PUBLIC-KEY CRYPTOGRAPHY

NIST Special Publication 800-2

James Nechvatal
Security Technology Group
National Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

April 1991

PREFACE

This publication 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:

1. The theory of public-key cryptography.
2. Comparisons to conventional (secret-key) cryptography.
3. A largely self-contained summary of relevant mathematics.
4. A survey of major existing public-key systems.
5. An exploration of digital signatures and hash functions.
6. A survey of public-key implementations in networks.
7. An introduction to zero-knowledge protocols and probabilistic encryption.
8. An exploration of security issues and key sizes.

The treatment of public-key cryptography in this publication includes both theory and practice. Much of the existing published work, including those documents listed in the references, treats either the theory or specific systems/implementations, but not both. The viewpoint here is that the theory and practice are inseparable.

Any mention of commercial products is for purposes of explanation and illustration only. Also, the selection of cryptosystems and hash functions mentioned in this publication serve only to provide examples. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that systems or functions identified are necessarily the best available for the purpose.

The focus is on issues such as criteria for systems and

protocols for usage. These are presumably long-term, in contrast, to the set of existing public-key systems which is more volatile. Thus we provide information which 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 fund transfer systems.

The core of this exposition is sections 1 to 5. Sections 1 to 3 cover the fundamentals of public-key cryptography and the related topics of hash functions and digital signatures. Extensive coverage of key management is also included, with a focus on certificate-based management. 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. The major example is the International Organization for Standardization (ISO) authentication framework.

Section 6 gives a sample proposal for a local-area network implementation of public-key cryptography. It draws heavily on the work of ISO.

A variety of topics are covered in the appendices, including a summary of relevant mathematics and algorithms. Also included is a brief introduction to zero-knowledge protocols, probabilistic encryption and identity-based public-key systems.

In the following, letters refer to appendices; e.g. lemma G.2.1 refers to a lemma appearing in section 2 of appendix G.

The author wishes to thank Dr. Ronald L. Rivest, Dr. Gustavus Simmons, and Dr. Dennis Branstad for providing many comments and suggestions, and Dr. Burton S. Kaliski Jr. for providing information on implementations of the RSA public-key system. The paper was edited by Miles Smid.

This paper was supported in part by the United States Department of Computer-Aided Logistics Supports, Department of Defense.

CONTENTS

LIST OF ILLUSTRATIONS

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; i.e., 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 which may be provided is nonrepudiation of origin, i.e., protection against a sender of a message later denying transmission. Variants of these services, and other services, are discussed in [ISO-87].

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

a. Authenticity as a consideration which rivals and sometimes exceeds secrecy in importance.

b. 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 which 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, i.e., using public keys; providing secrecy; and providing authenticity. Unfortunately no single technique proposed to date has met all three criteria. Conventional systems such as 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 which can meet all of the above goals: secrecy, authenticity and ease of key management.

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

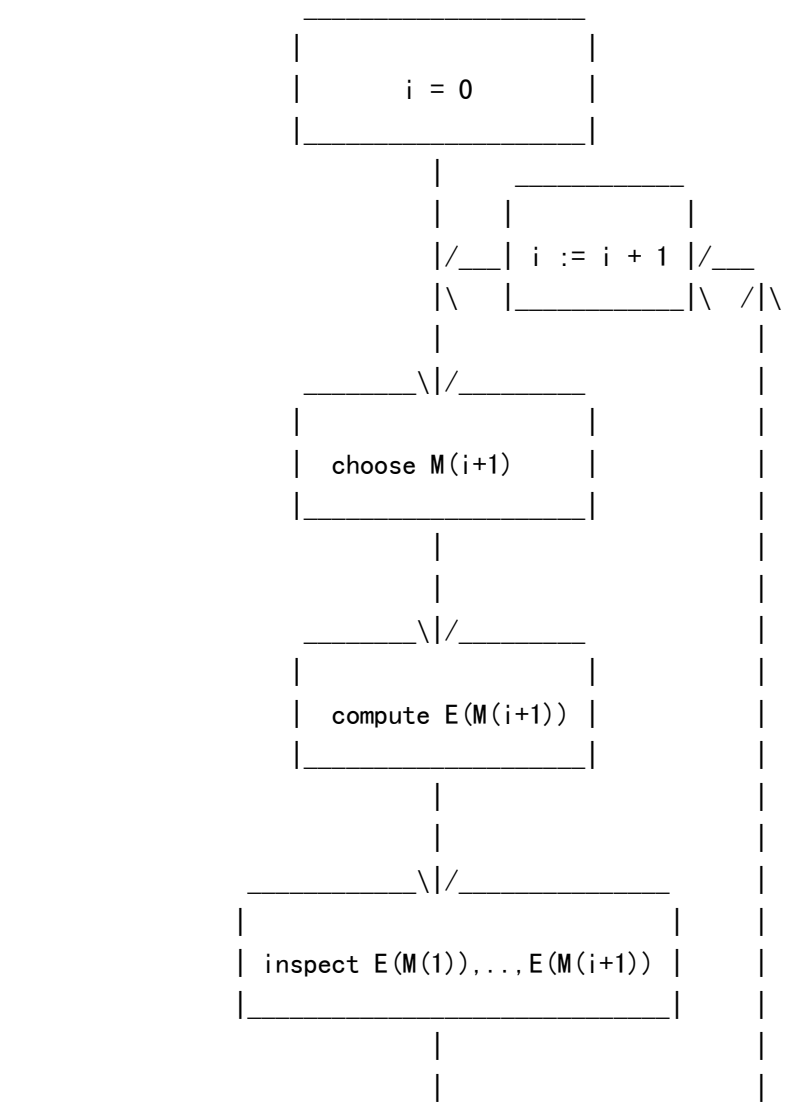
1.1 Requirements for secrecy.

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:

a. A cryptanalyst should not be able to determine M from E(M); i.e., the cryptosystem should be immune to ciphertext-only attacks.

b. A cryptanalyst should not be able to determine D given {E(Mi)} for any sequence of plaintexts {M1,M2,...}; i.e. the cryptosystem should be immune to known-plaintext attacks. This should remain true even when the cryptanalyst can choose {Mi} (chosen-plaintext attack), including the case in which the cryptanalyst can inspect {E(M1),...,E(Mj)} before specifying Mj+1 (adaptive chosen-plaintext attack).

Adaptive chosen plaintext attack is illustrated below.



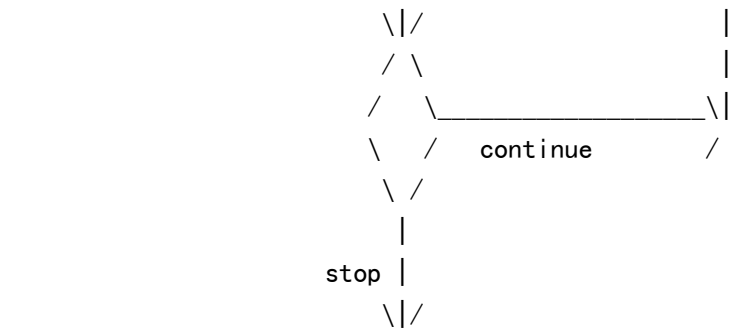


Figure 1. Adaptive Chosen Plaintext Attack

To illustrate the difference between these two categories, we use two examples. First, suppose E(M) = M3 mod N, N = p * q, where p and q are large secret primes. Then (cf. sec. 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 (a), i.e., vulnerability to ciphertext-only attack, can frequently be corrected by slight modifications of the encoding scheme, as in the M3 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:

a. It should be possible for the recipient of a message to ascertain its origin.

b. 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; i.e., we may add a third requirement if desired:

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

We might also wish to add:

d. 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 DK(EK(M)) = M. It is often the case that the algorithms for obtaining DK and EK from K are public, although both EK and DK 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 which are both private (since an eavesdropper cannot compute DK(C)) and authenticated (since a would-be masquerader cannot compute EK(M)). In some cases (e.g., transmission of a random bit string), this does not assure integrity; i.e., 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 (Data Encryption Standard). 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 which 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/decryption. Both permutations and table lookups are easily implemented, especially in hardware. Thus encryption rates exceeding 40 Mbit/sec have been obtained (e.g., [BANE82],[MACM81]). This makes DES an efficient bulk encryptor, especially when implemented in hardware.

The security of DES is produced in classical 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 {Ki} 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 Mi denotes the output of stage i, we have

L(Mi) = R(Mi-1),

R(Mi) = L(Mi-1) XOR f(L(Mi),Ki).

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

1.5 Another conventional cipher: exponentiation.

Pohlig and Hellman [POHL78] noted a type of cipher which deviated from the classical methods such as transposition and substitution. Their technique was conceptually much simpler. Let GCD denote greatest common divisor (app. G). Suppose $p > 2$ is a prime and suppose K is a key in [1,p-1) with 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

$$E(M) = M^K \bmod p.$$

Now the condition GCD(K,p-1) = 1 implies (lem. 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 (app. H). We may set

$$D(C) = C^I \bmod p.$$

It may then be shown (cor. G.2.3) that D(E(M)) = M, as required. Furthermore, E(D(C)) = C as well; i.e., 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 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 non-trivial. 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 = MK mod p for some K. He should still be unable to find K; i.e., 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).

Computing modulo p is equivalent to using the Galois field GF(p); it is possible to use GF(pn) instead (app. G). There are both advantages and disadvantages to the extension. Arithmetic in GF(pn) is generally easier if n > 1, and especially so in GF(2n). On the other hand, taking discrete logarithms in GF(pn) 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(2n) can be computed in time

exp(c * n1/3 * (log n)2/3).

For a survey of the discrete logarithm problem see, e.g., [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 [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 sec. 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 EK and DK are parameterized by a single key K. Hence if EK is known then it may be assumed that K has been compromised, whence it may also be assumed that DK is also known, or vice-versa. For example, in DES, both E and D are computed by essentially 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 EB, the public transformation of B (note that subscripts refer to users rather than keys in this

context). Now A encrypts M via C = EB(M) and sends C to B. On receipt, B employs his private transformation DB for decryption; i.e., B computes DB(C) = DB(EB(M)) = M. If A's transmission is overheard, the intruder cannot decrypt C since DB is private. Thus secrecy is ensured. However, presumably anyone can access EB; 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/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 DA to compute C = DA(M) and send C to B. Now B can use A's public transformation EA to find EA(C) = EA(DA(M)) = M. 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 EA: if a cryptanalyst, starting with a message M, could find C' such that EA(C') = M, this would imply that he can invert EA, 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 EA(C). Actually, however, a slightly more complex procedure is generally employed: an auxiliary public function H is used to produce a digital signature S = DA(H(M)) which A sends to B along with M. On receipt B can compute H(M) directly. The latter may be checked against EA(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 EA. 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 = DA(M) presumably has access to EA and hence can compute M = EA(C). Thus in either case M is accessible to an eavesdropper.

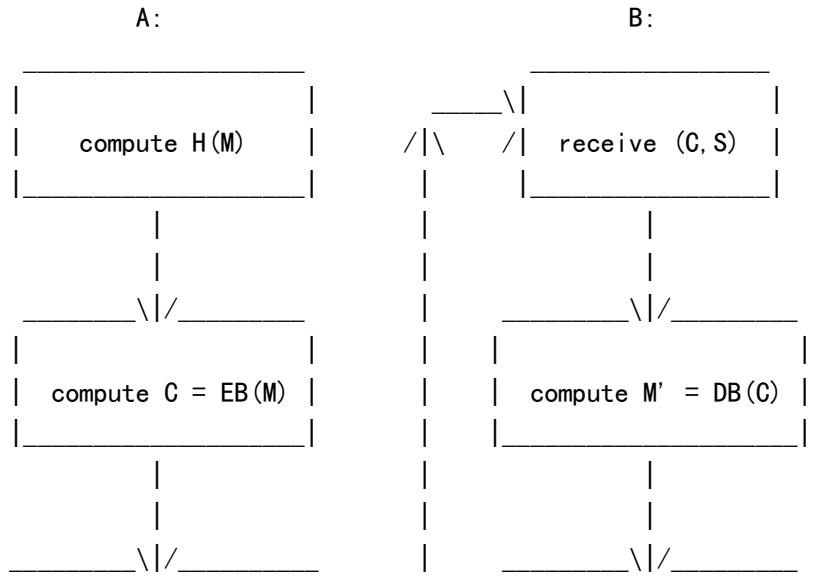
It may be necessary to use a combination of systems to provide secrecy, authenticity and integrity. However, in some cases it is possible to employ the same public-key system for these services simultaneously. We note that for authenticity/integrity purposes, D is applied to M or H(M); for secrecy, E is applied to M. 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; i.e., 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 EA, DA, EB, and DB all have a common domain. Then both secrecy and authenticity can be accomplished with a single transmission: A sends C = EB(DA(M)) to B; then B computes EA(DB(C)) = EA(DA(M)) = M. An intruder cannot decrypt C since he lacks DB; hence secrecy is assured. If the intruder sends C' instead of C, C' cannot produce a valid M since DA is needed to produce a valid C. This assures authenticity.

In actuality there are no common systems versatile enough for the last usage without modification. In fact there is only one major system (RSA) which 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.

If the question of domains is ignored, the usage of a system satisfying E(D(M)) = D(E(M)) = M with a hash function H is illustrated below. There EA,DA,EB,DB are the public transformations of A and B, respectively.



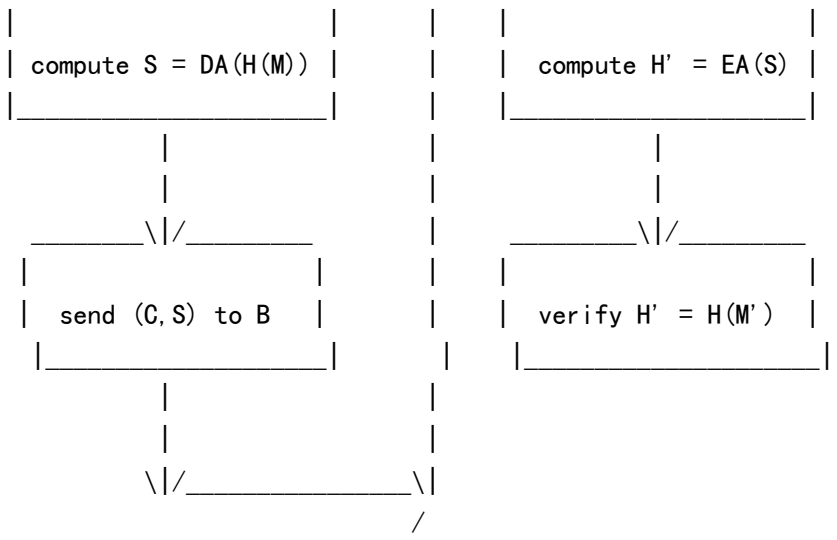


Figure 2. Using Public-Key for Secrecy and Authenticity

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 which 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:

a. Distribution of secret keys.

b. Digital signatures.

The first involves using public-key systems for secure and authenticated exchange of data-encrypting keys between two parties (sec. 2). Data-encrypting keys 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 which 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 (sec. 3). They provide authentication, nonrepudiation and integrity checks. As noted earlier, 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, i.e., when A sent DA(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 (sec. 3). In both cases the data to be encrypted or decrypted is 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: in order 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; e.g., 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 which 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 [BRAN75]. 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 on-line. In large networks it might also

become a bottleneck, since each pair of users needing a key must access a central node at least once. 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 highly unlikely in large networks. 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 may not be robust: failure of the central authority could disrupt 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.

Another alternative is a key translation center which requires only one key-encrypting key exchange through a certification authority.

Some of these disadvantages can also be mitigated by a public-key approach to key distribution.

2.2 Public distribution of secret keys.

It was noted by 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 upon a common prime p and a common primitive root g modulo p (app. K). Then (lem. K.2) each number in [1,p) can be expressed as gx mod p for some x. Both p and g may be public.

To establish a common secret key, A chooses a random x(A) in [0,p-1] and B chooses a random x(B) in [0,p-1]; these are kept secret. Then A computes

$$y(A) = gx(A) \bmod p$$

while B computes

$$y(B) = gx(B) \bmod p.$$

These need not be secret. Finally A sends y(A) to B and B sends y(B) to A. Now A knows x(A) and y(B) and can compute

$$K = y(B)x(A) \bmod p = gx(B)x(A).$$

Similarly, B knows x(B) and y(A) and can compute

$$K = y(A)x(B) \bmod p = gx(A)x(B).$$

The Diffie/Hellman algorithm is illustrated below.

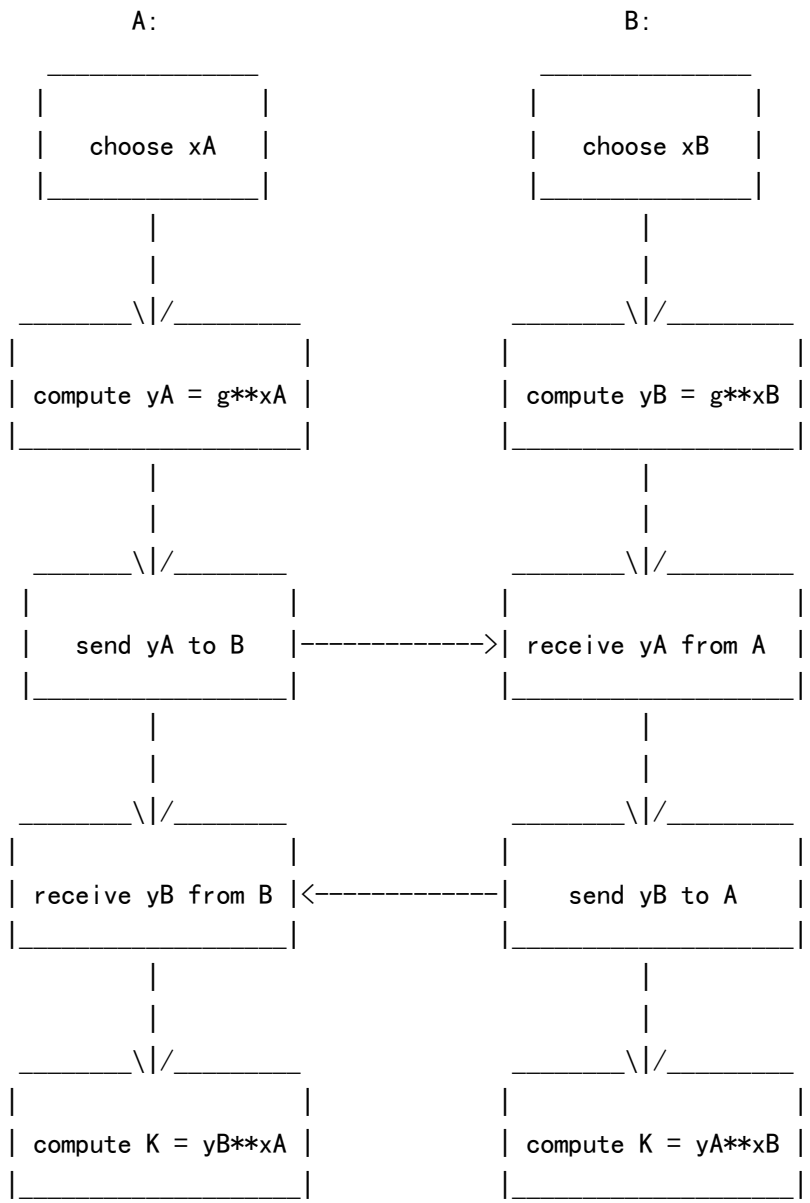

A:
B:
choose xA
choose xB
compute yA = g**xA
compute yB = g**xB
send yA to B
receive yA from A
receive yB from B
send yB to A
compute K = yB**xA
compute K = yA**xB

Figure 3. The Diffie/Hellman Key Exchange

This establishes K as a secret common quantity which can be used in some agreed-upon fashion to generate a secret key. 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 the fact that y = gx mod p for some x does not yield x. Thus, if an intruder knows p and g and intercepts y(A) or y(B) or both, he cannot find x(A) or x(B) and hence cannot compute K.

A disadvantage of this scheme is lack of support for authenticity. If an intruder C intercepts y(B), sends y(C) = gx(C) mod p to B and B thinks he is receiving y(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 which 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 (EA and EB in sec. 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, e.g., be managed by an on-line or off-line directory service; they can also be exchanged directly by users. However, authenticity is an issue as in the previous section. If A thinks that EC is really EB then A might encrypt using EC and inadvertently allow C to decrypt using DC. 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 on-line; 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 CA as in the secret-key case. Again it must be assumed that each user can communicate securely with the CA. This is relatively simple in the present instance: it merely requires that each user possess ECA, the public transformation of the CA. Then each user A may register EA with the CA. Since EA 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 the CA. Alternatively, registration can be handled by a tree-structured system: the CA 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 the

CA (see sec. 3 for a more thorough discussion of signatures) and containing EA. That is, the CA constructs a message M containing EA, identification information for A, a validity period, etc. Then the CA computes CERTA = DCA(M) which becomes A's certificate. The latter is then a public document which both contains EA and authenticates it, since the certificate is signed by the CA. Certificates can be distributed by the CA or by users, or used in a hierarchical system which 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 re-distributed by the CA each time they are used), the CA 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; e.g., 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, e.g., in a file with read protection. If greater security is desired, a second option is hardware keys ([DENN79],[FLYN78],[RIVE78]).

In the classical version of this scheme, each public/private key pair is stored on a pair of 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 may be insecure; moreover, the large size of public keys makes this impractical. However, there is also an obvious disadvantage to the ROM mode of hardware key storage, namely the fact that the party which loads a private key to ROM could retain a copy. An alternative, but still classical, 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 smart token, smart card or other device which 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 which stores both cryptographic algorithms and data.

In the classical 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.

If a token or smart card is used, it is possible (at least in theory; the technology is still evolving at the time of this writing) to generate and store keys on the same device which executes encryption and decryption algorithms. Again, security is optimal if all of these functions are combined onto one or a minimal number of chips. Use of such auxiliary devices mitigates

some of the problems encountered in the classical case. However, a token or smart card may be stolen, permitting the thief to masquerade as the user. Passwords (PINs) or biometrics used to control access to devices can substantially lessen this threat.

Capabilities such as signature generation should become feasible on devices such as smart cards in the near future. This will 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 earlier, 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 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.

The use of public-key cryptography thus reduces the problem of secret key distribution to the problem of binding user identities and user public keys. The latter is a simpler problem in that no communication of secret information is necessary. That is, users can generate their own private/public key pairs and need not expose their private keys to anyone, including themselves. However, it is critical that user public keys be properly authenticated. This is a nontrivial consideration in large networks.

Since only integrity and authenticity are considerations, users may be able to register their public components without use of a secure channel. In a large network this might require a distributed system of certifying authorities, arranged in hierarchical fashion. The feasibility of such an approach requires transitivity of trust. For example, a central certifying authority could certify a second level of certifying agents who in turn certify user public components. Other levels could be added as well. Thus a user is certified via an authentication path. Such a path need not require use of insecure channels if the nodes in the path can communicate securely. For example, a user might register with a local certifying authority in person. The local authority might be affiliated with an organization and may be able to use the user's organization ID to certify the user's public key. If the local authority can communicate securely with the central authority, the user's public key may then be forwarded to the central authority for distribution.

Assuming that user public components can be certified and distributed, 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.

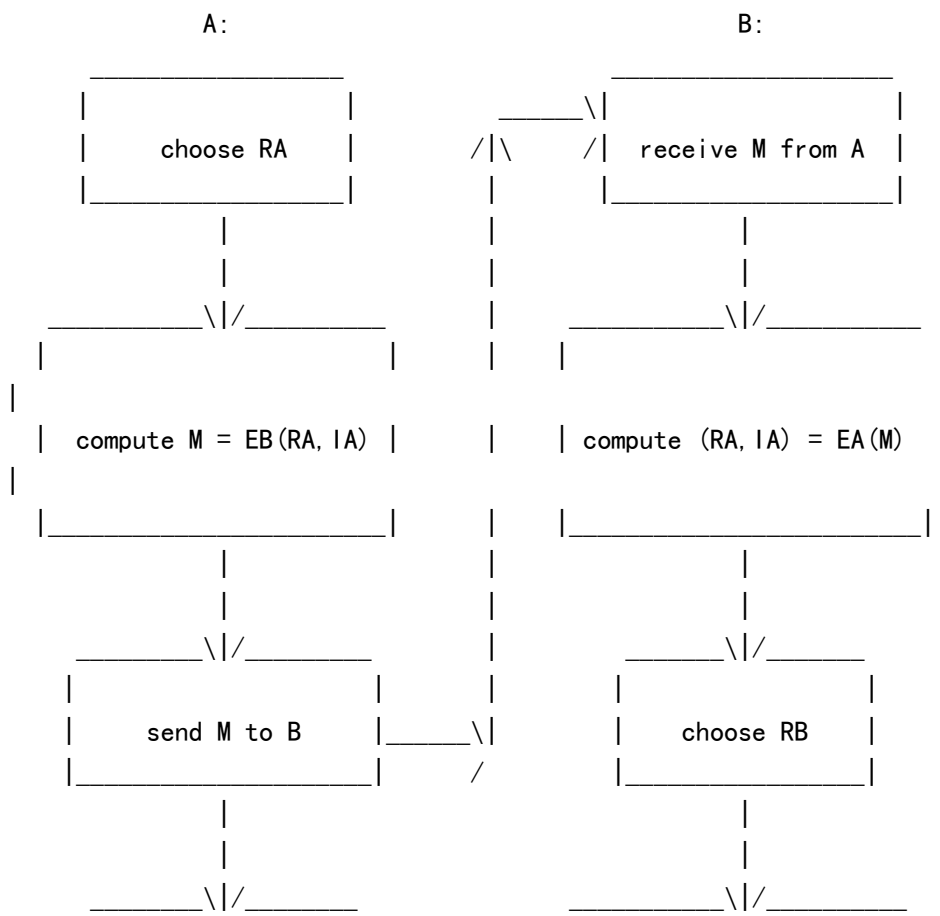
2.4.1 A protocol for key exchange.

Suppose A and B wish to establish a shared secret key. Suppose further that they have obtained each other's public components. Some possible modes for binding user public keys to user identities and distributing certified user public keys are discussed in sections 5.3.1 - 5.3.2 and 5.4.6. 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, e.g., DES is used in cipher feedback mode or cipher block-chaining mode [NATI80]. Then A might form an expanded message which contains K and other data, which could include an identification for A, a timestamp, a sequence number, a random number, etc. 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 of all A may send to B the message C = EB(RA,IDA), where EB is B's public transformation, IDA is the identification of A, and RA is a random number. Now B can decrypt C and obtain IDA. Now B generates a random number RB and sends C' = EA(RA,RB) to A. Upon decryption of C', A can verify in real time that B has received RA, since only B could decrypt C. Finally A sends C″ = EB(RB) to B; when B decrypts C″ he can verify in real time that A has received RB, since only A could decrypt C'. Thus A and B are authenticated to each other in real time; i.e., each knows that the other is really there.

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

The authentication stage of the preceding protocol is illustrated below.



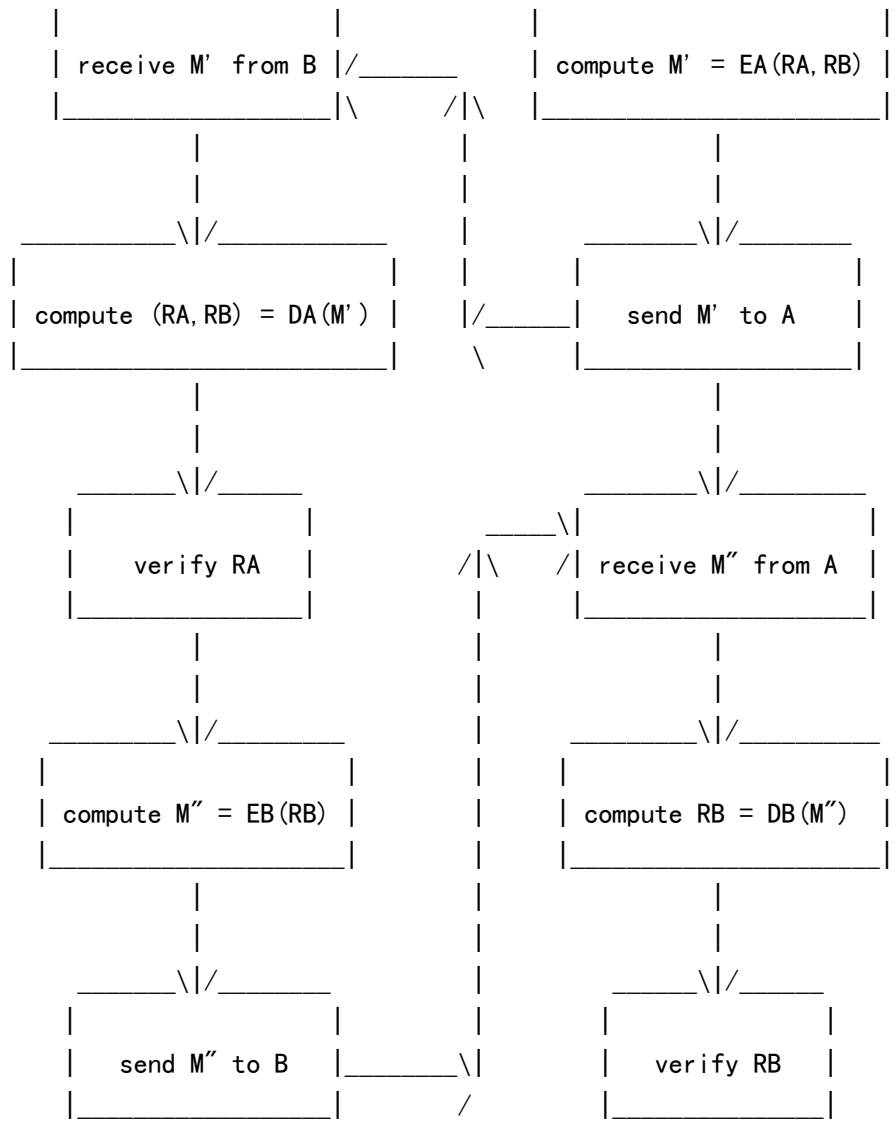


Figure 4. A Protocol for Real-Time Authentication

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 CA has created certificates CERTA and CERTB for A and B, respectively. These contain EA and EB (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 CERTA and CERTB, 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 on-line 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 the CA, assuming that the latter is on-line. In this event A requests CERTA and CERTB from S. We recall that each CERT has the form DS(M) where M contains an E. Thus when A receives the requested certificates, he can validate CERTA by computing ES(CERTA) and recovering EA. This validates the source of the certificates, thus validating CERTB as well. Hence A may retrieve a duly authenticated EB from CERTB. The disadvantage is the requirement of an on-line 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 involving the CA will not be done regularly. Thus the above procedure will only be 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 mode of key management 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 prerequisite for the proper functioning of such a key management system is the existence of a system for binding user public keys and identities. Such a system requires distributed trust.

A disadvantage of this mode of management 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, e.g., 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]. However, if an intruder accesses the central authority's private component, the intruder could forge certificates. Some scenarios for intrusions of this sort, explorations of consequences, and means for minimizing damage are explored in [DENN83b] and [DIFF88].

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 etc. as described in the protocol in the previous section. This message also requests B's certificate. Upon 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 CB by computing ES(CB) as usual. Then EB 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 which 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, e.g., on a monthly basis. Hence a user receiving a certificate from another user would not have complete assurance 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 on-line 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 on-line 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, or entries could be altered.

3. Digital signatures and hash functions.

Digital signatures were introduced by Diffie and Hellman [DIFF76b]; for surveys see, e.g., [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:

a. A receiver must be able to validate the sender's signature.

b. A signature must not be forgeable.

c. 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 which 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:

a. true signatures.

b. 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, i.e., in validating the identity of a sender. They can also serve as cryptographic checksums (i.e., error-detecting codes), 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 re-usable. 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 re-used 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 minimized by 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 which 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 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.

However, this security gain is offset 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 earlier, 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 which 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 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 previously that, like secret-key systems, public-key systems can be used for authentication. There are several distinctive features of a public-key implementation of signatures, including:

a. The possibility of incorporating both secrecy and authenticity simultaneously, assuming that the system supports both services.

b. The re-use of key material in generating signatures algorithmically.

c. 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.

There are several possible protocols for signing in public-key cryptography, depending on the services desired. In the simplest case, A simply signs a document M by sending S = DA(M) to B.

If a hash function H is used, A computes S = DA(H(M)) and sends both M and S to B. B validates A's signature S by computing H(M) = EA(S). We also noted earlier that because EA is a trapdoor one-way function, it should not be possible for an intruder to find S' such that H(M) = EA(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 EA and hence can verify H(M) = EA(S); assuming the trapdoor DA is indeed secret, only A could have sent S. Thus a priori we have nonrepudiation (but see below).

For both authenticity and secrecy, A could send

M' = EB(M,DA(H(M)))

to B; B computes

(M,DA(H(M))) = DB(M')

and then verifies that

H(M) = EA(DA(H(M))).

The last protocol is illustrated below.

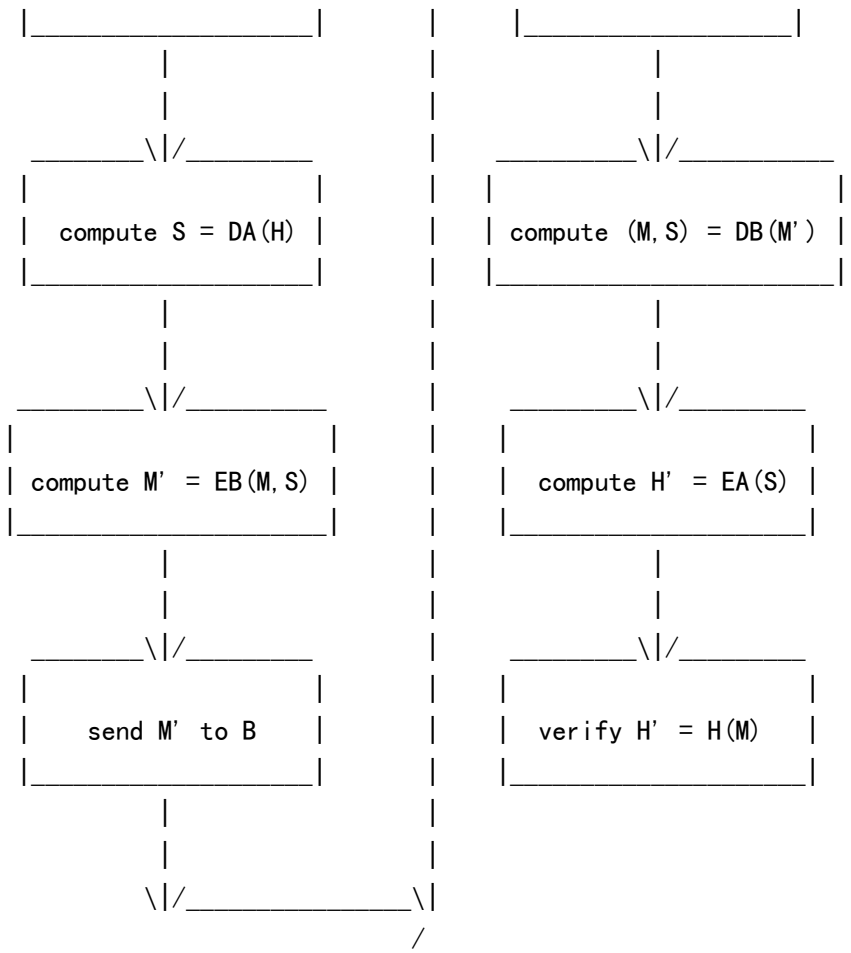


Figure 5. A Protocol for Signing with Hash Function and Secrecy

For nonrepudiation in the last case, B retains DB(M') = (M,DA(H(M))). A judge can apply EA to DA(H(M)) and compare to H(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, i.e., preventing senders from denying they have sent messages, is contingent upon users keeping their private components secret (e.g., [NEED78],[POPE79]). In the above, if DA should be compromised, then A might be able to repudiate messages sent even before the compromise. On the other hand, as in the case

of lost or stolen credit cards, A may still be held liable for messages signed before the compromise was reported to a central authority. 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 which 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. A general protocol for usage of an arbitrator A when U is sending a message M to R is given in [AKL-83]:

a. U computes S1 = ER(DU(M)).

b. U computes a header m = identifying information.

c. U computes S2 = DU(m,S1).

d. U sends m and S2 to A.

e. A applies EU to S2 and checks the identifying information in m, thereby verifying the origin of M.

f. A computes M' = (m,S1,timestamp).

g. A computes S' = DA(M').

h. A sends S' to R.

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

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; i.e., that 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 on-line central authority and real-time validity checks and timestamps, which may again create bottlenecks. Furthermore, such schemes require a network-wide 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 which 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, i.e., protocols which 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:

a. A computes X = EB(DA(M)).

b. A sends X to B.

c. B computes M = EA(DB(X)).

d. B computes Y = EA(DB(M)).

e. B acknowledges receipt of M by sending Y to A.

If this protocol is standard procedure then A can assume nonreception unless he receives acknowledgement 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:

a. C intercepts Y = EA(DB(M)).

b. C sends Y to A.

c. A thinks that this is a legitimate message from C,

unrelated to his communication with B. Following protocol:

c.1. A computes M' = EC(DA(Y)).

c.2. A computes Z = EC(DA(M')).

c.3. A acknowledges receipt of M' by sending Z to C.

d. C computes EB(DC(EA(DC(Z)))) = EB(DC(M')) = EB(DA(Y)) = M.

This of course occurs because of step c, 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 acknowledgement 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) will be much smaller than M; e.g., 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 an MDC = manipulation detection code or MAC = message authentication code). 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 which 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.

Earlier we referred 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 will exceed 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 which 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 which hash to the same value. We return to the distinction between these two criteria in section 3.2.3.

3.2.1 Usage 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 S = DA(H(M)) to B. The latter uses EA to retrieve H(M) from S, then computes H(M) directly and compares the two values for authentication. For nonrepudiation, B retains M, H(M) and S. If A attempts to repudiate M, a judge can use the three items to resolve the dispute as before: he computes H(M) = EA(S) 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 S 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:

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

b. F produces a fixed-size output.

c. F(x) is relatively easy to compute for any given x.

d. For any given y it is computationally infeasible to find x with F(x) = y.

e. For any fixed x it is computationally infeasible to find x' ≠ x with F(x') = F(x).

The most important properties are (d) and (e). In particular, (e) 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 (e) can be strengthened as we note momentarily.

Property (d) 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 earlier 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 which do not have trapdoors.

The concept of (non-trapdoor) one-way functions originated in connection with log-in procedures ([WILK68] p. 91): Needham 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].

Usage 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 {EA} in a public-key cryptosystem. In the latter, (d) above becomes a dichotomy: it is computationally infeasible for anyone except A to find M from C = EA(M); but it is easy for A, the unique holder of the trapdoor DA, to compute M = DA(C).

An example of functions which are not one-way are the private transformations in public-key systems: anyone can solve S = D(M) for M; i.e., M = E(S). This shows that signature schemes are not one-way functions; i.e., they do not satisfy (d) above. On the other hand, a deterministic signature function S satisfies (e) above; i.e., 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 (e) is trivially satisfied.

Merkle's requirements above are that a hash function must be both one-way (d) and effectively collisionless (e). In order to satisfy (a) and (b) concurrently, collisions must exist; (e) 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 (a) - (e) of the previous section may be termed a weak hash function. Property (e) 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 (a) - (d) still hold, but (e) is modified as follows: it is computationally infeasible to find any {x1,x2} with H(x1) = H(x2).

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 280 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 2-80k by the binomial theorem if the latter is small. For example, the cryptanalyst will probably have to compute H for about k = 274 = 1022 values of M' to have even one chance in a million of finding one M' which collides with M. Thus H is secure in the weak sense.

Suppose for the same H the cryptanalyst merely seeks any values {M1,M2} with H(M1) = H(M2). By example F.1, if he examines H(M) for at least 1.17 * 240 < 2 * 1012 random values of M, the probability of at least one collision exceeds 1/2. A supercomputer could probably find M1 and M2 with H(M1) = H(M2) in at most a few days. Thus H is not secure in the strong sense.

The latter attack is called a birthday attack (app. 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. 1012 above) of variations of a message to find two which 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 previously noted 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, e.g., 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:

a. Systems which have been broken.

b. Systems which are considered secure.

b.1. Systems which are of questionable practicality.

b.2. Systems which are practical.

b.2.1. Systems suitable for key distribution only.

b.2.2. Systems suitable for digital signatures only.

b.2.3. Systems suitable for key distribution and

digital signatures.

From the standpoint of sheer numbers, most systems have proven to be insecure. This is unfortunate, since some of the techniques producing insecure systems have relatively high bandwidths, where bandwidth refers to rates of encryption/decryption. Knapsack ciphers are an example (sec. 4.2.1, app. E)) of high-bandwidth but generally insecure systems. Of the systems which 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 which has defied all attempts at solution.

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

Category (b.2.3) 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 high enough bandwidth 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:

a. discrete exponentiation.

b. knapsack problems.

c. 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 (sec. 4.1).

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

a. If p is a given prime and g and M are given integers, find x such that gx M (mod p).

b. If N is a product of two secret primes:

   1. factor N.

   2. Given integers M and C, find d such that Md C (mod N).

   3. Given integers e and C, find M such that Me C (mod N).

   4. Given an integer x, decide whether there exists an integer y such that x y2 (mod N).

Gill's suggestion was based on (a), i.e., on the difficulty of computing discrete logarithms modulo a prime. This has generated systems (e.g. sec. 2.2) suitable for key distribution, and others suitable for signatures (e.g. sec. 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 ((b.2) above) is difficult, as is taking roots (b.3) and deciding quadratic residuosity (b.4). These problems have generated the most widely-known public-key system (sec. 4.1) and various others. We remark

in passing that the probabilistic scheme mentioned in section 3 and appendix P are based on (b.4); more generally, quadratic residuosity forms the basis of many zero-knowledge schemes.

Diffie's knapsack suggestion is one of many attempts to utilize 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 GCD(e,m) = 1. He finds the multiplicative inverse of e, modulo m; i.e. he finds (app. H) d in [1,m-1] with e * d 1 (mod m). Now n and e are public; d, p, q and m are secret. That is, for a user A, nA and eA constitute the public component, and dA, pA, qA the private component.

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

$$E(M) = Me \bmod n,$$

$$D(C) = Cd \bmod n.$$

In the 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; i.e. 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 which can be used for both secrecy and authentication. That is, for secrecy, A sends EB(M) to B as usual; for authentication, A sends DA(M) as usual. For both secrecy and authentication, suppose first that message digests are not

employed. Assuming nA nB, A computes

C = EB(DA(M))

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

M = EA(DB(EB(DA(M)))).

This process is illustrated below:

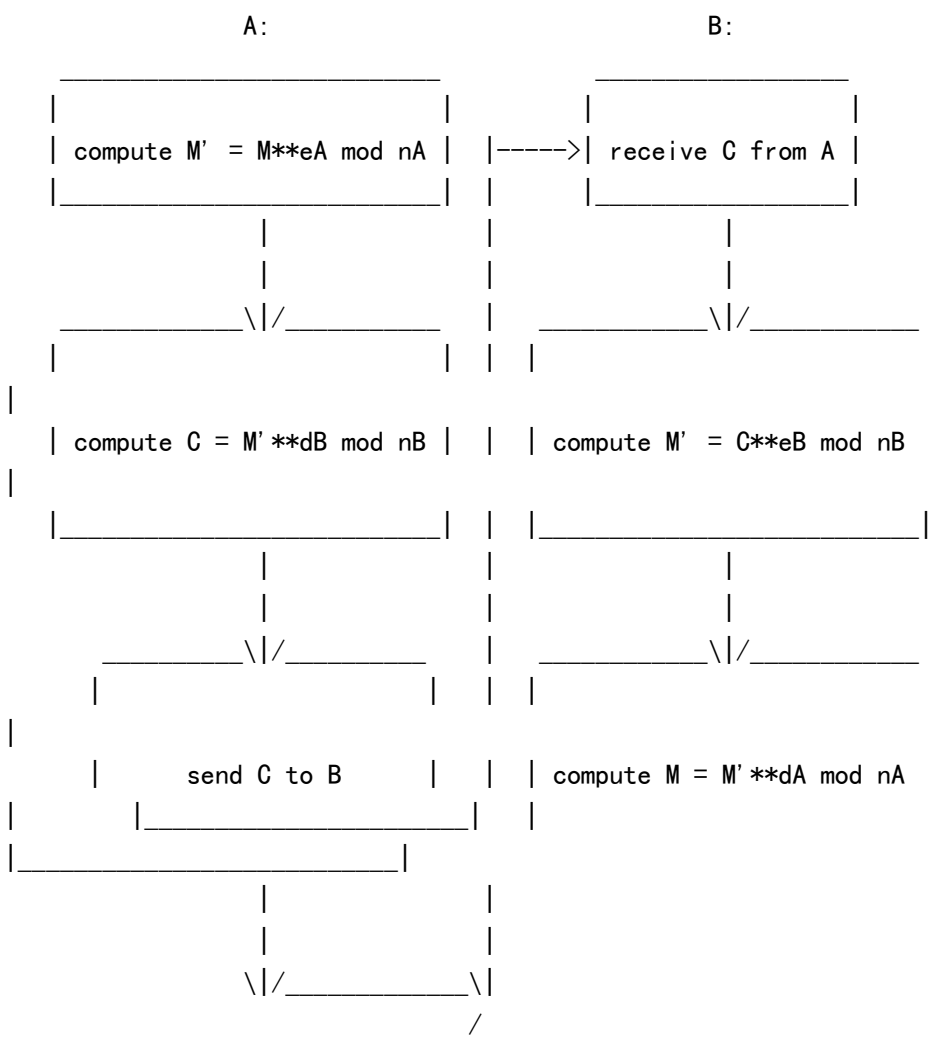


Figure 6. Using RSA for Authenticity and Secrecy

As noted in section 3.1.1, for nonrepudiation B may retain

DA(M).

This implementation only works because the range of DA is a subset of the domain of EB; i.e. [0,nA-1] is a subset of [0,nB-1]. In the event that nA nB, Kohnfelder [KOHN78b] notes that A can instead transmit

C' = DA(EB(M)).

Then B can recover M as

M = DB(EA(DA(EB(M)))).

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

However, in [DAVI83] and [DENN83b] it is noted that there is a disadvantage to this solution: A signs EB(M) rather than M. In section 3.1.1 the judge was able to apply EA to the stored quantity DA(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 DA(H(M)) and send C = EB(M') to B. This gives both secrecy and authentication; also, C may be retained for nonrepudiation. Moreover, M and DA(H(M)) are reblocked in forming M', so that no problem arises from the sizes of nA and nb.

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 at least 75 to 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, e.g., by Solovay and Strassen [SOL077]: randomly choose a set of about 100 numbers in [1,b-1]. For each number a in the set, test to see if GCD(a,b) = 1, where GCD = greatest common divisor. If this condition holds, compute the Jacobi symbol (a/b) (app. J). Both GCDs and Jacobi symbols are easy to compute via well-known algorithms (app. H,J). Now check to see if

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

If b is not prime, this check will fail with probability at least 1/2 (app. L). Thus, if 100 random a's are tested and all pass the above test, the probability of b not being prime is about 1 in 2100. 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 will not have access to high-performance computers. The probabilistic algorithms have acceptable run times 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); i.e. 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; i.e. only a small number of modular multiplications are needed. The condition GCD(e,m) = 1 presents no problem; in fact the probability that two random numbers are relatively prime is about 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 (app. 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, e.g., DES. Storage for each of p, q, and d will require 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, e.g., 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 225 Kbits/sec in hardware [SHAN90]. Decryption with a 512-bit modulus has been effected at about 11 Kbits/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 1 (mod m) is useless for determining d from e.

The difficulty of computing discrete logarithms (cf. sec. 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 = Cd mod n; i.e., he must find a discrete logarithm.

Ciphertext-only attacks are equivalent to taking roots modulo a composite number with unknown factorization, i.e. solving C = Me 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 (lem. N.3.2; cf. [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 e.g. [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 = Me mod n and C' = Me' mod n. If both C and C' are intercepted, an intruder can find (app. H) r and s with r * e + s * e' = 1; now M = Cr * 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:

a. p and q should be about the same length.

b. 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, e.g., around 100 digits. Condition (a) 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. Recent announcements of work in progress indicate that integers of around 116 digits may be factored shortly, using the same network used in the 106-digit case and another variant of the quadratic sieve.

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 which 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 1010 operations per second, at a cost of $10 million or more. Even if a future computer reached 1012 operations per second (and presumably a cost exceeding $100 million) 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 (cf. app. 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 is likely to be secure until the late 1990s, barring major algorithmic advances (cf. app. 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 above runtime estimate is generic; in particular it assumes that runtime is essentially independent of the integer being factored. However, there are algorithms which have the above as their worst-case time. 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 which 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, e.g., perhaps one 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 which 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 rd + s or rd - 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 exponent 2 for which successful ciphertext-only attacks are essentially as difficult as factoring the modulus (app. 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 usage. However, 3 and other very low exponents suffer from some 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 $M3 >> n$. This is because $C = M3 \bmod n$ becomes simply $C = M3$ if $M3 < n$; 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, e.g., e = 3 [HAST88]. Suppose A sends the same message M to each of {Bi}, i = 1,2,3. Suppose Bi uses modulus ni, and $M < ni$ for each i (this will be true, e.g., if the sender uses a modulus smaller than each of the {ni}). Assuming that each of {n1,n2,n3} is generated as a product of two random primes, the probability of duplicates among the 6 primes used is near zero. Hence it may be safely assumed that

the {ni} are pairwise relatively prime. Let Ci = M3 mod ni. Suppose all 3 {Ci} are intercepted. Using the Chinese Remainder Theorem (app. I), the intruder can find x in [0,n'), n' = n1 * n2 * n3, with x Ci (mod ni), i = 1,2,3. Thus x M3 (mod n'). But M3 < n', so M3 = x, so M = x1/3 (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. These are briefly surveyed here. As noted earlier, most of these are either insecure, of questionable practicality, or limited to one of signatures/secrecy but not both. 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 the use of the secure systems for specific applications such as digital signatures.

4.2.1 Knapsack systems.

These were proposed by Merkle and Hellman [MERK78b]. However, the essence is the use of NP-complete problems (e.g., [GARE79],[HORO78]) which had been suggested for use in designing public-key systems in [DIFF76b]. The Merkle/Hellman approach was to employ a superincreasing sequence {ai}, i.e., 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 {Mi} such that for a given Y,

$$Y = a1 * M1 + \ldots + an * Mn.$$

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

$$u > a1 + \ldots + an,$$

$$GCD(u,w) = 1,$$

a disguised knapsack problem may be produced from {ai} by defining

$$bi = w * ai \bmod u.$$

Then the associated knapsack problem

$$C = b1 * M1 + \ldots + bn * Mn$$

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

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

Then

$$W * C \equiv a1 * M1 + \ldots + an * Mn \pmod{u}.$$

Since 0 Mi 1,

$$0 \le a1 * M1 + \ldots + an * Mn < u,$$

from which

$$W * C = a1 * M1 + \ldots + an * Mn.$$

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 above modular disguise can be iterated; i.e., new w' and u' chosen and the {bi} disguised, etc.

The trapdoor knapsack yields a public-key system: if the binary representation of plaintext M is Mn...M1, let

$$E(M) = b1 * M1 + \ldots + bn * Mn.$$

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 {bi} and the private component is u, w and {ai} (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 above knapsack 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 authentication 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) = 2M1 + 3M2.

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, e.g., 1 could not be signed by computing D(1).

4.2.2 The ElGamal signature scheme.

A signature scheme derived from a modification of exponentiation ciphers was proposed by ElGamal [ELGA85].

First of all a prime p and a base a which is a primitive root modulo p (app. 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(A) = ax(A) mod p.

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

k * I 1 (mod (p-1)).

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

r = ak mod p,

s I * (M - r * x(A)) (mod (p-1)).

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

$$C = aM \bmod p,$$

$$C' = y(A)^r * r^s \bmod p.$$

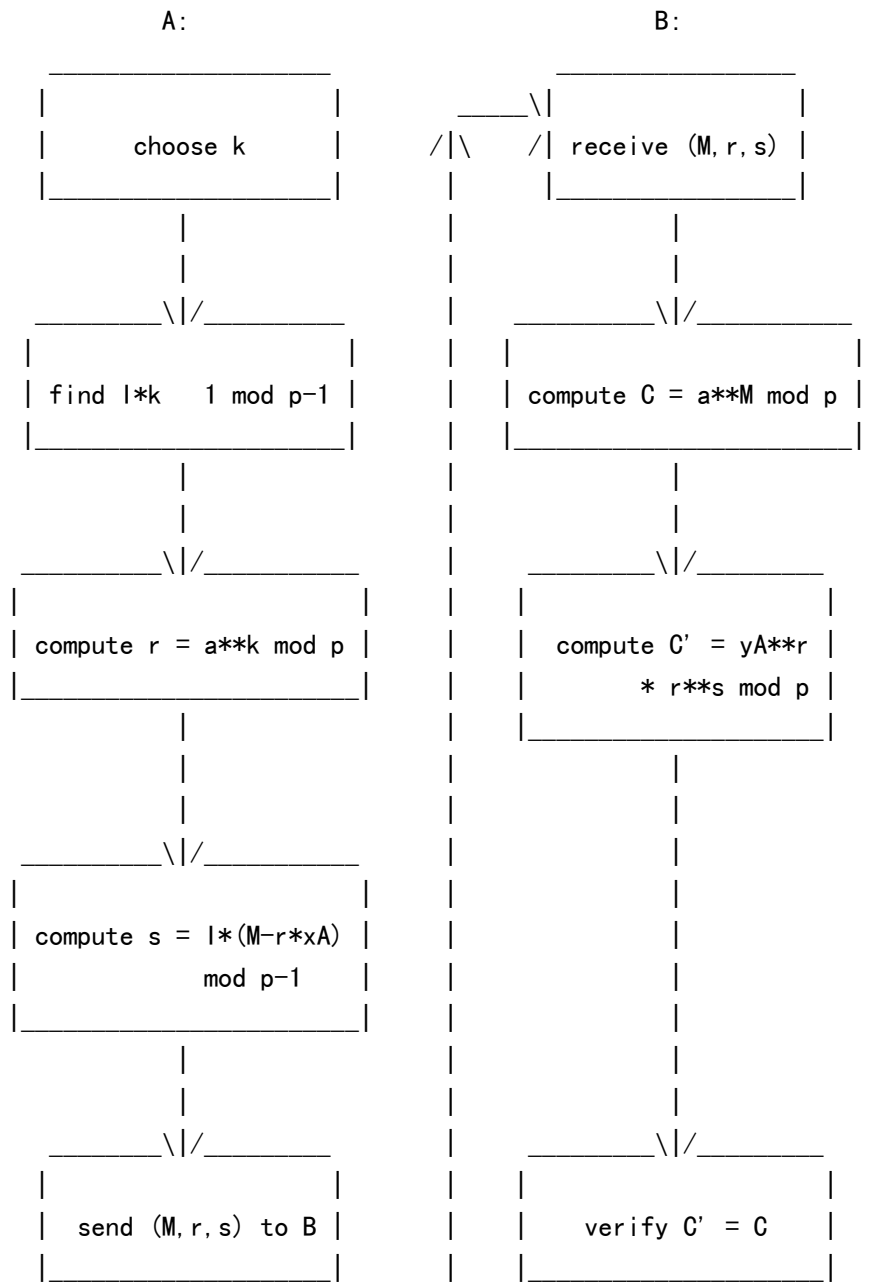
We note that

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

Hence if M is authentic and valid,

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

The ElGamal algorithm is illustrated below.


A:
B:
choose k
find l*k 1 mod p-1
compute r = a**k mod p
compute s = l*(M-r*xA) mod p-1
send (M,r,s) to B
receive (M,r,s)
compute C = a**M mod p
compute C' = yA**r * r**s mod p
verify C' = C

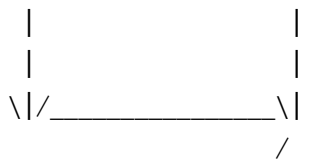


Figure 7. The ElGamal Signature Algorithm

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 = ar mod p and u * rs 1 (mod p). Then the intruder knows

aM y(A)r * rs (mod p).

Hence

u * aM y(A)r (mod p).

Thus

dx(A) (ax(A))r y(A)r (mod p).

Finally

dx(A) u * aM (mod p).

Finding x(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(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

$$aM \equiv y(A)^r * r^s \pmod{p}.$$

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

In comparing this scheme to RSA, we note that both employ exponentiation, so that their speeds of encryption/decryption should be comparable. Generating keys in the two methods is similar; finding appropriate primes is the main step in either. Security of ElGamal is more difficult to assess. The major attack against RSA, i.e., factorization, is a problem which has been studied for centuries. It is safe to say that far less time has been invested in attempts to cryptanalyze the ElGamal scheme.

Cryptanalytically, the last attack (varying r and s simultaneously) may have a complexity substantially lower than factoring or discrete logarithm; hence security of ElGamal is at best no better than RSA, and possibly much inferior, with respect to secrecy of components (it must be emphasized that this is a conservative characterization; it could also turn out that no substantial advantage is gained by varying r and s simultaneously). 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

$$M = E(r,s) = (x(A) * r + k * s) \bmod (p-1).$$

This maps the ordered pair (r,s) to a unique M. The condition

GCD(k,p-1) = 1 guarantees that an (r,s) can always be found; i.e., E is surjective. But it is not injective (one-to-one): e.g., if p = 7, k = 5, x(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, ElGamal notes that extension to GF(pn) (app. G) is feasible. However, it is not clear that extensions to finite fields are desirable (cf. sec. 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 which have keys, are insecure, or both.

4.3.1 Merkle's meta-method.

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 pre-certified 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 EK, where K is a key, which 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 EK(M) is available which satisfies these requirements. In deference to the potential use of DES to define E, assume K is 56 bits, M is 64 bits, and EK(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 EK(DK(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 = 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) = EK(M) 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 non-randomly. 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 "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 which outputs more than 64 bits, to deflect "birthday" or "square root" attacks (app. 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., 264 hash values) would be vulnerable to an attack using only 232 = 4 billion messages. On the other hand, a 112-bit output would require exhaustive search of on the order of 256 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

CAT(f(CAT("0",x)),f(CAT("1",x))) = CAT(y,z),

where y is 112 bits. Then define F0(x) = y. In the above, " " denotes constant bit string. We note that F0 hashes 119 bits to 112 bits. This is not very efficient, but the 112-bit output will deter a birthday attack. The latter would require 256 = 7*1016 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 F0 which is also a fixed-size strong one-way hash function. Finally F0 is extended to a one-way hash function F accepting inputs of arbitrary size by cascading copies of F0. Given an input x, suppose

x = CAT(x1,...,xn),

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

R0 = 0,

Ri = F0(CAT(Ri-1,xi)) (1 i n),

where in the first equation the right side is a string of 112 zeroes. Let F(x) = Rn.

Now Merkle claims that F is as secure as F0. That is, if x x' can be found with F(x') = F(x), then F0 can also be broken. The proof is inductive: if n = 1 we have F(x) = F0(CAT(0,x)). If F(x) = F(x') and x x' let z = CAT(0,x) and z' = CAT(0,x'); then F0(z) = F0(z') and z z', so that F0 is broken. Now suppose the claim is true for n. Suppose

zi = CAT(x1,...,xi) (1 i n+1),

x = zn+1,

zi' = CAT(x1',...,xi') (1 i n+1),

x ' = zn+1',

Ri = F0(CAT(Ri-1,xi)) (1 i n+1),

Ri' = F0(CAT(Ri-1',xi')) (1 i n+1),

where each xi and xi' is 7 bits. Suppose x x' but F(x) = F(x'), i.e., Rn+1 = Rn+1'. Let y = CAT(Rn,xn+1) and y' = CAT(Rn',xn+1'). Then F0(y) = F0(y'). If xn+1 xn+1' then y y' and F0 is broken. Suppose xn+1 = xn+1'. Then Rn = Rn'; but zn zn' since x x'. Now we observe that by definition Ri = F(zi) and Ri' = F(zi'). Hence F(zn) = F(zn'). By the induction hypothesis, F0 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 which 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 which 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 H0 be random (Rabin uses 0) and suppose a message M is divided into fixed-size blocks M1,...,Mn. Suppose E(K,N) represents encryption of block N with key K using a conventional system such as DES. For i = 1,..,n let

Hi = E(Mi,Hi-1).

Then a hash value is given by (H0,Hn). Coppersmith's attack assumes that an opponent knows H0 and Hn. He then constructs a bogus message whose hash value is also (H0,Hn). Assume blocksize is 64 bits. The opponent begins by specifying M1,...,Mn-2 using H0. He then generates 232 trial values X. For each X he computes a trial Hn-1 of the form

$$H(n-1,X) = E(X,Hn-2).$$

He sorts these 232 values and stores them. Now he generates 232 trial values Y. For each Y he computes a trial Hn-1 of the form

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

where D is the decryption function corresponding to E. We note that H(n-1,X) is the value that Hn-1 would have if Mn-1 = X, and H'(n-1,Y) is the value Hn-1 would have if Mn = 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 (app. F), the probability of finding X and Y is at least 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 233 encryptions, 232 storage and 64 * 232 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 meta-method this uses no external system as an adjunct. It is not keyless; an initialization vector is used. This limits the scope of usage; 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 H0 = 0. However, he also chooses a secret seed M0 which is changed every message and transmitted as a prefix to the message. Thus Assuming 32-bit blocks (to use the Mersenne prime 231-1), for i = 0,...,n he computes

$$H_{i+1} = (H_i + M_i)2 \bmod (231-1).$$

Then the hash value is Hn. 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 4 words. Recent results 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

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

Recent work calls this into question as well.

4.4 Hardware and software support.

As noted earlier, 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 64 kbps (kilobits per second) 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, ai mod b, and perhaps multiplicative inverses or greatest common divisors. As Sedlak notes further, 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 tradeoffs in such design schemes are discussed by Rivest [RIVE84]. He classifies architectures as serial (S), serial/parallel (S/P), or parallel/parallel (P/P). He then notes the following time and hardware costs:

a. Multiplying two k-bit numbers modulo a k-bit number:

   1. O(k2) time and O(1) gates on an S.

   2. O(k) time and O(k) gates on an S/P.

   3. O(log k) time and O(k2) gates on a P/P.

b. Exponentiating a k-bit number modulo a k-bit number:

   1. O(k3) time and O(1) gates on an S.

   2. O(k2) time and O(k) gates on an S/P.

   3. O(k log k) time and O(k2) gates on a P/P.

c. Key generation, k-bit primes:

   1. O(k4) time and O(1) gates on an S.

   2. O(k3) time and O(k) gates on an S/P.

   3. O(k2 log k) time and O(k2) gates on a P/P.

This is a somewhat oversimplified version of [RIVE84] 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 above to give some sample timings:

a. Decryption rate, 200-digit modulus:

1. 0.005 kbps on an S.

2. 1.3 kbps on an S/P.

3. 95 kbps on a P/P.

b. Key generation, 100-digit primes:

1. 1,200,000 ms = 20 minutes on an S.

2. 5,000 ms = 5 seconds on an S/P.

3. 70 ms on a P/P.

It may be seen that reasonably high bandwidths can be achieved, but the time/cost tradeoff 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 um CMOS which should encrypt at 40 kbps for 512-bit moduli.

Sedlak [SEDL87] discusses a highly-optimized chip which makes substantial usage 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 (see app. D). Support is provided not only for RSA but also for the ISO hash function (see above). Sedlak claims a deciphering rate of nearly 200 kbps is achievable for 780-bit moduli using a single 160,000-transistor chip with dual ciphering units, in 1.5 um CMOS. He also claims a key generation time of 2 seconds. A 5,000-transistor, 5

um CMOS prototype has been realized.

In [BRIC89] it is noted that chips capable of up to 450 kbps are being designed.

5. Implementations of public-key cryptography.

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

5.1 MITRENET.

One of the earliest implementations of public-key cryptography was in the MEMO (MITRE Encrypted Mail Office) system, a secure electronic mail system for MITRENET ([SCHA82],[SCHA80]). MITRENET is a broadband cable system with a bandwidth of 1 mbps. It uses a carrier sense protocol (e.g., [TANE81]); i.e., 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 establishment of secret keys. To implement Diffie/Hellman, use of GF(2n), with 2n - 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 is a separate network node containing public components in EPROM. Private components can be generated by users or by the system.

Each user workstation establishes secure communication with the Public Key Distribution Center. 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. However, the use of Diffie/Hellman alone does not permit authentication to be introduced.

5.2 ISDN.

In [DIFF87] a testbed secure Integrated Services Digital Network terminal developed at Bell-Northern Research is described. It can carry voice or data at 64 kbps. 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 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 DSP (digital signal processing) 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 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 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. It can authenticate calls from network users. 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. These are 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 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 which 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 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 off-line 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 on-line, off-line, or both for redundancy.

The user receives a copy of the certificate obtained from the CA. This can be cached; e.g., 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 DIT contains the certificates issued for it by other CAs, as well as certificates it issues for other nodes.

Semi-formally 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 usage 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 which 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 A1,...,An, knowing the public component of Ai permits recovery of the public component of Ai+1 from the certificate for Ai+1 issued by Ai.

For a user A to obtain B's certificate involves finding a common trusted point, i.e., a node which 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. A 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 non-hierarchical 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 C1 and C2 have certified users who communicate frequently, C1 and C2 may certify each other. Then the certificate issued by C1 for C2 can be stored in the directory entry of C1 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, e.g., 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, e.g., 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

IA = identity of A,

IB = identity of B,

DA = private transformation of A,

EA = public transformation of A,

DB = private transformation of B,

EB = public transformation of B,

TA = timestamp by A,

TB = timestamp by B,

RA = random number generated by A,

RB = random number generated by B,

CA = 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:

a. A:

1. generates an RA.

2. constructs message M = (TA,RA,IB,<data>) where <data> is arbitrary. The latter may include data encrypted under

EB for secrecy, e.g., when A is sending a data-encrypting key to B.

3. sends (CA,DA(M)) to B.

b. B:

1. decrypts CA and obtains EA. Also checks certificate expiration dates.

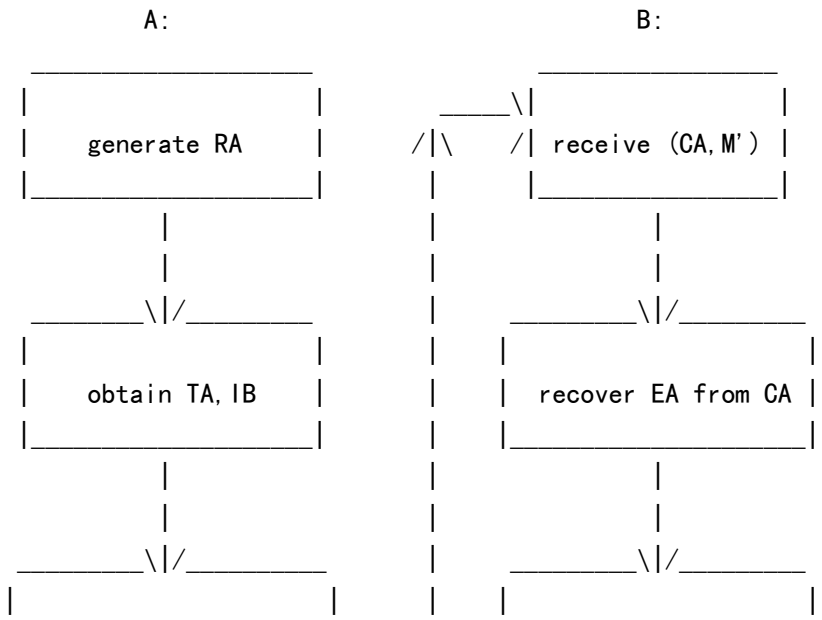
2. uses EA to decrypt DA(M), verifying both A's signature and the integrity of the signed information.

3. checks the IB contained in M for accuracy.

4. checks the TA in M for currency.

5. optionally, checks the RA in M for replay.

The one-way protocol is illustrated below, with CA denoting CA, RA denoting RA, etc.



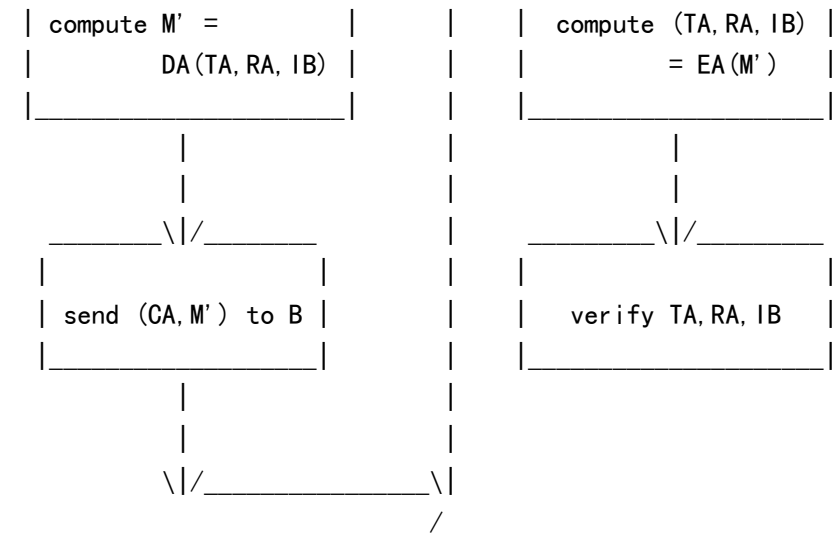


Figure 8. One-Way Authentication Protocol

The two-way protocol is:

a. as above.

b. as above.

c. B:

1. generates an RB.

2. constructs M' = (TB,RB,IA,RA,<data>) where RA was obtained previously and <data> may include data encrypted using EA.

3. sends DB(M') to A.

d. A:

1. decrypts DB(M'), verifying B's signature and the integrity of the enclosed information.

2. checks the IA contained in M' for accuracy.

3. checks the TB in M' for currency.

4. optionally checks RB for replay.

The three-way protocol is:

a. A:

   1. generates an RA.

   2. constructs M = (TA,RA,IB,<data>). Unlike the previous cases, TA may be zero.

   3. sends (CA,DA(M)) to B.

b. B:

   1. decrypts CA and obtains EA. Also checks certificate expiration dates.

   2. uses EA to decrypt DA(M), verifying both A's signature and the integrity of the signed information.

   3. checks the IB contained in M for accuracy.

   4. optionally, checks the RA in M for replay.

   5. generates an RB.

   6. constructs M' = (TB,RB,IA,RA,<data>) where RA was obtained previously; TB may be zero.

   7. sends DB(M') to A.

c. A:

   1. decrypts DB(M'), verifying B's signature and the integrity of the enclosed information.

   2. checks the IA contained in M' for accuracy.

   3. optionally checks RB for replay.

   4. checks the received version of RA against the version sent to B.

5. sends DA(RB) to B.

d. B:

1. decrypts DA(RB), verifying the signature and data integrity.

2. checks the RB 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 DA(RB,IB) to B rather than just DA(RB). This would prevent a third party from intercepting random numbers and using them to subvert the protocol.

Also, authentication tokens may be of the form M = (TA,RA,IB,RB, EB(encdata)). Then encrypted data is signed, which may cause problems. For example, suppose C intercepts DA(M) on its way to B. Then C may decrypt M and construct (TC,RC,IB,RB,EB(encdata)). In a communication between B and C, B may think that EB(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 = (TA,RA,IB,RB,encdata), M' = (TA,RA,IB,RB,EB(encdata)), and M″ = (M',DA(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 = 216 + 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 ASN.1 syntax for the authentication framework. Algorithm identifiers are included in ASN.

5.4 DARPA-Internet.

The Privacy and Security Research Group is in the process of developing a proposal for privacy-enhanced electronic mail for the DARPA-Internet community. Since this is work in progress we give only a very brief description here. Details on the current state of this project (at the time of this writing) may be found in [IAB-90],[LINN89].

Public-key cryptography has been recommended for distribution of secret keys and in support of digital signatures. DES will be used for encryption of messages and can be used for Message Integrity Check (MIC) computations. RSA has been recommended for key distribution and digital signatures. The hash functions currently supported are MD2 and MD4 [RIVE90]. Fields are included in messages to identify cryptographic algorithms and hash functions. It is specified that all implementations should support RSA and DES.

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

6. A sample proposal for a LAN implementation.

We present here a sample proposal for the implementation of privacy enhancement in a packet-switched local area network, 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. The sample proposal presented here is not unique, and is intended for illustration purposes only.

A hybrid of public-key cryptography and conventional cryptography presents advantages. 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 could 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 (cf. ann. A of [CCIT87]) include:

a. masquerade

b. replay

c. interception of data

d. manipulation of messages

e. repudiation

Masquerade refers to users representing themselves as other users. Replay refers to recording and re-sending a message at a later time. Interception of data refers to passive eavesdropping on communications. Manipulation refers to unauthorized insertions, deletions or other changes to messages. Repudiation refers to denial of sending (or possibly receipt) of a message.

There are other threats which are outside the present scope per se. For example, mis-routing 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 which can be provided include:

a. authentication

b. secrecy

c. integrity

d. 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 include basic support for the above services. This kernel could be standardized to a large extent. However, outer layers of an implementation would need to interface with various external frameworks as well. For example, a system is needed to bind user public components and user identities; this forms an important part of the authentication service. Also, support for nonrepudiation is a complex matter 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 (4) is an example of a service which relies heavily on an interface with an external system which most likely would be implementation-dependent, since its structure depends on considerations such as contractual agreements which cannot be incorporated into a generic structure.

There are other services which could be provided as well. One example is access control with differing capabilities for different classes of users. However, a public key system 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, e.g. 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 which can implement security services. These include:

a. encryption

b. manipulation detection codes

c. signatures

d. 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 compressed versions of the text. Manipulation detection in conjunction with authentication 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 form of the latter 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 which could be provided are noted, e.g., 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 which 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 NIST and Dr. Ronald Rivest) are:

a. security

b. versatility

c. bandwidth

d. data expansion

e. key size

f. key generation time

g. patent restrictions/license fees

h. software support

i. hardware support

j. number of pseudorandom bits needed

k. interoperability

Most of these are self-explanatory. The reference to pseudorandom bits in (j) is related to key generation, which requires a stream of pseudorandom bits generated from a random seed. Interoperability refers to the ability to communicate with other equipment designed to provide the same security mechanisms or functions.

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 which 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 longstanding problems such as factoring, discrete logarithm and discrete root extraction have a certain degree of security in this regard. In contrast, for example, the ElGamal signature scheme can be broken if the "transcendental" equation c = brrs (mod n) can be solved for some r and s. This is easier than solving the logarithm or root problem y = xa (mod n); and furthermore in all likelihood the ElGamal equation has not been studied as extensively.

6.5.2 Numerical 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 which 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 tradeoffs in this regard. That is, it does not seem possible to construct a system which is secure and also scores well on all numeric counts. The classic example is key size; e.g., 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 nd 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 service or the other, although hybrids can achieve the capabilities of RSA.

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 which have been proposed for hash functions, including measures of security. Some of the discussion of 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 Example of a LAN security framework.

Finally we give a brief outline of a framework for incorporating public-key cryptography into a local area network.

6.7.1 Key management.

The framework for key management described in this section 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. Data-encrypting keys are generated for individual sessions. These may be associated to an arbitrary conventional cryptosystem; DES is a possibility. 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 proposed. Since we are focusing on local-area networks, 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 proposed 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 that 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 data-encrypting keys 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. Data-encrypting keys can be produced dynamically by pseudorandom number generators which are time-variant.

An example of a generator for data-encrypting keys, as well as initializing vectors, is given in appendix C of [ANSI85]. Let E(K,Y) be encryption by DEA (essentially equivalent to DES) with key K. Let K be a DEA key reserved for usage in key generation and let V0 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 Ri 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 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 on-line.

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 on-line. 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, e.g., 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 on-line. 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/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 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 re-compute the message digest and compare to the transmitted form.

Appendix A. 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 which 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 upon the more practical matter of choosing key sizes, i.e., 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.

A.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.

A.2 Classical complexity theory.

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

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 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 classical combinatorial search problems can be given NP-complete formulations. Traveling salesman and knapsack are examples (see [GARE79] for a long list).

The class of NP-hard problems are those problems, decision or otherwise, which are at least as hard as NP-complete problems. Some NP-hard problems are so hard 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 (app. C).

A.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.

Secondly, it is important to note that the classical 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(n,1/2) = O(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; i.e., 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, often there exists a heuristic technique which 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 could be made practical.

A.4 Probabilistic algorithms.

Another important distinction between the classical theory of complexity and cryptocomplexity is that the classical 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 which are probably but not necessarily correct. Such algorithms are inappropriate in many contexts, e.g., 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 classical criteria, which focus on worst-case runtime. Instead, a probabilistic algorithm may involve a tradeoff 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 classical Turing machine model (e.g., [HORO78]) to the probabilistic Turing machine (app. D). This has proven valuable in the analysis of cryptographic systems, and probabilistic encryption schemes in particular.

A.5 Status of some relevant problems.

The security of several major cryptosystems mentioned herein 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 classical 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 classical 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 classical complexity and cryptocomplexity.

The status of the problem of taking roots mod n, i.e., 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 (lem. 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.

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, in order 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.

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

a. GaAs gates have a higher switching speed.

b. Off-chip communication in GaAs pays a relatively higher penalty.

c. GaAs chips have lower density.

Gate delays in GaAs (DCFL E/D-MESFET) may be as low as 50 picoseconds, as opposed to at least 1 nanosecond in Silicon (NMOS). Similarly, on-chip memory access in GaAs may take as little as 500 picoseconds, as opposed to at least 10 nanoseconds 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]. Inter-chip 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 CPU speed versus other subsystem speeds include cache and multilevel memory hierarchies; these may not carry over mutatis mutandis to GaAs.

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 ns with silicon technology. Furthermore, at these speeds memory access becomes a bottleneck, not to mention I/O. Thus it is not likely that uniprocessors will exceed 109 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 which 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:

a. Cost-effectiveness.

b. The interconnection network.

c. 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 which 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 be subject to tight controls.

Even if computers with 1000 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, i.e., 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 modules, 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 which 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. 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 tradeoffs 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 1012 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 which 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 which 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 = 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; 100-digit numbers should take about a year. The key to this match of algorithm and architecture is the utilization 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, utilizing 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 9 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.

This is a subject which has been explored fairly thoroughly (e.g., [BART63]), and we do 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(pn). 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 2n - 1 is (a Mersenne) prime, e.g., 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(2n); e.g., 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 n-th power seems to take log n steps regardless of the number of processors available.

Another classical example is greatest common divisor. 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 the Cray 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 which 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 which is 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 which 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 paper, and factoring in particular. The Warp is already in production.

B.4.2 Proposal for a quadratic sieve machine.

Pomerance et al. [POME88] describe a pipeline architecture which 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 is carefully analyzed, as is critical for an application-specific architecture. They speculate that a $50,000 version of the architecture should factor 100-digit numbers in about 2 weeks; or 140 digits in a year on a $10 million version; or 200 digits in 1 year on a $100 billion version. It should be noted that the last is clearly predicated on the notion that the architecture and algorithm will scale linearly with the number of processing units; we noted earlier that various bottlenecks such as inter-processor communication and memory access make this somewhat dubious. 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 and should be able to handle variable strides without the considerable loss of efficiency which 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 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 (SIMD mode). Such machines are generally applied to real-time image or signal processing, or to large-scale matrix operations.

The 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 DAP (e.g., [HOCK81]), another massively parallel machine with 4096 processors. Unlike the MPP, the DAP has been marketed.

Appendix C. The classical theory of computation.

In appendix A a number of topics from the classical theory of computation were mentioned. Here we give a more precise treatment of some notions from the classical 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:

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

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

c. s = initial state; it is in K.

d. D : K x S -> (K + {h}) x (S + {L,R}) where + denotes union.

A Turing machine = M may be interpreted semi-physically 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 # which 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:

a. q = state (element of K + {h}).

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

c. a = symbol under the head (element of S).

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

If e denotes the empty string, u in (d) is required to be an element of (S*)(S-{#})+{e}; i.e., 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 -> 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), i.e., 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, i.e., 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, i.e., there exists a Turing machine which 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; i.e., 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; i.e., 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

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

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 C. The classical theory of computation.

In appendix A a number of topics from the classical theory of computation were mentioned. Here we give a more precise treatment of some notions from the classical 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:

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

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

c. s = initial state; it is in K.

d. D : K x S -> (K + {h}) x (S + {L,R}) where + denotes union.

A Turing machine = M may be interpreted semi-physically 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 # which 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:

a. q = state (element of K + {h}).

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

c. a = symbol under the head (element of S).

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

If e denotes the empty string, u in (d) is required to be an element of (S*)(S-{#})+{e}; i.e., 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 -> 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), i.e., 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, i.e., 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, i.e., there exists a Turing machine which 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; i.e., 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; i.e., 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

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

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 logarithm etc. Here we review an extension of the classical 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 run time is needed, rather than measures such as maximum run time 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., [MCEL77]): 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} > 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) < 1/2. Suppose there exists a constant c < 1/2 such that e(x) <= 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 1/2), another concept important in zero-knowledge frameworks.

Again leaving the random tape unspecified, Pr{M(x) = y 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 run time T(x) of M is defined to be infinity if x is not accepted by M, i.e., 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) in time n} > 1/2. This is somewhat analogous to defining the run time 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 which are probabilistically computable in polynomial probabilistic run time and bounded error probability, but are not known to be computable in polynomial deterministic time, i.e., in P. An example is the characteristic function of the set of primes, which is probabilistically computable in polynomial time (e.g., [SOL077]), 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 etc., 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

$$C = b1 * M1 + \ldots + bn * Mn,$$

where the {bi} are obtained from superincreasing {ai} by modular "disguising:"

$$bi = w * ai \bmod u.$$

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 {ki},

$$ai = W * bi - u * ki.$$

In particular

$$a1 = W * b1 - u * k1,$$

and hence

$$bi * k1 - b1 * ki = (b1 * ai - bi * a1)/u.$$

Since $u > a1 + \ldots + an$, $bi < u$, and $a1 < ai$,

$$|bi * k1 - b1 * ki| < uai/(a1+\ldots+an).$$

Now it is easily shown that

$$ai+j \quad 2j-1 * (ai + 1),$$

and hence

$$|bi * k1 - b1 * ki| < 2i+1-n * u.$$

Thus

$$|k1/ki - b1/bi| < 2i+1-n*u/kibi < 2i+1-n*u/bi.$$

This shows that the {ki} are in fact not random; they are determinable via the above inequalities if u is known. Shamir [SHAM84b] observed that an intruder merely seeks any trapdoor, as represented by any u, w, and {ai} which 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 classical methods of Diophantine approximation. This yields the {ki}, 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 b1,...,bn defined by density = n/(log max {bi}). 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, i.e., 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 nu). Both definitions are derived from approximations to the actual ciphertext size, which is log(b1 * M1+...+bn * Mn). Lagarias notes that the attack in [SHAM84b] runs in time O(P(n)*nd) 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 usages of birthday attacks against hash functions. Here we give a brief summary of the relevant mathematics.

Suppose H is a function which 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 mk. 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 m-2 cells; etc. The total number of ways of distributing the objects is m(m-1)...(m-k+1) (sometimes called a falling factorial). The probability of no collisions is m(m-1)...(m-k+1)/mk. Hence the probability of at least one collision is

$$P(m,k) = 1 - (m-1)...(m-k+1)/mk-1$$

$$= 1 - (1 - 1/m)...(1 - (k-1)/m).$$

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...

Proof: for 0 < x < 1,

$$1 - x < 1 - x + x2(1 - x/3)/2 + x4(1 - x/5)/24 + ...$$

$$= e-x.$$

For k < m this gives

$$(1 - 1/m)...(1 - (k-1)/m) < e-1/m...-(k-1)/m)$$

$$= e-k(k-1)/2m.$$

Thus

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

The lemma follows. QED.

EXAMPLE F.1. Suppose the H of lemma F.1 is evaluated k times where k > (2(ln 2)m)1/2 = 1.17 * m1/2. Then the probability of at least one collision is > 1/2.

This suggests an attack on hash functions. Its name derives from the classical 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 Zn = {0,1,...,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; i.e., GCD(x,y) = n if n is the largest positive integer dividing x and y. For n > 1 let

Zn* = {x Zn : x > 0 and GCD(x,n) = 1}.

For example, Z10* = {1,3,7,9}. That is, Zn* consists of the nonzero elements of Zn which are relatively prime to n.

If a Zn* let

a * Zn* = {a * x mod n : x Zn*}.

For example, 2 * Z10* = {2,6,14,18} mod 10 = {2,6,4,8}, 3 * Z10* = {3,9,21,27} mod 10 = Z10*. We have:

LEMMA G.1. Suppose n > 1 and a Zn*. Then

a. For any x and y: a * x a * y (mod n) iff x y (mod n).

b. a * Zn* = Zn*.

c. a has a multiplicative inverse modulo n; i.e. there exists b Zn* such that b * a 1 (mod n).

Proof: if x y (mod n) then trivially a * x a * y (mod n). Conversely, suppose a * x a * y (mod n). Then n | (a * (x - y)). But GCD(a,n) = 1, so n | (x - y), i.e. x y (mod n). Thus (a) holds.

For (b): if x Zn* then GCD(x,n) = 1, so GCD(a*x,n) = 1. Thus a * x mod n Zn*. Hence a * Zn* is contained in Zn*. By (a), if a * x a * y (mod n), then x y (mod n). Thus a * Zn* consists of n distinct elements, and cannot be a proper subset of Zn*; (b) follows. In particular, since 1 Zn*, there exists some b Zn* such that a * b mod n = 1; (c) follows. QED.

G.1 The Euler Phi function.

The cardinality of a set S is denoted by |S|. In particular,|Zn*| is denoted by (n), the Euler totient function.

LEMMA G.1.1.

a. If p is prime then (p) = p-1.

b. If n = p * q, p and q prime, then (n) = (p-1)(q-1).

Proof: (a) is trivial, since Zp* = {1,...,p-1}.

For (b): let Yp = {p,...,(q-1)p}, i.e., the nonzero elements of Zn divisible by p. Let Yq = {q,...,(p-1)q}, i.e., the nonzero elements of Zn divisible by q. If a * p = b * q, then p | b and q | a; hence a * p and b * q cannot be in Yp and Yq, respectively. Thus Yp and Yq are disjoint. Letting + denote disjoint union,

Zn = {0} + Yp + Yq + Zn*.

Taking cardinalities,

p * q = 1 + q-1 + p-1 + |Zn*|.

Then (b) follows. QED.

G.2 The Euler-Fermat Theorem.

LEMMA G.2.1 (Euler's Theorem). Suppose n > 0 and a Zn*. Then

a(n) 1 (mod n).

Proof: if Zn* = {r1,...,rm} then a restatement of (b) of lemma G.1 is

{a * r1 mod n, ... , a * rm mod n} = Zn*.

Hence

a * r1 * ... * a * rm r1 * ... * rm (mod n).

Thus

am * r1 * ... * rm r1 *... * rm (mod n).

By (a) of lemma G.1, each ri above may be canceled, leaving

am 1 (mod n).

Noting that by definition m = (n) gives Euler's Theorem. QED.

COROLLARY G.2.1 (Fermat's Theorem). Suppose p is prime and a Zp*. Then

ap-1 1 (mod p).

Proof: use (a) of lemma G.1.1 in lemma G.2.1. QED.

COROLLARY G.2.2. Suppose n > 0, a Zn*, and x 1 (mod m) where m = (n). Then ax a (mod n).

Proof: we have x = m * y + 1 for some y. Now by lemma G.2.1,

ax (am)y * a 1y * a a (mod n).

QED.

COROLLARY G.2.3. Suppose n > 0. Let m = (n). Suppose e and d are in Zm* and e * d 1 (mod m). Let E(M) = Me mod n and D(C) = Cd mod n. Then D(E(M)) = E(D(M)) = M for any M in [0,n).

Proof:

D(E(M)) (Me mod n)d mod n Me*d mod n M (mod n).

The last step uses corollary G.2.2. Also 0 D(E(M)) < n and 0 M < n, so D(E(M)) = M. Similarly E(D(M)) = M. QED.

G.3 Galois fields.

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

In particular, if p is prime then Zp* = {1,...,p-1} is a group; i.e., each nonzero element of Zp has a multiplicative inverse modulo p. Thus the ring Zp = {0,1,...,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 pn elements for some prime p. These are called the Galois fields, and denoted GF(pn). We have already noted that GF(p) = Zp is defined by doing arithmetic modulo p. The elements of Zp are called the residues modulo p; i.e., each integer x has a unique representation x = q * p + r where r Zp. To define GF(pn) 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); i.e., write g(x) 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(pn). These consist of polynomials of degree n-1 with coefficients from Zp. Each of the n coefficients of a residue has p possible values, accounting for the pn element count for GF(pn).

Appendix H. Euclid's algorithm.

On a number of occasions we referred to Euclid's algorithm, which can be used to find greatest common divisors 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 greatest common divisor 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 greatest common divisor of
      x1,...,xm > 0 */
   if (m = 1) return(x1)
   else return(GCD(xm,Multi_GCD(m-1,x)));
end Multi_GCD;
```

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

For x Zn*, with latter as in appendix G, a simple extension of GCD yields the multiplicative inverse of x modulo n, i.e., u with x * u 1 (mod n). With [y] denoting the largest integer y, the extended Euclid algorithm to find u is:

```
function INVERSE(n,x) returns integer;
   /* for n > 0 and x  Zn* return u  Zn*
      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 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 7/18, 2/5, 1/3, 1/2, 0/1 in which each neighboring pair is a pair of Farey fractions; a/b and c/d are Farey fractions (e.g., [RADE64]) if a * d - b * c = 1. We have found 2 * 18 - 5 * 7 = 1, so that -5 * 7 1 (mod 18), or equivalently 13 * 7 1 (mod 18). Thus finding multiplicative inverses modulo n can also be done in O(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 x1,...,xm > 0 with GCD(x1,...,xm) = 1,
      find u1,...,um with u1x1 + ... + umxm = 1 */
   if (m = 1) return(x1)
   else
      g = Multi_GCD(m-1,x);
      for i := 1 to m-1 do yi := xi/g;
      Coeffs(m-1,y,u');
      um := INVERSE(g,xm);
      b := (1 - um*xm)/g;
      for i := 1 to m-1 do ui := b*ui';
   end else;
end Coeffs;
```

This runs in O(m*log max{xi}) 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 a later appendix that its use increases the efficiency of decryption in RSA.

We begin with a special case: with Zn as in appendix G, we have

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

a. For arbitrary a Zp and b Zq, there exists a unique x Zpq with

$$x \equiv a \pmod{p},$$

$$x \equiv b \pmod{q}.$$

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

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

Proof: we note that for any y and s, 0 y mod s s-1. Thus if x and u are as in (b), 0 x (p-1) * q + q - 1 = p * q - 1. Hence x Zpq. 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 (a). 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', i.e., x' = x - p * q * k'. Since 0 x p * q - 1, if k' > 0 then x' < 0; if k' < 0 then x' p * q. Hence x' Zpq iff k' = 0, i.e., x' = x. QED.

We note that in (b) 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 x < p * q, and a = x mod p, b = x mod q. Then

a. x = (((a - (b mod p)) * u) mod p) * q + b
(a b mod p)

b. x = (((a + p - (b mod p)) * u) mod p) * q + b
(a b mod p)

Proof: immediate from lemma I.1. QED.

The corollary provides an optimal framework for representing x via a and b, i.e., 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 {p1,...,pn} and arbitrary {a1,...,an}, there exists a unique x [0,p1*..*pn) satisfying x ai (mod pi) 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 a later appendix. In appendices O and P we also note their usage in encryption.

Let Zn* be as in appendix G. For a positive integer n and a Zn*,

a is called a quadratic residue modulo n if x2 a (mod 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 y2 a (mod n) has the form y = x + k * n where k is an integer and x Zn*.

Proof: suppose y2 a (mod n). Let x = y mod n. Then x [0,n). Also, x2 a (mod n). Now x2 = a + j * n for some j. Suppose for some m we have m > 0, m | x and m | n. Then m | a, so m = 1 since GCD(a,n) = 1. Hence x Zn*; also y = x + k * n for some k. QED.

Thus the square roots of a quadratic residue modulo n can be taken to be elements of Zn*; 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 if and only if x2 a (mod p) for some x with 0 < x < p. For example, the quadratic residues modulo 7 are

{12,22,32,42,52,62} mod 7 = {1,2,4}.

The quadratic nonresidues modulo 7 are {3,5,6}.

Given a prime p, let s = (p-1)/2 and

Sp = {x2 mod p : 0 < x s}.

Then Sp 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

a. The elements of Sp are precisely the quadratic residues modulo p.

b. There are s quadratic residues modulo p.

c. There are s quadratic nonresidues modulo p.

Proof: as noted above, Sp is a subset of the set of quadratic residues modulo p. Furthermore, if x2 y2 (mod p) then p | x2-y2; hence p | x-y or p | x+y. If x and y are in (0,s] then 1 < x+y < p; thus p | x-y; hence x = y. It follows that Sp contains distinct elements. QED.

Now suppose a is a quadratic residue, x2 a (mod p), and x Zp*. We note

(p - x)2 p2 - 2 * p * x + x2 a (mod p).

Since 0 < x < p, either x or p - x is in (0,s]. It follows that a Sp. Thus the set of quadratic residues modulo p is contained in S. Hence the two sets are identical, establishing (a). Since |Sp| = s, (b) follows. Also, the complement of Sp in Zp* is the set of quadratic nonresidues modulo p. Since |Zp*| = 2s, the complement of Sp also has cardinality s; (c) follows. QED.

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]):

a. (a/p) = 1 if a is a quadratic residue mod p.

b. (a/p) = -1 if a is a quadratic nonresidue mod p.

More generally, if k > 1 is odd and h is in Zk*, the Jacobi symbol (h/k) may be defined as follows:

a. The Jacobi symbol (h/p) coincides with the Legendre symbol if p is prime.

b. If k = p1*...*pm with pi prime, (h/k) = (h/p1)*...*(h/pm).

An efficient mode for computing the Jacobi symbol is via the recursion:

a. (1/k) = 1.

b. (a*b/k) = (a/k)(b/k).

c. (2/k) = 1 if (k2-1)/8 is even, -1 otherwise.

d. (b/a) = ((b mod a)/a).

e. If GCD(a,b) = 1:

i. (a/b)(b/a) = 1 if (a-1)(b-1)/4 is even.

ii. (a/b)(b/a) = -1 if (a-1)(b-1)/4 is odd.

The key step in the recursion is (e), which is Gauss' law of quadratic reciprocity (e.g., [RADE64]). The above 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 x2 a (mod p), i.e., 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

a. a has exactly two square roots modulo p in Zp*. One of

these lies in (0,s] and the other in (s,p-1].

b. the square roots of a modulo p can be found in probabilistic polynomial time.

Proof: by lemma J.1.1 we know that a Sp as defined in J.1; hence there exists a unique x in (0,s] with x2 a (mod p). Then y = p - x satisfies y2 a (mod p), and y is in (s,p-1]. Conversely, suppose y is in (s,p-1] and y2 a (mod p). Then x = p - y is in (0,s], and x2 a (mod p). Hence y is unique. Thus we have (a).

For (b) we invoke a probabilistic polynomial-time algorithm for finding square roots modulo a prime (e.g., [PERA86] or p. 22 of [KRAN86]). QED.

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 Zp* is a quadratic residue modulo p, define

ep(a) = as mod p (s = (p-1)/2).

Then we have

LEMMA J.4.1. If p > 2 is a prime and a Zp*, ep(a) = 1 if and only if a is a quadratic residue modulo p; ep(a) = p - 1 if and only if a is a quadratic nonresidue modulo p.

Proof: by the Euler/Fermat Theorem (cor. G.2.1), ep(a)2 1 (mod p). By lemma J.3.1, 1 has exactly two square roots modulo p in Zp*, namely 1 and p - 1. Hence ep(a) = 1 or p - 1. If a is a quadratic residue modulo p, then a = x2 mod p for some x with 0 < x < p. Then ep(a) = xp-1 mod p. Again by Euler/Fermat, ep(a) = 1. Thus all s quadratic residues a satisfy as - 1 = 0 (mod p). An s-th degree congruence modulo p has at most s solutions (e.g., [RADE64] p. 21).

Thus all quadratic nonresidues b must have ep(b) = p - 1. QED.

Also, ep can be evaluated in O(log p) time. Thus quadratic residuosity mod 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 Zp* be as in appendix G. Suppose p is a prime and a Zp*. Suppose m > 0, am 1 (mod p), but ar ! 1 (mod 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 (cor. G.2.1), which shows m < p. Let

Cp(a) = {ax mod p : 0 x < m}.

Then we have

LEMMA K.1. Suppose p is prime, a Zp*, and a belongs to the exponent m modulo p. Then:

a. Cp(a) contains distinct elements; i.e. |Cp(a)| = m.

b. Cp(a) is a cyclic subgroup of Zp*.

c. m | p-1.

Proof: suppose Cp(a) does not contain distinct elements. Then for some {k,j} in [0,m) with k < j we have ak aj (mod p). Thus aj-k 1 (mod p), contradiction. This gives (a). Now suppose x and y are in [0,m) and x + y = k * m + r where r is in [0,m). Since am 1 (mod p), ax * ay mod p = ar mod p. Thus Cp(a) is closed under

multiplication, and forms a subgroup of Zp*; this is called the cyclic subgroup generated by a (e.g., [HERS64]). This gives (b). The order of a subgroup divides the order (= p - 1) of the whole group Zp* (ibid). This gives (c). QED.

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 Cp(g) = Zp*; that is, each y [1,p) has a unique representation y = gx mod p for some x [0,p-1).

Proof: Cp(g) is a subset of Zp*. The result follows from lemma K.1, which shows |Cp(g)| = p - 1 = |Zp*|. QED.

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, e.g., 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 Zp*. Thus g is a generator of Zp*.

LEMMA K.3. Suppose p > 2 is prime and p-1 has k distinct prime factors which are known. Then:

a. The number of primitive roots modulo p is (p-1), where is the Phi function (app. 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 (a) see, e.g., [RADE64] p. 49. For (b), if the exponent of a modulo p is m, then m | p-1 by lemma K.1. Now m < p - 1 iff m | (p-1)/q for some prime factor q of p, whence a(p-1)/q 1 (mod p). Thus a is a primitive root modulo p iff a(p-1)/q ! 1 (mod 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). QED.

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: (p-1) = q - 1 = (p-3)/2; thus for large p, a random element of Zp* has about a 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:

a. gs -1 (mod p).

b. g is a quadratic nonresidue modulo p.

Proof: for (a): we note g0 1 (mod p). Since g is a generator of {1,g,...,gp-2} and 0 < s < p-1 we thus know gs ! 1 (mod p). Also, g2s 1 (mod p), so that gs 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 gs -1 (mod p).

For (b): suppose x2 g (mod p). Now x gr (mod p) for some r; hence g2r g (mod p), and g2r-1 1 (mod p). Thus 2r - 1 0 (mod p-1), impossible since p - 1 is even. QED.

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..),

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 evens may be ignored, so that roughly (ln x)/2 odd numbers need be tested. For example, about (ln 10100)/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]).

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 run time 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, e.g., 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 [SOL077] mentioned in section 4.1.1 (although this approach is commonly attributed to Solovay and Strassen, it was noted implicitly by Lehmer [LEHM76]; cf. [LENS86]). If n is an odd positive integer and a Zn*, with the latter as in Appendix G, let

e(a,n) a(n-1)/2 (mod n), -1 e(a,n) 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) ep(a) (mod p) with ep as in appendix J.4. Lemma J.4.1 shows that e(a,p) = (a/p), where the latter is the Jacobi symbol (app. J.3). The latter equality thus provides evidence of primality. That is, if p is prime and a Zn* then e(a,p) = (a/p). Per se this is useless, but the contrapositive provides a proof of compositeness: if (a/n) 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 Zn*: (a/n) = e(a,n)}.

Now the Jacobi symbol is multiplicative; i.e.,

(a*b/n) = (a/n) * (b/n).

Also

e(a*b,n) e(a,n) * e(b,n) (mod n).

That is, e is also multiplicative. It follows that G is a subgroup of Zn*. The order of G must divide the order of Zn* (e.g., [HERS64] p. 35). Thus if G Zn*, G has cardinality at most (n-1)/2. Solovay and Strassen showed that indeed G Zn* if n is composite. Hence if n is composite and a is chosen at random, the probability that a is in G is at most 1/2. We thus test as follows:

```
function Solovay_Strassen(a,n) returns charstring;
   /* for n > 2, n odd, a  Zn, 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:

a. n is prime.

b. n is composite and GCD(a,n) > 1.

c. n is composite, GCD(a,n) = 1, and (a/n) e(a,n).

In the remaining case, i.e. 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 1/2 in this case. If 100 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 (see [SOL077]).

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 Zn*}.

We note that e' differs only slightly from e, taking on the value p - 1 instead of -1.

He shows that if n is odd, G = {1,p-1} iff n is prime. Again 100 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, conclude n is prime if any -1's are found; otherwise 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 * 2k where u is odd and k > 0, let exp(i) = 2i, 0 i k - 1. If a Zn* then a is a witness to the compositeness of n if au ! 1 (mod n) and au*exp(i) ! -1 (mod n), 0 i k - 1. Again, e.g., 100 witnesses may be tested; if none attest to compositeness of n then n may be assumed prime.

When n 3 (mod 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; i.e., E(M) = Mk mod p, D(C) = CI mod p, where K * I 1 (mod 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 (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) = Me mod n, D(C) = Cd

mod n, n = p * q, e * d 1 (mod m), m = (p-1)(q-1). By lemma G.1.1 we have (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, i.e., 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 1 (mod m). This can be done in O(log n) steps using INVERSE from appendix H. 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 = Cd and x = M = y mod 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 = (Cp-1)k * Cr mod p = 1k * Cr mod p = (C mod p)r mod p.

Similarly if d = j * (q - 1) + s,

b = (C mod q)s mod q.

Also r = d mod p-1 and s = d mod q-1. Thus by corollary I.1, an algorithm for decryption [QUIS82] is as follows:

a. Compute

i. a = (C mod p)d mod (p-1) mod p,

ii. b = (C mod q)d mod (q-1) mod q.

b. Find u with 0 < u < p and

u * q 1 (mod p).

c. Use one of

i. M = (((a - (b mod p)) * u) mod p) * q + b
(a b mod p),

ii. 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 INVERSE, 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, i.e., 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 appendices O and P. 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 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 (cf. sec. 4.1.4) to obtain a modification of RSA which is provably equivalent to factoring. Essentially his scheme was to encrypt via

$$EN(x) = x2 \bmod N.$$

Let ZN* 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:

a. Suppose z ZN*. 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.

b. A quadratic residue z modulo N has exactly four square roots of the form {x,N-x,y,N-y} in ZN*.

c. 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 ZN*, 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 Zp* and Zq*, respectively, we have r2 z (mod p) and t2 z (mod q). By the Chinese Remainder Theorem (app. I) there exists w in ZN with w r (mod p) and w t (mod q). Then w2 z (mod p or q), so w2 z (mod N). Thus z is a quadratic residue modulo N. This proves one direction of (a).

Now suppose z is a quadratic residue modulo N. Let zp = z mod p and zq = z mod q. Then z ZN*, and hence zp Zp* and zq Zq*. Also, w2 z (mod N) for some w in ZN*. Thus w2 z (mod p or q) and hence w2 zp (mod p) and w2 zq (mod q). Thus zp and zq are quadratic residues modulo p and q, respectively, proving (a) in the other direction.

Furthermore, by lemma J.3.1, zp has exactly two square roots {x1,x2} in Zp*, and zq has exactly two square roots {y1,y2} in Zq*. Hence w xi (mod p) and w yj (mod 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 ZN by a given i and j; hence z has at most four square roots modulo N in ZN*. 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 ZN*. Let x denote one root. Then N - x is another. If y is a third, then N - y is the fourth. This proves (b).

By lemma J.3.1, {xi} and {yj} can be found in probabilistic polynomial time; the corresponding w's can then be found in polynomial time. This proves (c). QED.

COROLLARY N.1.1. Suppose N = p * q, p and q primes > 2. Then:

a. EN is a four-to-one function.

b. If p and q are known and z is a quadratic residue modulo N, the four values of x for which EN(x) = z can be found in probabilistic polynomial time.

Proof: immediate from the previous lemma. QED.

N.2 The Jacobi symbol once more.

Suppose N = p * q where p and q are primes > 2. Then for x in ZN*, 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 mod p and x mod 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, ZN* 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	Q00(N)	1
-1	-1	Q11(N)	1
1	-1	Q01(N)	-1
-1	1	Q10(N)	-1

LEMMA N.2.1. Suppose N = p*q, p and q primes > 2. Then:

a. The {Qij(N)} partition ZN* into disjoint classes.

b. Q00(N) is the set of quadratic residues modulo N; the other Q's contain nonresidues.

c. For i = 0 or 1 and j = 0 or 1, |Qij(N)| = (p-1)(q-1)/4.

Proof: (a) is trivial, and (b) is immediate from lemma N.1.1. For (c), let g and h be generators for Zp* and Zq*, respectively. For w ZN*, suppose w gr (mod p) and w hs (mod q), where 0 r < p - 1 and 0 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 ZN* to Zp x Zq. Also, if f(w) = (r,s) then

(w/p) = (gr/p) = (g/p)r,

(w/q) = (hs/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 mod 2 and j = s mod 2. Then (w/p) = (-1)i and (w/q) = (-1)j. Hence w is in Qij(N). For k = 0,1 let Zpk and Zqk be the subsets of Zp* and Zq*, respectively, which contain elements = k (mod 2). Then for i = 0,1 and j = 0,1, f is a bijection from Qij(N) to Zpi x Zqj. The latter has cardinality (p-1)(q-1)/2; this gives (c). QED.

Thus exactly half of the elements of ZN* have (x/N) = -1; these are nonresidues modulo N. The other half have (x/N) = 1. Of the latter, half (i.e., Q00(N)) are quadratic residues and half (i.e., Q11(N)) are nonresidues. The quadratic residuosity conjecture is that determining which of the elements of ZN* 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:

a. For x,y ZN*, x * y is a quadratic residue modulo N if and only if x and y are in the same Qij(N).

b. The product of quadratic residues is a quadratic residue.

c. The product of a quadratic residue and a nonresidue is a nonresidue.

Proof: suppose x Qij(N) and y Qrt(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 mod N will be a residue if and only if i = r and j = t. This gives (a). In particular, x * y mod N is a residue if both are in QOO(N), yielding (b). If x is a residue and y a nonresidue, x is in QOO(N) but y is not; (c) follows. QED.

Thus the quadratic residues modulo N form a subgroup of ZN*.

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 ZN*, x2 y2 (mod N), and y ! x or -x (mod N). Then possession of x and y permits factoring N in deterministic polynomial time.

Proof: we have x - y ! O (mod N) and x + y ! O (mod N), so GCD(N,y+x) < N and GCD(N,y-x) < N. Now x2 - y2 (x-y)(x+y) O (mod N). Hence (x-y)(x+y) O (mod p). Thus p | y-x or p | y+x; also p | N. If p | y-x, p | GCD(N,y-x) < N, so p = GCD(N,y-x). The latter can be found via Euclid's algorithm (app. H). If p | y+x, p | GCD(N,y+x) < N, so p = GCD(N,y+x). QED.

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/logc N of quadratic residues, c a constant positive integer. Then N can be factored in probabilistic polynomial time.

Proof: let m = logc N. Choose x1,...,xm randomly in ZN* and compute zi = xi2 mod N, i = 1,...,m. If any zi has a square root w computable in probabilistic polynomial time via the hypothetical algorithm, find w. The probability that w ! xi or -xi (mod N) is 1/2. In this event possession of xi and w factors N by lemma N.3.1. The probability that some zi has a computable square root is at least 1/2. Thus the probability of factoring N is at least 1/4. If this procedure is repeated m times, the probability of success is at least 1 - (3/4)m. Also, the problem size is log N, so m is a polynomial in the problem size. QED.

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 EN above is a trapdoor one-way function, with (p,q) forming the trapdoor.

Proof: by the previous theorem, an algorithm to invert EN would yield an algorithm to factor N. QED.

N.4 Quadratic residuosity and Blum integers.

Suppose N = p * q where p 3 (mod 4) and q 3 (mod 4). Then N is called a Blum integer (normally p and q are specified to have roughly the same size). For example, N = 77 is used in appendix 0.

LEMMA N.4.1. If p,q 3 (mod 4) then (-1/p) = (-1/q) = -1.

Proof: in lemma J.4.1 take a = -1. We recall that (-1/p) = 1 if and only if -1 is a quadratic residue modulo p. Thus (-1/p) = (-1)(p-1)/2 (mod p) and similarly (-1/q) = (-1)(q-1)/2 (mod q). The result follows. QED.

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). QED.

LEMMA N.4.3. Suppose N = p * q is a Blum integer, x and y are in ZN*, x2 y2 (mod N), and x ! y or -y (mod N). Then (x/N) = -(y/N).

Proof: x2 y2 (mod p), so (y-x)(y+x) 0 (mod p), so y d * x (mod p) where d = 1 or -1. Similarly y e * x (mod 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 d * x (mod q or p), so y d * x (mod N). But y ! x or -x (mod N), contradiction. Thus e d and e * d = -1. QED.

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 Qij(N), i = 0,1, j = 0,1.

Proof: let {x,N-x,y,N-y} be the square roots of z in ZN*. Since N-x -x (mod p), by lemma N.4.2, (N-x/p) = -(x/p). Similarly (N-x/q) = -(x/q), so if x Qij(N), N-x Q1-i,1-j(N); similarly for y and N-y. Now x2 y2 (mod N) and y ! x or -x (mod 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 Qi,1-j(N) and Q1-i,j(N) in some order. QED.

For Blum integers we can restrict the function EN to QOO(N); i.e., let BN : QOO(N) -> QOO(N) by BN(x) = x2 mod N. Then we have

LEMMA N.4.5. If N = p * q is a Blum integer then:

a. BN is a permutation on QOO(N), i.e., on the set of quadratic residues modulo N.

b. If the factorization of N is computationally intractable then BN 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 Q00(N), the equation EN(x) = y has exactly four solutions which can be found in probabilistic polynomial time. By lemma N.4.4, exactly one of these, x00, lies in Q00(N); x00 is easily extracted from the set of four since only it satisfies (x/p) = (x/q) = 1. Hence x00 is the unique solution of BN(x) = y. Thus BN is a permutation on Q00(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 BN can be converted to an algorithm to factor N. QED.

LEMMA N.4.6. If N = p * q, p and q prime, then it is possible to find x in Q11(N), i.e., x ZN* such that x is a quadratic nonresidue modulo N and (x/N) = 1, in probabilistic polynomial time.

Proof: choose a in Zp* 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 1/2, so that the probability of not finding an a with a(p-1)/2 = 1 (mod 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 ZN* with y a (mod p) and y b (mod q), in polynomial time (in length of N). QED.

Appendix O. 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 which 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 appendix P.

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 which 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. Upon 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 usage of zero-knowledge which 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; e.g., take N = 77. Everyone knows this number. The authority may then choose, e.g., two numbers which form an ID for Alice. Suppose these are {58,67}. Everyone knows Alice's ID. The authority then computes two other numbers {9,10} which are given to Alice alone; she keeps these private. The latter numbers were chosen because 92 * 58 1 (mod 77) and 102 * 67 1 (mod 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

$$192 \equiv 53 \pmod{77},$$

$$242 \equiv 37 \pmod{77},$$

$$512 \equiv 60 \pmod{77}.$$

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

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

Bob sends E to Alice. On receipt, Alice computes

$$19 * 90 * 101 \equiv 36 \pmod{77},$$

$$24 * 91 * 100 \equiv 62 \pmod{77},$$

$$51 * 91 * 101 \equiv 47 \pmod{77}.$$

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

$$362 * 580 * 671 \equiv 53 \pmod{77},$$

$$622 * 581 * 670 \equiv 37 \pmod{77},$$

$$472 * 581 * 671 \equiv 60 \pmod{77}.$$

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 only been 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, e.g., 4, by the authority, with a secret also consisting of 4 numbers. Furthermore, Alice would have generated more random numbers, e.g., 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

$$92 * 58 \equiv 1 \pmod{77},$$

$$102 * 67 \equiv 1 \pmod{77}.$$

This says that 92 and 58 are multiplicative inverses modulo 77, as are 102 and 67.

Thus

$$362 * 580 * 671 \equiv 192 * 92^0 * 580 * 102^1 * 671$$

$$\equiv 192 * (92 * 58)^0 * (102 * 67)^1$$

$$\equiv 192 \equiv 36 \pmod{77},$$

$$622 * 581 * 670 \equiv 242 * 92^1 * 581 * 102^0 * 670$$

$$\equiv 242 * (92 * 58)^1 * (102 * 67)^0$$

$$\equiv 242 \equiv 37 \pmod{77},$$

$$472 * 581 * 671 \equiv 512 * 92^1 * 581 * 102^1 * 671$$

$$\equiv 512 * (92 * 58)^1 * (102 * 67)^1$$

$$\equiv 472 \equiv 53 \pmod{77}.$$

Thus the checks that Bob uses serve their purpose properly; i.e., 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 92 and 102 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; i.e., 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:

$$x2 * 580 * 671 \equiv 53 \pmod{77},$$

$$y2 * 581 * 670 \equiv 37 \pmod{77},$$

$$z2 * 581 * 671 \equiv 60 \pmod{77}.$$

Carol will have succeeded in her masquerade if she chose {x,y,z} to make these checks come out right. But $67^{-1} \equiv 102 \equiv 23 \pmod{77}$ and $58^{-1} \equiv 92 \equiv 4 \pmod{77}$, so $x2 \equiv 53 * 23 \equiv 64 \pmod{77}$, $y2 \equiv 37 * 4 \equiv 71 \pmod{77}$ and $z2 \equiv 60 * 23 * 4 \equiv 53 \pmod{77}$. Could Carol solve these quadratic equations to find, e.g., 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 (app. N). Thus, anyone who does not know Alice's secret ({9,10} in the above) cannot impersonate her when N is large.

Another possibility is that Alice's interaction with Bob might give Bob information which could allow impersonation of Alice, at least on one occasion, by replay. Suppose Bob tries to convince Carol that 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

$$362 * 581 * 671 \quad 71 \pmod{77}.$$

Since $71 \quad 53 \pmod{77}$, 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

$$r2 * 581 * 671 \quad 53 \pmod{77},$$

$$s2 * 580 * 670 \quad 37 \pmod{77},$$

$$t2 * 580 * 671 \quad 60 \pmod{77}.$$

As before, this gives $r2 \quad 58\text{-}1 * 67\text{-}1 * 53 \quad 25 \pmod{77}$, and similarly $s2 \quad 37 \pmod{77}$, $t2 \quad 71 \pmod{77}$. 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 above there is one chance in 64 that Carol would choose F = E, in which case Bob's deception would go unmasked; 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 himself to Bob, she may have inadvertently revealed some information which could enable Bob to learn his secret, i.e., {9,10}, which 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 that Alice started with three numbers {a,b,c}, but he doesn't know these were {19,24,51}. He knows that Alice's secret is {u,v} but doesn't

know these are {9,10}. He knows that the authority computed u and v from

$$u2 \equiv 58-1 \cdot 4 \pmod{77},$$

$$v2 \equiv 67-1 \cdot 23 \pmod{77}.$$

He also knows that Alice computed

$$a2 \equiv 53 \pmod{77},$$

$$b2 \cdot 37 \pmod{77},$$

$$c2 \equiv 60 \pmod{77},$$

$$a * u0 * v1 \cdot 36 \pmod{77},$$

$$b * u1 * v0 \equiv 62 \pmod{77},$$

$$c * u1 * v1 \cdot 47 \pmod{77}.$$

This gives Bob 8 equations from which to deduce {u,v}. However, the last 3 are redundant, and as we have noted, the first 5 cannot be solved when N is large.

This shows informally that the above protocol works:

a. It identifies Alice by proving that she possesses the secret {9,10}.

b. It identifies Alice uniquely: anyone who doesn't know this secret cannot impersonate Alice.

c. Alice's secret is not revealed in the process of proving she possesses it.

Actually, (a) means that 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); i.e., (c) states loosely that the protocol is zero-knowledge. All of this depends on the assumption that equations of the form x2 y (mod N) cannot be solved if N is the product of two large primes which are not known, but can be solved easily if the primes are known (app. 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 above example demonstrates its essence in a context which has been proposed as a candidate for actual implementation in smart-card based identification schemes.

Appendix P. Alternatives to the Diffie/Hellman model.

The text of this treatise has been based on the work of Diffie and Hellman [DIFF76b]. This model of public-key cryptography includes two characteristics which have received some criticism:

a. Security of Diffie/Hellman type systems is generally established empirically, i.e., by failure of cryptanalysis.

b. Security of Diffie/Hellman type systems is dependent upon a superstructure which binds user IDs and public components.

In appendix A we noted that it has proven to be difficult to develop a comprehensive axiomatic framework in which to establish the security of Diffie/Hellman type public-key systems in some provable sense. 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 when the entire plaintext cannot be found.

In section 5 we noted some examples of authentication frameworks (e.g., use of certificates) which 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.

P.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:

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

b. 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; i.e., the adversary does not have a c-advantage in guessing P.

In (a) the reference is to the problem of deciding for a given x whether there is a y such that y2 x (mod N). As in the case of computing discrete logarithms or extracting roots, this is computationally infeasible if the factorization of N is unknown (app. N).

An example of (b): if the messages consist of uniformly distributed bit strings and P is "least significant bit = 0," then p = 1/2. If the Goldwasser/Micali scheme is employed, an adversary cannot guess the least significant bit of plaintext with probability greater than 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 (app. 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 = (m1,...,mk) to A, B randomly selects {x1,...,xk} in ZN* and computes

$$z_i = x_i^2 \bmod N \quad (i = 1,...,k).$$

Then B sets ei = zi if mi = 0, ei = y * zi otherwise. The encoding of m is e = (e1,...,ek), which B sends to A. To decode e, A sets mi = 1 if ei is a quadratic residue modulo N, and mi = 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 * zi, 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 ZN* with (r/N) = 1, where (r/N) is the Jacobi symbol (app. 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 1/2; i.e., exactly half of the elements of ZN* 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 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 above scheme is only one of a class of encryption schemes which Rivest and Sherman [RIVE82] term randomized encryption techniques; various other schemes are summarized there. We have characterized schemes such as the 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.

P.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 trivializes 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, e.g., 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 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 usage 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 appendix O.

REFERENCES

[ADLE79] L. Adleman, "A subexponential algorithm for the discrete logarithm problem with applications to cryptography," in 20th Annual Symposium on Foundations of Computer Science, 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 D. Chaum, R. L. Rivest, and A. T. Sherman, Eds., Advances in Cryptology: proceedings of CRYPTO 82, a Workshop on the Theory and Application of Cryptographic Techniques, Santa Barbara, CA, August 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 D. S. Johnson, T. Nishizeki, A. Nozaki, and H. S. Wilf, Eds., Discrete Algorithms and Complexity, proceedings of the Japan-US Joint Seminar, Kyoto, Japan, 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," Annals of Mathematics, Vol. 117, 1983, pp. 173-206.

[AKL-83] S. G. Akl, "Digital signatures: a tutorial survey," Computer, Vol. 16, No. 2, February 1983, pp. 15-24.

[AKL-84] S. G. Akl and H. Meijer, "A fast pseudo random permutation generator with applications to cryptology," in G. R. Blakley and D. Chaum, Eds., Lecture Notes in Computer Science Vol. 196: Advances in Cryptology: Proceedings of CRYPTO 84, a Workshop on the Theory and Application of Cryptographic Techniques, Santa Barbara, CA, August 19-22, 1984, pp. 269-275. Berlin/New York: 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 Transactions on Computers, Vol. C-36, No. 12, December 1987, pp. 1523-1538.

[BANE82] S. K. Banerjee, "High speed implementation of DES," Computers & Security, Vol. 1, No. 3, November 1982, pp. 261-267.

[BART63] T. C. Bartee and D. I. Schneider, "Computation with Finite Fields," Information and Control, Vol. 6, 1963, pp. 79-98.

[BATC80] K. E. Batcher, "Design of a massively parallel processor," IEEE Transactions on Computers, Vol. C-29, No. 9, September 1980, pp. 836-840.

[BLAK84] I. F. Blake, R. Fuji-Hara, R. C. Mullin, and S. A. Vanstone, "Computing logarithms in finite fields of characteristic two," SIAM Journal on Algebraic and Discrete Methods, Vol. 5, No. 2, June 1984, pp. 276-285.

[BLAK84b] I. F. Blake, R. C. Mullin, and S. A. Vanstone, "Computing logarithms in GF(2n)," in G. R. Blakley and D. Chaum, Eds., Lecture Notes in Computer Science Vol. 196: Advances in Cryptology: Proceedings of CRYPTO 84, a Workshop on the Theory and Application of Cryptographic Techniques, Santa Barbara, CA, August 19-22, 1984, pp. 73-82. Berlin/New York: Springer-Verlag, 1985.

[BLAK83] G. R. Blakley, "A computer algorithm for calculating the product AB modulo M," IEEE Transactions on Computers, Vol. C-32, No. 5, May 1983, pp. 497-500.

[BLUM84] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," SIAM Journal on Computing, Vol. 13, No. 4, November 1984, pp. 850-864.

[BOOT81] K. S. Booth, "Authentication of signatures using public key encryption," Communications of the ACM, Vol. 24, No. 11, November 1981, pp. 772-774.

[BRAN75] D. K. Branstad, "Encryption protection in computer data communications," in Proceedings of the 4th Data Communications Symposium, October 7-9, 1975, pp. 8.1 - 8.7. IEEE.

[BRAS79] G. Brassard, "A note on the complexity of cryptography," IEEE Transactions on Information Theory, Vol. IT-25, No. 2, March 1979, pp. 232-233.

[BRAS83] G. Brassard, "Relativized cryptography," IEEE Transactions on Information Theory, Vol. IT-29, No. 6, November 1983, pp. 877-894.

[BRAS88] G. Brassard, Lecture Notes in Computer Science Vol. 325: Modern Cryptology: a Tutorial. Berlin/New York: Springer-Verlag, 1988.

[BREN83] R. P. Brent and H. T. Kung, "Systolic VLSI arrays for linear-time GCD computation," in F. Anceau and E. J. Aas, Eds., VLSI 83, proceedings of the IFIP TC 10/WG 10.5 International Conference on VLSI, 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 R. E. A. Mason, Ed., IFIP Congress Series Vol. 9: Information Processing 83, proceedings of the IFIP 9th World Congress, Paris, France, September 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 D. Chaum, R. L. rivest, and A. T. Sherman, Eds., Advances in Cryptology: proceedings of CRYPTO '82, a Workshop on the Theory and Application of Cryptographic Techniques, Santa Barbara, CA, August 23-25, 1982, pp. 51-60. New York: Plenum Press, 1983.

[BRIC83] E. F. Brickell, "Solving low density knapsacks," in D. Chaum, Ed., Advances in Cryptology: proceedings of CRYPTO 83, a Workshop on the Theory and Application of Cryptographic Techniques, Santa Barbara, CA, August 22-24, 1983, pp. 25-37. New York: Plenum Press, 1984.

[BRIC84] E. F. Brickell, "Breaking iterated knapsacks," in G. R. Blakley and D. Chaum, Eds., Lecture Notes in Computer Science Vol. 196: Advances in Cryptology: Proceedings of CRYPTO 84, a Workshop on the Theory and Application of Cryptographic Techniques, Santa Barbara, CA, August 19-22, 1984, pp. 342-358. Berlin/New York: Springer-Verlag, 1985.

[BRIC89] E. F. Brickell, "A survey of hardware implementations of RSA," in G. Brassard, Ed., Lecture Notes in Computer Science Vol. 435: Advances in Cryptology - Proceedings of CRYPTO '89, pp. 368-370. Berlin/New York: Springer-Verlag, 1990.

[BRIC88] E. F. Brickell and A. M. Odlyzko, "Cryptanalysis: a survey of recent results," Proceedings of the IEEE, Vol. 76, No. 5, May 1988, pp. 578-593.

[BRIG76] H. S. Bright and R. L. Enison, "Cryptography using modular software elements," in S. Winkler, Ed., AFIPS Conference Proceedings Vol. 45: National Computer Conference, New York, NY, June 7-10, 1976, pp. 113-123. Montvale, NJ: AFIPS Press, 1976.

[CCIT87] CCITT, Draft Recommendation X.509: The Directory - Authentication Framework. Gloucester, November 1987.

[CARO87] T. R. Caron and R. D. Silverman, "Parallel implementation of the quadratic scheme," Journal of Supercomputing, Vol. 1, No. 3, 1987.

[COHE87] H. Cohen and A. K. Lenstra, "Implementation of a new primality test," Mathematics of Computation, Vol. 48, No. 177, January 1987, pp. 103-121.

[COHE84] H. Cohen and H. W. Lenstra, Jr., "Primality testing and Jacobi sums," Mathematics of Computation, Vol. 42, No. 165, January 1984, pp. 297-330.

[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 Transactions on Information Theory, Vol. IT-30, No. 4, July 1984, pp. 587-594.

[COPP85] D. Coppersmith, "Another birthday attack," in H. C. Williams, Ed., Lecture Notes in Computer Science Vol. 218: Advances in Cryptology - CRYPTO '85, proceedings of a Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, CA, August 18-22, 1985, pp. 14-17. Berlin/New York: Springer-Verlag, 1986.

[COPP87] D. Coppersmith, "Cryptography," IBM Journal of Research

and Development, Vol. 31, No. 2, March 1987, pp. 244-248.

[COPP86] D. Coppersmith, A. M. Odlyzko, and R. Schroeppel, "Discrete logarithms in GF(p)," Algorithmica, Vol. 1, No. 1, 1986, pp. 1-15.

[CRAY80] Cray Research, Inc., Cray 1-S Series Hardware Reference Manual. Mendota Heights, MN, June 1980.

[CRAY85] Cray Research, Inc., Cray X-MP Series of Computer Systems. Mendota Heights, MN, August 1985.

[DAVI83] D. W. Davies, "Applying the RSA digital signature to electronic mail," Computer, Vol. 16, No. 2, February 1983, pp. 55-62.

[DAVI80] D. W. Davies and W. L. Price, "The application of digital signatures based on public key cryptosystems," NPL Report DNACS 39/80, National Physics Laboratory, Teddington, Middlesex, England, December 1980.

[DAVI83b] J. A. Davis and D. B. Holdridge, "Factorization using the quadratic sieve algorithm," in D. Chaum, Ed., Advances in Cryptology: proceedings of CRYPTO 83, a Workshop on the Theory and Application of Cryptographic Techniques, Santa Barbara, CA, August 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 T. Beth, N. Cot, and I. Ingemarsson, Eds., Lecture Notes in Computer Science Vol. 209: Advances in Cryptology: Proceedings of EUROCRYPT 84, a Workshop on the Theory and Application of Cryptographic Techniques, Paris, France, April 9-11, 1984, pp. 183-215. Berlin/New York: Springer-Verlag, 1985.

[DEMI83] R. DeMillo and M. Merritt, "Protocols for data security," Computer, Vol. 16, No. 2, February 1983, pp. 39-51.

[DENN79] D. E. Denning, "Secure personal computing in an insecure network," Communications of the ACM, Vol. 22, No. 8, August 1979, pp. 476-482.

[DENN83b] D. E. Denning, "Protecting public keys and signature keys," Computer, Vol. 16, No. 2, February 1983, pp. 27-35.

[DENN81] D. E. Denning and G. M. Sacco, "Timestamps in key distribution protocols," Communications of the ACM, Vol. 24, No. 8, August 1981, pp. 533-536.

[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 Transactions on Information Theory, Vol. IT-30, No. 4, July 1984, pp. 601-611.

[DIFF82] W. Diffie, "Conventional versus public key cryptosystems," in G. J. Simmons, Ed., Secure Communications and Asymmetric Cryptosystems, pp. 41-72. Boulder, CO: Westview Press, 1982.

[DIFF84] W. Diffie, "Network security problems and approaches," Proceedings of the National Electronics Conference, Vol. 38, 1984, pp. 292-314.

[DIFF86] W. Diffie, "Communication security and national security business, technology, and politics," Proceedings of the National Communications Forum, Vol. 40, 1986, pp. 734-751.

[DIFF88] W. Diffie, "The first ten years of public-key cryptography," Proceedings of the IEEE, Vol. 76, No. 5, May 1988, pp. 560-577.

[DIFF76] W. Diffie and M. E. Hellman, "Multiuser cryptographic techniques," in S. Winkler, Ed., AFIPS Conference Proceedings Vol. 45: National Computer Conference, New York, NY, June 7-10, 1976, pp. 109-112. Montvale, NJ: AFIPS Press, 1976.

[DIFF76b] W. Diffie and M. E. Hellman, "New directions in cryptography," IEEE Transactions on Information Theory, Vol. IT-22, No. 6, November 1976, pp. 644-654.

[DIFF79] W. Diffie and M. E. Hellman, "Privacy and authentication: an introduction to cryptography," Proceedings of the IEEE, Vol. 67, No. 3, March 1979, pp. 397-427.

[DIFF87] W. Diffie, B. O'Higgins, L. Strawczynski, and D. Steer, "An ISDN secure telephone unit," Proceedings of the National Communications Forum, Vol. 41, No. 1, 1987, pp. 473-477.

[DIXO81] J. D. Dixon, "Asymptotically fast factorization of integers," Mathematics of Computation, Vol. 36, No. 153, January 1981, pp. 255-260.

[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, October 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," presented at EUROCRYPT '90, Aarhuis, DEnmark, May 21-24, 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 Systems Journal, Vol. 17, No. 2, 1978, pp. 106-125.

[ELGA85] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," IEEE Transactions on Information Theory, Vol. IT-31, No. 4, July 1985, pp. 469-472.

[ELGA85b] T. ElGamal, "On computing logarithms over finite fields," in H. C. Williams, Ed., Lecture Notes in Computer Science Vol. 218: Advances in Cryptology - CRYPTO '85, proceedings of a Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, CA, August 18-22, 1985, pp. 396-402. Berlin/New York: Springer-Verlag, 1986.

[EVAN74] A. Evans Jr. and W. Kantrowitz, "A user authentication scheme not requiring secrecy in the computer," Communications of the ACM, Vol. 17, No. 8, August 1974, pp. 437-442.

[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, NY, 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 A. M. Odlyzko, Ed., Lecture Notes in Computer Science Vol. 263: Advances in Cryptology - CRYPTO '86, proceedings of a Conference on the

Theory and Applications of Cryptographic Techniques, Santa Barbara, CA, August 11-15, 1986, pp. 186-194. Berlin/New York: Springer-Verlag, 1987.

[FLYN78] R. Flynn and A. S. Campasano, "Data dependent keys for a selective encryption terminal," in S. P. Ghosh and L. Y. Liu, Eds., AFIPS Conference Proceedings Vol. 47: National Computer Conference, 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 Journal on Computing, Vol. 6, No. 4, December 1977, pp. 675-695.

[GOLD84] S. Goldwasser and S. Micali, "Probabilistic encryption," Journal of Computer and System Sciences, Vol. 28, No. 2, April 1984, pp. 270-299.

[GOLD89] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," SIAM Journal on Computing, Vol. 18, No. 1, February 1989, pp. 186-208.

[GOLD88] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," SIAM Journal on Computing, Vol. 17, No. 2, April 1988, pp. 281-308.

[GORD84b] J. Gordon, "Strong RSA keys," Electronics Letters, Vol. 20, No. 12, June 7, 1984, pp. 514-516.

[GORD84] J. A. Gordon, "Strong primes are easy to find," in T. Beth, N. Cot, and I. Ingemarsson, Eds., Lecture Notes in Computer Science Vol. 209: Advances in Cryptology: Proceedings of EUROCRYPT 84, a Workshop on the Theory and Application of Cryptographic Techniques, Paris, France, April 9-11, 1984, pp. 216-223. Berlin/New York: Springer-Verlag, 1985.

[GROL88] J. Grollman and A. L. Selman, "Complexity measures for public-key cryptosystems," SIAM Journal on Computing, Vol. 17, No. 2, April 1988, pp. 309-335.

[HAST88] J. Hastad, "Solving simultaneous modular equations of low

degree," SIAM Journal on Computing, Vol. 17, No. 2, April 1988, pp. 336-341.

[HAWK87] S. Hawkinson, "The FPS T Series: a parallel vector supercomputer," in W. J. Karplus, Ed., Multiprocessors and Array Processors, pp. 147-155. San Diego: 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, October 1986, pp. 6-17.

[HAYK88] M. E. Haykin and R. B. J. Warnar, "Smart card technology: new methods for computer access control," NIST Special Publication 500-157, September 1988.

[HENR81] P. S. Henry, "Fast decryption algorithm for the knapsack cryptographic system," Bell System Technical Journal, Vol. 60, No. 5, May-June 1981, pp. 767-773.

[HERS64] I. N. Herstein, Topics in Algebra. Waltham: 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: Adam Hilger, 1981.

[HORO78] E. Horowitz and S. Sahni, Fundamentals of Computer Algorithms. Rockville: Computer Science Press, 1978.

[HWAN84] K. Hwang and F. A. Briggs, Computer Architecture and Parallel Processing. New York: McGraw-Hill, 1984.

[IAB-90] IAB Privacy and Security Research Group, "Privacy enhancement for Internet electronic mail: Part I: Message encipherment and authentication procedures," RFC 1113B, December 18, 1990.

[ISO-87] International Organization for Standards, Draft International Standard ISO/DIS 7498-2, Information processing systems - Open Systems Interconnection Model - Part 2: Security Architecture, 1987.

[JUEN82] R. R. Jueneman, "Analysis of certain aspects of output feedback mode," in D. Chaum, R. L. Rivest, and A. T. Sherman, Eds., Advances in Cryptology - proceedings of CRYPTO 82, a Workshop on the Theory and Application of Cryptographic Techniques, Santa Barbara, CA, August 23-25, 1982, pp. 99-127. New York: Plenum Press, 1983.

[JUEN86] R. R. Jueneman, "A high speed manipulation detection code," in A. M. Odlyzko, Ed., Lecture Notes in Computer Science Vol. 263: Advances in Cryptology - CRYPTO '86, proceedings of a Conference on Theory and Applications of Cryptographic Techniques, Santa Barbara, CA, August 11-15, 1986, pp. 327-346. Berlin/New York: Springer-Verlag, 1987.

[KHAC79] L. G. Khacian, "A polynomial algorithm in linear programming," Dokl. Akad. Nauk. SSSR 244, pp. 1093-1096. English translation in Soviet Math. Dokl. 20, pp. 191-194.

[KLIN79] C. S. Kline and G. J. Popek, "Public key vs. conventional key encryption," in R. E. Merwin, Ed., AFIPS Conference Proceedings Vol. 48: National Computer Conference, June 4-7, 1979, New York, NY, 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," Communications of the ACM, Vol. 21, No. 2, February 1978, p. 179.

[KOHN78] L. M. Kohnfelder, "A method for certification," MIT Laboratory for Computer Science, Cambridge, MA, 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, January 1982, pp. 37-46.

[KUNG78] H. T. Kung and C. Leiserson, "Systolic arrays (for VLSI)," in I. S. Duff and G. W. Stewart, Eds., Sparse Matrix Proceedings, 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 Transactions on Computers, Vol. C-31, No. 11, November 1982, pp. 1054-1066.

[LAGA84] J. C. Lagarias, "Performance analysis of Shamir's attack on the basic Merkle-Hellman knapsack system," in J. Paredaens, Ed., Lecture Notes in Computer Science Vol. 172: Automata, Languages and Programming: 11th Colloquium, 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 Journal of the Association for Computing Machinery, Vol. 32, No. 1, January 1985, pp. 229-246.

[LAKS83] S. Lakshmivarahan, "Algorithms for public key cryptosystems: theory and application," Advances in Computers, Vol. 22, 1983, pp. 45-108.

[LAWS71] B. A. Laws, Jr. and C. K. Rushforth, "A cellular-array multiplier for GF(2m)," IEEE Transactions on Computers, Vol. 20, No. 12, December 1971, pp. 1573-1578.

[LEHM82] D. J. Lehman, "On primality tests," SIAM Journal on Computing, Vol. 11, No. 2, May 1982, pp. 374-375.

[LEHM76] D. H. Lehmer, "Strong Carmichael numbers," Journal of the Australian Mathematical Society, Vol. 21 (Series A), 1976, pp. 508-510.

[LEMP79] A. Lempel, "Cryptology in transition," ACM Computing Surveys, Vol. 11, No. 4, December 1979, pp. 285-303.

[LENS82] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovasz, "Factoring polynomials with rational coefficients," Mathematische

Annalen, Vol. 261, 1982, pp. 515-534.

[LENS90] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard, "The number field sieve."

[LENS89] A. K. Lenstra and M. S. Manasse, "Factoring by electronic mail," to appear in proceedings of EUROCRYPT '89.

[LENS83] H. W. Lenstra, Jr., "Integer programming with a fixed number of variables," Mathematics of Operations Research, Vol. 8, No. 4, November 1983, pp. 538-548.

[LENS86] H. W. Lenstra, Jr., "Primality testing," in J. W. de Bakker et al., Eds., Mathematics and Computer Science, CWI Monographs, I, pp. 269-287. Amsterdam/New York: North-Holland, 1986.

[LENS87] H. W. Lenstra, Jr., "Factoring integers with elliptic curves," Annals of Mathematics, Vol. 126, 1987, pp. 649-673.

[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 Proceedings of the 12th National Computer Security Conference, Baltimore, MD, October 10-13, 1989, pp. 215-229.

[MACM81] D. MacMillan, "Single chip encrypts data at 14 Mb/s," Electronics, Vol. 54, No. 12, June 16, 1981, pp. 161-165.

[MASS88] J. L. Massey, "An introduction to contemporary cryptology," Proceedings of the IEEE, Vol. 76, No. 5, May 1988, pp. 533-549.

[MATY78] S. M. Matyas and C. H. Meyer, "Generation, distribution, and installation of cryptographic keys," IBM Systems Journal, Vol. 17, No. 2, 1978, pp. 126-137.

[MCCU89] K. S. McCurley, "The discrete logarithm problem," preprint.

[MCEL78] R. J. McEliece, "A public-key cryptosystem based on algebraic coding theory," DSN Progress Report 42-44, Jet Propulsion Laboratory, 1978, pp. 114-116.

[MERK78] R. C. Merkle, "Secure communications over insecure channels," Communications of the ACM, Vol. 21, No. 4, April 1978, pp. 294-299.

[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 G. J. Simmons, Ed., Secure Communications and Asymmetric Cryptosystems, pp. 73-104. Boulder, CO: Westview Press, 1982.

[MERK89] R. C. Merkle, "One way hash functions and DES," preprint.

[MERK78b] R. C. Merkle and M. E. Hellman, "Hiding information and signatures in trapdoor knapsacks," IEEE Transactions on Information Theory, Vol. 24, No. 5, September 1978, pp. 525-530.

[MILL76] G. L. Miller, "Riemann's hypothesis and tests for primality," Journal of Computer and System Sciences, Vol. 13, No. 3, December 1976, pp. 300-317.

[MILU88] V. M. Milutinovic, Computer Architecture: Concepts and Systems. New York: North-Holland, 1988.

[MONT85] P. L. Montgomery, "Modular multiplication without trial division," Mathematics of Computation, Vol. 44, No. 170, April 1985, pp. 519-521.

[MOOR88] J. H. Moore, "Protocol failures in cryptosystems," Proceedings of the IEEE, Vol. 76, No. 5, May 1988, pp. 594-602.

[MORR75] M. A. Morrison and J. Brillhart, "A method of factoring and the factorization of F7," Mathematics of Computation, Vol. 29, No. 129, January 1975, pp. 183-205.

[NATI77] National Bureau of Standards, Federal Information Processing Standards Publication 46: Data Encryption Standard, January 15, 1977.

[NATI80] National Bureau of Standards, Federal Information Processing Standards Publication 81: DES Modes of Operation, December 2, 1980.

[NATI81] National Bureau of Standards, Federal Information Processing Standards Publication 74: Guidelines for Implementing and Using the NBS Data Encryption Standard, April 1, 1981.

[NEED78] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," Communications of the ACM, Vol. 21, No. 12, December 1978, pp. 993-999.

[ODLY84] A. M. Odlyzko, "Cryptanalytic attacks on the multiplicative knapsack cryptosystem and on Shamir's fast signature scheme," IEEE Transactions on Information Theory, Vol. IT-30, No. 4, July 1984, pp. 594-601.

[ODLY84b] A. M. Odlyzko, "Discrete logarithms in finite fields and their cryptographic significance," in T. Beth, N. Cot, and I. Ingemarsson, Eds., Lecture Notes in Computer Science Vol. 209: Advances in Cryptology: Proceedings of EUROCRYPT 84, a Workshop on the Theory and Application of Cryptographic Techniques, Paris, France, April 9-11, 1984, pp. 224-314. Berlin/New York: 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 A. M. Odlyzko, Ed., Lecture Notes in Computer Science Vol. 263: Advances in Cryptology - CRYPTO '86, proceedings of a Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, CA, August 11-15, 1986, pp. 277-301. Berlin/New York: 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 Transactions on Information Theory, Vol. 32, No. 6, November 1986, pp. 846-847.

[POHL78] S. C. Pohlig and M. E. Hellman, "An improved algorithm for computing logarithms over GF(p) and its cryptographic significance," IEEE Transactions on Information Theory, Vol. IT-24, No. 1, January 1978, pp. 106-110.

[POME84] C. Pomerance, "The quadratic sieve factoring algorithm,"

in T. Beth, N. Cot, and I. Ingemarsson, Eds., Lecture Notes in Computer Science Vol. 209: Advances in Cryptology: proceedings of EUROCRYPT 84, a Workshop on the Theory and Application of Cryptographic Techniques, Paris, France, April 9-11, 1984, pp. 169-182. Berlin/New York: Springer-Verlag, 1985.

[POME86] C. Pomerance, "Fast, rigorous factorization and discrete logarithm algorithms," in D. S. Johnson, T. Nishizeki, A. Nozaki, and H. S. Wilf, Eds., Discrete Algorithms and Complexity, proceedings of the Japan-US Joint Seminar, Kyoto, Japan, 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 Journal on Computing, Vol. 17, No. 2, April 1988, pp. 387-403.

[POPE78] G. J. Popek and C. S. Kline, "Encryption protocols, public key algorithms and digital signatures in computer networks," in R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton, Eds., Foundations of Secure Computation, pp. 133-153. New York: Academic Press, 1978.

[POPE79] G. L. Popek and C. S. Kline, "Encryption and secure computer networks," ACM Computing Surveys, Vol. 11, No. 4, December 1979, pp. 331-356.

[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," Electronics Letters, Vol. 18, No. 21, October 14, 1982, pp. 905-907.

[RABI76] M. O. Rabin, "Probabilistic algorithms," in J. F. Traub, Ed., Algorithms and Complexity: New Directions and Recent Results, proceedings of a Symposium, Pittsburgh, PA, April 7-9, 1976, pp. 21-39. New York: Academic Press, 1976.

[RABI78] M. O. Rabin, "Digitalized signatures," in R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton, Eds., Foundations of Secure Computation, pp. 155-168. New York: Academic Press, 1978.

[RABI79] M. O. Rabin, "Digitalized signatures and public-key

functions as intractable as factorization," MIT Laboratory for Computer Science, Technical Report LCS/TR-212, January 1979.

[RABI80] M. O. Rabin, "Probabilistic algorithms for testing primality," Journal of Number Theory, Vol. 12, 1980, pp. 128-138.

[RADE64] H. Rademacher, Lectures on Elementary Number Theory. New York: Blaisdell, 1964.

[RIVE84] R. L. Rivest, "RSA chips (past/present/future)," in T. Beth, N. Cot, and I. Ingemarsson, Eds., Lecture Notes in Computer Science Vol. 209: Advances in Cryptology: proceedings of EUROCRYPT 84, a Workshop on the Theory and Application of Cryptographic Techniques, Paris, France, April 9-11, 1984, pp. 159-165. Berlin/New York: Springer-Verlag, 1985.

[RIVE90] R. L. Rivest, "The MD4 message digest algorithm," February 1990.

[RIVE78] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Communications of the ACM, Vol. 21, No. 2, February 1978, pp. 120-127.

[RIVE82] R. L. Rivest and A. T. Sherman, "Randomized encryption techniques," in D. Chaum, R. L. Rivest, and A. T. Sherman, Eds., Advances in Cryptology: Proceedings of CRYPTO 82, a Workshop on the Theory and Application of Cryptographic Techniques, Santa Barbara, CA, August 23-25, 1982, pp. 145-163. New York: Plenum Press, 1983.

[SIAM90] "Number field sieve produces factoring breakthrough," SIAM News, Vol. 23, No. 4, July 1990.

[SALO85] Salomaa, "Cryptography," in A. Salomaa, Encyclopedia of Mathematics and its Applications Vol. 25: Computation and Automata, pp. 186-230. Cambridge, UK: Cambridge University Press, 1985.

[SCHA82] B. P. Schanning, "Applying public key distribution to local area networks," Computers & Security, Vol. 1, No. 3, November 1982, pp. 268-274.

[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, October 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," Mathematics of Computation, Vol. 43, No. 167, July 1984, pp. 289-311.

[SEDL87] H. Sedlak, "The RSA Cryptography Processor," in D. Chaum and W. L. Price, Eds., Lecture Notes in Computer Science Vol. 304: Advances in Cryptology - EUROCRYPT '87, a Workshop on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 13-15, 1987, pp. 95-105. Berlin/New York: Springer-Verlag, 1988.

[SEIT85] C. L. Seitz, "The Cosmic Cube," Communications of the ACM, Vol. 28, No. 1, January 1985, pp. 22-33.

[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 G. R. Blakley and D. Chaum, Eds., Lecture Notes in Computer Science Vol. 196: Advances in Cryptology: Proceedings of CRYPTO 84, a Workshop on the Theory and Application of Cryptographic Techniques, Santa Barbara, CA, August 19-22, 1984, pp. 47-53. Berlin/New York: Springer-Verlag, 1985.

[SHAM84] A. Shamir, "A polynomial-time algorithm for breaking the basic Merkle-Hellman cryptosystem," IEEE Transactions on Information Theory, Vol. IT-30, No. 5, September 1984, pp. 699-704.

[SHAN90] M. Shand, P. Bertin, and J. Vuillemin, "Hardware speedups in long integer multiplication," presented at the Second ACM Symposium on Parallel Algorithms and Architectures, Crete, July 2-6, 1990.

[SILV87] R. D. Silverman, "The multiple polynomial quadratic sieve," Mathematics of Computation, Vol. 48, No. 177, January 1987, pp. 329-339.

[SIMM79] G. J. Simmons, "Symmetric and asymmetric encryption," ACM Computing Surveys, Vol. 11, No. 4, December 1979, pp. 305-330.

[SIMM88] G. J. Simmons, "How to ensure that data acquired to verify treaty compliance are trustworthy," Proceedings of the IEEE, Vol. 76, No. 5, May 1988, pp. 621-627.

[SMID81] M. E. Smid, "Integrating the data encryption standard into computer networks," IEEE Transactions on Computers, Vol. COM-29, No. 6, June 1981, pp. 762-772.

[SMID88b] M. E. Smid and D. K. Branstad, "The Data Encryption Standard: past and future," Proceedings of the IEEE, Vol. 76, No. 5, May 1988, pp. 550-559.

[SOL077] R. Solovay and V. Strassen, "A fast Monte-Carlo test for primality," SIAM Journal on Computing, Vol. 6, No. 1, March 1977, pp. 84-85. Erratum: Ibid., Vol. 7, No. 1, February 1978, p. 118.

[STEI86] L. K. Steiner, "The ETA-10 supercomputer system," in Proceedings of the 1986 IEEE International Conference on Computer Design, Port Chester, NY, October 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 G. R. Blakley and D. Chaum, Eds., Lecture Notes in Computer Science Vol. 196: Advances in Cryptology - Proceedings of CRYPTO 84, a Workshop on the Theory and Applications of Cryptographic Techniques, Santa Barbara, CA, August 19-22, 1984, pp. 19-36. Berlin/New York: 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(2m)," IEEE Transactions on Computers, Vol. C-34, No. 8, August 1985, pp. 709-717.

[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 Transactions on Information Theory, Vol. IT-26, No. 6, November 1980, pp. 726-729.

[WILL87] H. C. Williams and M. C. Wunderlich, "On the parallel generation of the residues for the continued fraction factoring algorithm," Mathematics of Computation, Vol. 48, No. 177, January 1987, pp. 405-423.

[WUND83] M. C. Wunderlich, "Factoring numbers on the Massively Parallel computer," in D. Chaum, Ed., Advances in Cryptology - proceedings of CRYPTO 83, a Workshop on the Theory and Applications of Cryptographic Techniques, Santa Barbara, CA, August 22-24, 1983, pp. 87-102. New York: Plenum Press, 1984.

[WUND85] M. C. Wunderlich, "Implementing the continued fraction factoring algorithm on parallel machines," Mathematics of Computation, Vol. 44, No. 169, January 1985, pp. 251-260.

[YAO-82] A. C. Yao, "Theory and applications of trapdoor functions," in 23rd Annual Symposium on Foundations of Computer Science, Chicago, IL, November 3-5, 1982, pp. 80-91. IEEE Computer Society Press, 1982.
□