasano, "Data dependent keys for selective encryption terminal," in *Am. Fed. Inform. Process. Socs. Conference Proc., Vol 47: National Computer Conf.*, S. P. Ghosh and L. Y. Liu, Eds., Anaheim, CA, June 5–8, 1978, pp. 1127–1129. Montvale, NJ: AFIPS Press, 1978.

[GARE79] M. R. Garey and D. S. Johnson, *Computers and Intractability.* New York: W. H. Freeman, 1979.

[GILL77] J. Gill, "Computational complexity of probabilistic Turing machines," *SIAM J. Comput.*, vol. 6, no. 4, pp. 675–695, Dec. 1977.

[GOLD82] S. Goldwasser and S. Micali, "Probabilistic encryption and how to play mental poker keeping secret all partial information," in *Proc. 14th ACM Symp. Theory of Computing*, San Francisco, May 5–7, 1982, pp. 365–377. New York: Association for Computing Machinery, 1982.

[GOLD84] S. Goldwasser and S. Micali, "Probabilistic encryption," *J. Comput. Syst. Sci.*, vol. 28, no. 2, pp. 270–299, April 1984.

[GOLD89] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM J. Comput.*, vol. 18, no. 1, pp. 186–208, Feb. 1989.

[GOLD88] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM J. Comput.*, vol. 17, no. 2, pp. 281–308, Apr. 1988.

[GORD84b] J. Gordon, "Strong RSA keys," *Electron. Lett.*, vol. 20, no. 12, pp. 514–516, June 7, 1984.

[GORD84] J. A. Gordon, "Strong primes are easy to find," in *Lecture Notes in Computer Science 209; Advances in Cryptology: Proc. Eurocrypt'84*, T. Beth, N. Cot, and I. Ingemarsson, Eds., Paris, France, April 9–11, 1984, pp. 216–223. Berlin: Springer-Verlag, 1985.

[GROL88] J. Grollman and A. L. Selman, "Complexity measures for public-key cryptosystems," *SIAM J. Comput.*, vol. 17, no. 2, pp. 309–335, Apr. 1988.

[HAST88] J. Hastad, "Solving simultaneous modular equations of low degree," *SIAM J. Comput.*, vol. 17, no. 2, pp. 336–341, Apr. 1988.

[HAWK87] S. Hawkinson, "The FPS T Series: A parallel vector supercomputer," in *Multiprocessors and Array Processors*, W. J. Karplus, Ed., pp. 147–155. San Diego, CA: Simulation Councils, Inc., 1987.

[HAYE86] J. P. Hayes, T. Mudge, Q. F. Stout, S. Colley, and J. Palmer, "A microprocessor-based hypercube supercomputer," *IEEE Micro*, vol. 6, no. 5, pp. 6–17, Oct. 1986.

[HAYK88] M. E. Haykin and R. B. J. Warnar, *Smart Card Technology: New Methods for Computer Access Control*, NIST Special Publication 500-157. Washington, D.C.: National Institute of Standards and Technology, Sept. 1988.

[HENR81] P. S. Henry, "Fast decryption algorithm for the knapsack cryptographic system," *Bell Syst. Tech. J.*, vol. 60, no. 5, pp. 767–773, May–June 1981.

[HERS64] I. N. Herstein, *Topics in Algebra*. Waltham, MA: Blaisdell, 1964.

[HILL85] D. Hillis, *The Connection Machine*. Cambridge, MA: MIT Press, 1985.

[HOCK81] R. W. Hockney and C. R. Jesshope, *Parallel Computers: Architecture, Programming and Algorithms*. Bristol, England: Adam Hilger, 1981.

[HORO78] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*. Rockville, MD: Computer Science Press, 1978.

[HWAN84] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*. New York: McGraw-Hill, 1984.

[IAB-88] IAB (Internet Activities Board) Privacy Task Force, "Privacy enhancement for Internet electronic mail, Part I: Message encipherment and authentication procedures," *Request for Comments (RFC) 1040B*, Dec. 22, 1988.

[IAB88b] IAB (Internet Activities Board) Privacy Task Force, "Privacy enhancement for Internet electronic mail, Part II: Certificate-based key management," *Request for Comments (RFC) KM-RFC*, Dec. 23, 1988.

[ISO-87] International Organization for Standardization, *Draft International Standard ISO/DIS 7498-2, Information Processing Systems—Open Systems Interconnection Model, Part 2: Security Architecture*. Geneva: ISO, 1987.

[JUEN82] 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.

[JUEN86] 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, pp. 327–346. Berlin: Springer-Verlag, 1987.

[KELL89] S. S. Keller, "Comparison of data rate timings using BSAFE versus BSAFE with CYLINK," July 1989.

[KHAC79] L. G. Khachian, "A polynomial algorithm in linear programming," *Dokl. Akad. Nauk. SSSR*, vol. 244, pp. 1093–1096. English translation in *Sov. Math. Dokl.*, vol. 20, pp. 191–194.

[KLIN79] C. S. Kline and G. J. Popek, "Public key vs. conventional key encryption," in *Am. Fed. Inform. Process. Socs. Conference Proc., Vol 48: National Computer Conf.*, R. E. Merwin, Ed., New York, June 4–7, 1979, pp. 831–837. Montvale, NJ: AFIPS Press, 1979.

[KNUT81] D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms.* Reading, MA: Addison-Wesley, 1981.

[KOHN78b] L. M. Kohnfelder, "On the signature reblocking problem in public-key cryptosystems," *Commun. ACM,* vol. 21, no. 2, p. 179, Feb. 1978.

[KOHN78] L. M. Kohnfelder, "A method for certification," *MIT Laboratory for Computer Science,* Cambridge, MA: MIT Press, May 1978.

[KONH81] A. G. Konheim, *Cryptography: A Primer.* New York: John Wiley & Sons, 1981.

[KOWA85] J. Kowalik, Ed., *Parallel MIMD Computation: The HEP supercomputer and Its Applications.* Cambridge, MA: MIT Press, 1985.

[KRAN86] E. Kranakis, *Primality and Cryptography.* Chichester/New York: John Wiley & Sons, 1986.

[KUNG82] H. T. Kung, "Why systolic architectures," *Computer,* vol. 15, no. 1, pp. 37–46, Jan. 1982.

[KUNG78] H. T. Kung and C. Leiserson, "Systolic arrays (for VLSI)," in *Sparse Matrix Proceedings,* I. S. Duff and G. W. Stewart, Eds., pp. 245–282. Philadelphia: SIAM, 1978.

[KUNG82b] S. Y. Kung, K. S. Arun, R. J. Gal-Ezer, and D. V. B. Rao, "Wavefront array processor: Language, architecture, and applications," IEEE Trans. Comput., vol. C-31, no. 11, pp. 1054–1066, Nov. 1982.

[LAGA84] J. C. Lagarias, "Performance analysis of Shamir's attack on the basic Merkle-Hellman knapsack system," in *Lecture Notes in Computer Science, Vol. 172: Automata, Languages and Programming: 11th Colloquium,* J. Paredaens, Ed., Antwerp, Belgium, July 16–20, 1984, pp. 312–323. Berlin/New York: Springer-Verlag, 1984.

[LAGA83] J. C. Lagarias and A. M. Odlyzko, "Solving low-density subset sum problems," in *24th Annual Symposium on Foundations of Computer Science,* Tucson, AZ, November 7–9, 1983, pp. 1–10. Silver Spring, MD: IEEE Computer Society Press, 1983. Revised version in *J. ACM,* vol. 32, no. 1, pp. 229–246, Jan. 1985.

[LAKS83] S. Lakshmivarahan, "Algorithms for public key cryptosystems: Theory and application," *Advances in Computers,* vol. 22, pp. 45–108, 1983.

[LAWS71] B. A. Laws, Jr., and C. K. Rushforth, "A cellular-array multiplier for GF(2^m)," *IEEE Trans. Comput.,* vol. 20, no. 12, pp. 1573–1578, Dec. 1971.

[LEHM82] D. J. Lehmann, "On primality tests," *SIAM J. Comput.,* vol. 11, no. 2, pp. 374–375, May 1982.

[LEHM76] D. H. Lehmer, "Strong Carmichael numbers," *J. Austr. Math. Soc.,* vol. 21 (Ser. A), pp. 508–510, 1976.

[LEMP79] A. Lempel, "Cryptology in transition," *ACM Comput. Surveys,* vol. 11, no. 4, pp. 285–303, Dec. 1979.

[LENS82] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovasz, "Factoring polynomials with rational coefficients," *Mathematische Annalen,* vol. 261, pp. 515–534, 1982.

[LENS90] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard, "The number field sieve," *Proc. 22nd Annual ACM Symp. Theory of Computing,* Baltimore, May 14–16, 1990, pp. 564–572. New York: Association for Computing Machinery, 1990.

[LENS89] 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.

[LENS83] H. W. Lenstra, Jr., "Interger programming with a fixed number of variables," *Math. Oper. Res.*, vol. 8, no. 4, pp. 538–548, Nov. 1983

[LENS86] H. W. Lenstra, Jr., "Primality testing," in *Mathematics and Computer Science, CWI Monographs* Vol. I (CWI Symp., Nov. 19, 1983), J. W. de Bakker et al., Eds., pp. 269–287. Amsterdam/New York: North Holland, 1986.

[LENS87] H. W. Lenstra, Jr., "Factoring integers with elliptic curves," *Ann. Math.*, vol. 126, pp. 649–673, 1987.

[LEWI81] H. R. Lewis and C. H. Papadimitriou, *Elements of the Theory of Computation.* Englewood Cliffs, NJ: Prentice Hall, 1981.

[LINN89] J. Linn and S. T. Kent, "Privacy for DARPA-Internet mail," in *Proc. 12th Nat. Computer Security Conf.*, Baltimore, MD, pp. 215–229, October 1989.

[MACM81] D. MacMillan, "Single chip encrypts data at 14 Mb/s," *Electronics,* vol. 54, no. 12, pp. 161–165, June 16, 1981.

[MASS88] J. L. Massey, "An introduction to contemporary cryptology," *Proc. IEEE,* vol. 76, no. 5, pp. 533–549, May 1988.

[MATY78] S. M. Matyas and C. H. Meyer, "Generation, distribution, and installation of cryptographic keys," *IBM Syst. J.*, vol. 17, no. 2, pp. 126–137, 1978.

[MCCU89] K. S. McCurley, "The discrete logarithm problem," in *AMS Proc. Symp. Appl. Math., Vol. 42: Cryptology and Computational Number Theory,* C. Pomerance, Ed., pp. 49–74. Providence, RI: American Mathematical Society, 1991.

[MCEL78] R. J. McEliece, "A public-key cryptosystem based on algebraic coding theory," *DSN Progress Report 42–44,* Jet Propulsion Laboratory, Pasadena, CA, pp. 114–116, 1978.

[MERK78] R. C. Merkle, "Secure communications over insecure channels," *Commun. ACM,* vol. 21, no. 4, pp. 294–299, Apr. 1978.

[MERK82] R. C. Merkle, *Secrecy, Authentication, and Public Key Systems.* Ann Arbor: UMI Research Press, 1982.

[MERK82b] R. C. Merkle, "Protocols for public key cryptosystems," in *Secure Communications and Asymmetric Cryptosystems,* G. J. Simmons, Ed., pp. 73–104. Boulder, CO: Westview Press, 1982.

[MERK89] 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.

[MERK78b] R. C. Merkle and M. E. Hellman, "Hiding information and signatures in trapdoor knapsacks," *IEEE Trans. Inform. Theory,* vol. 24, no. 5, pp. 525–530, Sept. 1978.

[MILL76] G. L. Miller, "Riemann's hypothesis and tests for primality," *J. Comput. Syst. Sci.*, vol. 13, no. 3, pp. 300–317, Dec. 1976.

[MILU88] V. M. Milutinovic, *Computer Architecture: Concepts and Systems.* New York: North-Holland, 1988.

[MONT85] P. L. Montgomery, "Modular multiplication without trial division," *Math. Comput.*, vol. 44, no. 170, pp. 519–521, Apr. 1985.

[MOOR88] J. H. Moore, "Protocol failures in cryptosystems," *Proc. IEEE,* vol. 76, no. 5, pp. 594–602, May 1988.

[MORR75] M. A. Morrison and J. Brillhart, "A method of factoring and the factorization of F_7," *Math. Comput.*, vol. 29, no. 129, pp. 183–205, Jan. 1975.

[NATI77] National Bureau of Standards, *Federal Information Processing Standards Publication 46: Data Encryption Standard,* Washington, D.C.: U.S. Dept. Commerce, Jan. 15, 1977.

[NATI80] National Bureau of Standards, *Federal Information Processing Standards Publication 81: DES Modes of Operation,* Washington, D.C.: U.S. Dept. Commerce, Dec. 2, 1980.

[NATI81] National Bureau of Standards, *Federal Information Processing Standards Publication 74: Guidelines for Implementing and Using the NBS Data Encryption Standard,* Washington, D.C.: U.S. Dept. Commerce, Apr. 1, 1981.

[NEED78] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Commun. ACM,* vol. 21, no. 12, pp. 993–999, Dec. 1978.

[ODLY84] A. M. Odlyzko, "Cryptanalytic attacks on the multiplicative knapsack cryptosystem and on Shamir's fast signature scheme," *IEEE Trans. Inform. Theory,* vol. IT-30, no. 4, pp. 594–601, July 1984.

[ODLY84b] A. M. Odlyzko, "Discrete logarithms in finite fields and their cryptographic significance," in *Lecture Notes in Computer Science 209; Advances in Cryptology: Proc. Eurocrypt'84,* T. Beth, N. Cot, and I. Ingemarsson, Eds., Paris, France, April 9–11, 1984, pp. 224–314. Berlin: Springer-Verlag, 1985.

[ORTO86] G. A. Orton, M. P. Roy, P. A. Scott, L. E. Peppard, and S. E. Tavares, "VLSI implementation of public-key encryption algorithms," 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. 277–301. Berlin: Springer-Verlag, 1987.

[PATT87] W. Patterson, *Mathematical Cryptology for Computer Scientists and Mathematicians.* Totowa, NJ: Rowman & Littlefield, 1987.

[PERA86] R. C. Peralta, "A simple and fast probabilistic algorithm for computing square roots modulo a prime number," *IEEE Trans. Inform. Theory,* vol. 32, no. 6, pp. 846–847, Nov. 1986.

[POHL78] 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, no. 1, pp. 106–110, Jan. 1978.

[POME84] C. Pomerance, "The quadratic sieve factoring algorithm," in *Lecture Notes in Computer Science 209; Advances in Cryptology: Proc. Eurocrypt'84,* T. Beth, N. Cot, and I. Ingemarsson, Eds., Paris, France, April 9–11, 1984, pp. 169–182. Berlin: Springer-Verlag, 1985.

[POME86] C. Pomerance, "Fast, rigorous factorization and discrete logarithm algorithms," in *Discrete Algorithms and Complexity, Proceedings of the Japan-US Joint Seminar,* D. S. Johnson, T. Nishizeki, A. Nozaki, and H. S. Wilf, Eds., Kyoto, June 4–6, 1986, pp. 119–143. Orlando, FL: Academic Press, 1987.

[POME88] C. Pomerance, J. W. Smith, and R. Tuler, "A pipeline architecture for factoring large integers with the quadratic sieve algorithm," *SIAM J. Comput.*, vol. 17, no. 2, pp. 387–403, Apr. 1988.

[POPE78] G. J. Popek and C. S. Kline, "Encryption protocols, public key algorithms and digital signatures in computer networks," in *Foundations of Secure Computation,* R. A. DeMillo, D. P. Dobkin, A. L. Jones, and R. J. Lipton, Eds., pp. 133–153. New York: Academic Press, 1978.

[POPE79] G. L. Popek and C. S. Kline, "Encryption and secure computer networks," *ACM Comput. Surveys,* vol. 11, no. 4, pp. 331–356, Dec. 1979.

[QUIN87] M. J. Quinn, *Designing Efficient Algorithms for Parallel Computers.* New York: McGraw-Hill, 1987.

[QUIS82] J.-J. Quisquater and C. Couvreur, "Fast decipherment algorithm for RSA public-key cryptosystem," *Electron. Lett.*, vol. 18, no. 21, pp. 905–907, Oct. 14, 1982.

[RABI76] M. O. Rabin, "Probabilistic algorithms," in *Algorithms and Complexity: New Directions and Recent Results, Proceedings of a Symposium,* J. F. Traub, Ed., Pittsburgh, PA, April 7–9, 1976, pp. 21–39. New York: Academic Press, 1976.

[RABI78] M. O. Rabin, "Digitalized signatures," in *Foundations of Secure Computation,* R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton, Eds., pp. 155–168. New York: Academic Press, 1978.

[RABI79] M. O. Rabin, *Digitalized Signatures and Public-Key Functions as intractable as factorization,* Cambridge, MA: MIT Laboratory for Computer Science, Tech. Rept. LCS/TR-212, Jan. 1979.

[RABI80] M. O. Rabin, "Probabilistic algorithms for testing primality," *J. Number Theory,* vol. 12, pp. 128–138, 1980.

[RADE64] H. Rademacher, *Lectures on Elementary Number Theory.* New York: Blaisdell, 1964.

[RIVE84] R. L. Rivest, "RSA chips (past/present/future)," in *Lecture Notes in Computer Science 209; Advances in Cryptology: Proc. Eurocrypt'84,* T. Beth, N. Cot, and I. Ingemarsson, Eds., Paris, France, April 9–11, 1984, pp. 159–165. Berlin: Springer-Verlag, 1985.

[RIVE90] R. L. Rivest, "The MD4 message digest algorithm," in *Lecture Notes in Computer Science 537; Advances in Cryptology*: Proc. Crypto'90. A. J. Menezes and S. A. Vanstone, Eds. Santa Barbara, CA, Aug. 11–15, 1990, pp. 303–311. Berlin: Springer-Verlag, 1991.

[RIVE78] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital structures and public-key cryptosystems," *Commun. ACM,* vol. 21, no. 2, pp. 120–127, Feb. 1978.

[RIVE82] R. L. Rivest and A. T. Sherman, "Randomized encryption techniques," 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. 23–25. New York: Plenum Press, 1983.

[SIAM90] "Number field sieve produces factoring breakthrough," *SIAM News*, vol. 23, no. 4, July 1990.

[SALO85] A. Salomaa, "Cryptography," in *Encyclopedia of Mathematics and its Applications, Vol. 25: Computation and Automata*, A. Salomaa, pp. 186–230. Cambridge, United Kingdom: Cambridge University Press, 1985.

[SCHA82] B. P. Schanning, "Applying public key distribution to local area networks," *Computers and Security*, vol. 1, no. 3, pp. 268–274, Nov. 1982.

[SCHA80] B. P. Schanning, S. A. Powers, and J. Kowalchuk, "MEMO: Privacy and authentication for the automated office," in *Proceedings of the 5th Conference on Local Computer Networks, Minneapolis, MN, Oct. 6–7, 1980*, pp. 21–30. Silver Spring, MD: IEEE Computer Society Press, 1980.

[SCHN84] C. P. Schnorr and H. W. Lenstra, Jr., "A Monte Carlo factoring algorithm with linear storage," *Math. Comput.*, vol. 43, no. 167, pp. 289–311, July 1984.

[SEDL87] H. Sedlak, "The RSA cryptography processor," 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. 95–105. Berlin: Springer-Verlag, 1988.

[SEIT85] C. L. Seitz, "The Cosmic Cube," *Commun. ACM*, vol. 28, no. 1, pp. 22–33, Jan. 1985.

[SHAM79] A. Shamir, "On the cryptocomplexity of knapsack systems," in *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing, Atlanta, GA, April 30–May 2, 1979*, pp. 118–129. New York: ACM, 1979.

[SHAM84b] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Lecture Notes in Computer Science 196; Advances in Cryptology: Proc. Crypto'84*, G. R. Blakley and D. Chaum, Eds., Santa Barbara, CA, Aug. 19–22, 1984, pp. 47–53. Berlin: Springer-Verlag, 1985.

[SHAM84] A. Shamir, "A polynomial-time algorithm for breaking the basic Merkle-Hellman cryptosystems," *IEEE Trans. Inform. Theory*, vol. IT-30, no. 5, pp. 699–704, Sept. 1984.

[SHAN90] M. Shand, P. Bertin, and J. Vuillemin, "Hardware speedups in long integer multiplication," in *Proc. 2nd ACM Symposium on Parallel Algorithms and Architectures*, Island of Crete, Greece, July 2–6, 1990, pp. 138–145. New York: Association for Computing Machinery, 1991.

[SILV87] R. D. Silverman, "The multiple polynomial quadratic sieve," *Math. Comput.*, vol. 48, no. 177, pp. 329–339, Jan. 1987.

[SIMM79] G. J. Simmons, "Symmetric and asymmetric encryption," *ACM Comput. Surveys*, vol. 11, no. 4, pp. 305–330, Dec. 1979.

[SIMM88] G. J. Simmons, "How to ensure that data acquired to verify treaty compliance are trustworthy," *Proc. IEEE*, vol. 76, no. 5, pp. 621–627, May 1988.

[SMID81] M. E. Smid, "Integrating the data encryption standard into computer networks," *IEEE Trans. Comput.*, vol. COM-29, no. 6, pp. 762–772, June 1981.

[SIMM88b] M. E. Smid and D. K. Branstad, "The Data Encryption Standard: Past and future," *Proc. IEEE*, vol. 76, no. 5, pp. 550–559, May 1988.

[SOLO77] R. Solovay and V. Strassen, "A fast Monte-Carlo test for primality," *SIAM J. Comput.*, vol. 6, no. 1, pp. 84–85, March 1977. Erratum: *Ibid.*, vol. 7, no. 1, p. 118, Feb. 1978.

[STEI86] L. K. Steiner, "The ETA-10 supercomputer system," in *Proceedings of the 1986 IEEE International Conference on Computer Design, Port Chester, NY, Oct. 6–9,* p. 42. Washington, DC: IEEE Computer Society Press, 1986.

[TANE81] A. S. Tanenbaum, *Computer Networks.* Englewood Cliffs, NJ: Prentice Hall, 1981.

[WAGN84] N. R. Wagner and M. R. Magyarik, "A public-key cryptosystem based on the word problem," in *Lecture Notes in Computer Science 196; Advances in Cryptology: Proc. Crypto'84,* G. R. Blakley and D. Chaum, Eds., Santa Barbara, CA, Aug. 19–22, 1984, pp. 19–36. Berlin: Springer-Verlag, 1985.

[WANG85] C. C. Wang, T. K. Truong, H. M. Shao, L. J. Deutsch, J. K. Omura, and I. S. Reed, "VLSI architectures for computing multiplications and inverses in GF(2^m)," *IEEE Trans. Comput.*, vol. C-34, no. 8, pp. 709–717, Aug. 1985.

[WILK68] M. V. Wilkes, *Time-Sharing Computer Systems.* New York: Elsevier, 1968.

[WILL80] H. C. Williams, "A modification of the RSA public-key encryption procedure," *IEEE Trans. Inform. Theory,* vol. IT-26, no. 6, pp. 726–729, Nov. 1980.

[WILL87] H. C. Williams and M. C. Wunderlich, "On the parallel generation of the residues for the continued fraction factoring algorithm," *Math. Comput.*, vol. 48, no. 177, pp. 405–423, Jan. 1987.

[WUND83] M. C. Wunderlich, "Factoring numbers on the Massively Parallel computer," in *Advances in Cryptology: Proc. Crypto'83,* D. Chaum, Ed., Santa Barbara, CA, Aug. 22–24, 1983, pp. 87–102. New York: Plenum Press, 1984.

[WUND85] M. C. Wunderlich, "Implementing the continued fraction factoring algorithm on parallel machines," in *Math. Comput.*, vol. 44, no. 169, pp. 251–260, Jan. 1985.

[YAO-82] A. C. Yao, "Theory and applications of trapdoor functions," in *Proceedings of 23rd Ann. IEEE Symposium Foundations Computer Science,* Chicago, Nov. 3–5, 1982, pp. 80–91. Los Angeles: IEEE Computer Society Press, 1982.