RSA Laboratories

Answers to

Frequently Asked Questions About Today's Cryptography

RSA

LABORATORIES

CRYPTOGRAPHIC RESEARCH AND CONSULTATION

VERSION 3.0

Foreword

This document is the third version of RSA Laboratories' popular publication, *Frequently Asked Questions About Today's Cryptography*. Completely revised and updated, it covers twice as many questions as the second version, which was published in September 1993.

The FAQ represents the contributions of numerous individuals. Particular appreciation is due to Paul Fahn, who wrote the first and second versions while an RSA Laboratories research assistant in 1992 and 1993, and to Sambasivam Valliappan, who drafted the third version as an RSA Laboratories research assistant in Summer 1995.

Other contributors to the third version include Jim Bidzos, Victor Chang, Dana Ellingen, Patrick Lee, Hoa Ly, Paul Livesay, Tim Matthews, and Kurt Stammberger from RSA Data Security, Michael Baum from VeriSign, Stuart Haber from Bellcore, Bart Preneel from Katholieke Universiteit, and Scott Stornetta from Surety Technologies. Special thanks to Patrick Lee for carefully reading though the final draft and providing many helpful suggestions.

The technical editors are RSA Laboratories' chief scientist Burton S. Kaliski Jr., and research scientists Matthew J.B. Robshaw and Yiqun Lisa Yin.

Comments on the FAQ are encouraged. Address correspondence to:

FAQ Editor
RSA Laboratories
100 Marine Parkway, Suite 500
Redwood City, CA 94065-1031 USA
Tel: 415/595-7703
Fax: 415/595-4126
E-mail: `<faq-editor@rsa.com>`
Web: `http://www.rsa.com/rsalabs/`

CONTENTS

FIGURES AND TABLES

INTRODUCTION

1. What is Cryptography?

Cryptography, to most people, is concerned with keeping communications private. Indeed, the protection of sensitive communications has been the emphasis of cryptography throughout much of its history [Kah67]. As we will see, however, this is only one part of today's cryptography.

Encryption is the transformation of data into some unreadable form. Its purpose is to ensure privacy by keeping the information hidden from anyone for whom it is not intended, even those who can see the encrypted data. *Decryption* is the reverse of encryption ; it is the transformation of encrypted data back into some intelligible form.

Encryption and decryption require the use of some secret information, usually referred to as a *key*. Depending on the encryption mechanism used, the same key might be used for both encryption and decryption, while for other mechanisms, the keys used for encryption and decryption might be different (see Question 3).

But today's cryptography is more than secret writing, more than encryption and decryption. *Authentication* is as fundamental a part of our lives as privacy. We use authentication though out our everyday life, for instance when we sign our name to some document. As we move to a world where our decisions and agreements are communicated electronically, we need to replicate these procedures.

Cryptography provides mechanisms for such procedures. A *digital signature* (see Question 3) binds a document to the possessor of a particular key, while a *digital timestamp* (see Question 108) binds a document to its creation at a particular time. These cryptographic mechanisms can be used to control access to a shared disk drive, a high security installation or to a pay-per-view TV channel.

But the field of cryptography contains even more when we include some of the things cryptography enables us to do. With just a few basic tools it is possible to build elaborate schemes and protocols which allow us to pay using *electronic money* (see

Question 138), to prove we know certain information without revealing the information itself (see Question 107), and to share a secret quantity in such a way that no fewer than three from a pool of five people (for instance) can reconstruct the secret (see Question 103).

While modern cryptography is growing increasingly diverse, cryptography is fundamentally based on problems that are difficult to solve. A problem may be difficult because its solution requires some secret knowledge, such as decrypting an encrypted message or signing some digital document, or the problem may be hard because it is intrinsically difficult to complete, such as finding a message which produces a given hash value.

So as the field of cryptography has advanced, the dividing lines for what is and what is not cryptography have become blurred. Cryptography today might be summed up as the study of techniques and applications that depend on the existence of difficult problems. A *cryptanalyst* attempts to compromise cryptographic mechanisms, and *cryptology* (from the Greek *kryptós lógos,* meaning "hidden word") is the discipline of cryptography and cryptanalysis combined.

Surveys by Rivest [Riv90] and Brassard [Bra88] form an excellent introduction to modern cryptography. Some textbook treatments are provided by Stinson [Sti95] and Stallings [Sta95], while Simmons provides an in-depth coverage of the technical aspects of cryptography [Sim92]. A comprehensive review of modern cryptography can also be found in *Applied Cryptography* [Sch95b]; Ford [For94] provides detailed coverage of issues such as cryptography standards and secure communication.

2. What is the RSA Laboratories Frequently Asked Questions?

In the *RSA Laboratories Frequently Asked Questions About Today's Cryptography,* we provide answers to a wide variety of questions. Modern cryptography is a big subject (see Question 1) and it inspires a great many questions. We have not attempted to be, nor could we be, exhaustive in answering every possible one. Instead, we have answered some of the more important questions about some of today's better-known algorithms and techniques.

The questions are organized according to a fairly standard division. We consider cryptographic algorithms as basic building blocks and we treat questions about them first. Questions on using those algorithms in security protocols and services follow. We conclude with questions on cryptography's interaction with the real world through government involvement, standards, and products.

Part One

ALGORITHMS AND TECHNIQUES

PUBLIC-KEY ENCRYPTION AND DIGITAL SIGNATURES

General

3. What is Public-Key Cryptography?

Traditional cryptography is based on the sender and receiver of a message knowing and using the same secret key: the sender uses the secret key to encrypt the message, and the receiver uses the same secret key to decrypt the message. This method is known as *secret-key* or *symmetric cryptography*. The main problem is getting the sender and receiver to agree on the secret key without anyone else finding out. If they are in separate physical locations, they must trust a courier, or a phone system, or some other transmission medium to prevent the disclosure of the secret key being communicated. Anyone who overhears or intercepts the key in transit can later read, modify, and forge all messages encrypted or authenticated using that key. The generation, transmission and storage of keys is called key management; all cryptosystems must deal with key management issues. Because all keys in a secret-key cryptosystem must remain secret, secret-key cryptography often has difficulty providing secure key management, especially in open systems with a large number of users.

The concept of *public-key cryptography* was introduced in 1976 by Whitfield Diffie and Martin Hellman [DH76] in order to solve the key management problem. In their concept, each person gets a pair of keys, one called the *public key* and the other called the *private key*. Each person's public key is published while the private key is kept secret. The need for the sender and receiver to share secret information is eliminated; all communications involve only public keys, and no private key is ever transmitted or shared. No longer is it necessary to trust some communications channel to be secure against eavesdropping or betrayal. The only requirement is that public keys are associated with their users in a trusted (authenticated) manner (for instance, in a trusted directory). Anyone can send a confidential message by just using public information, but the message can only be decrypted with a private key, which is in the sole possession of the intended recipient. Furthermore, public-key cryptography can be used not only for privacy (*encryption*), but also for authentication (*digital signatures*).

Encryption

When Alice wishes to send a secret message to Bob, she looks up Bob's public key in a directory, uses it to encrypt the message and sends it off. Bob then uses his private key to decrypt the message and read it. No one listening in can decrypt the message. Anyone can send an encrypted message to Bob but only Bob can read it. Clearly, one requirement is that no one can figure out the private key from the corresponding public key.

Digital Signatures

To sign a message, Alice does a computation involving both her private key and the message itself; the output is called the digital signature and is attached to the message, which is then sent. Bob, to verify the signature, does some computation involving the message, the purported signature, and Alice's public key. If the result properly holds in a simple mathematical relation, the signature is verified as being genuine; otherwise, the signature may be fraudulent or the message might have been altered.

A good history of public-key cryptography is given by Diffie [Dif88].

4. What are the Advantages and Disadvantages of Public-Key Cryptography Compared with Secret-Key Cryptography?

The primary advantage of public-key cryptography is increased security and convenience. Private keys never need to transmitted or revealed to anyone. In a secret-key system, by contrast, the secret keys must be transmitted (either manually or through a communication channel), and there may be a chance that an enemy can discover the secret keys during their transmission.

Another major advantage of public-key systems is that they can provide a method for digital signatures. Authentication via secret-key systems requires the sharing of some secret and sometimes requires trust of a third party as well. As a result, a sender can repudiate a previously authenticated message by claiming that the shared secret was somehow compromised (see Question 113) by one of the parties sharing the secret. For example, the Kerberos secret-key authentication system (see Question 144) involves a central database that keeps copies of the secret keys of all users; an attack on the database would allow widespread forgery. Public-key authentication, on the other hand, prevents this type of repudiation; each user has sole responsibility for protecting his or her private key. This property of public-key authentication is often called *non-repudiation.*

A disadvantage of using public-key cryptography for encryption is speed; there are popular secret-key encryption methods that are significantly faster than any currently available public-key encryption method. Nevertheless, public-key cryptography can be used with secret-key cryptography to get the best of both worlds. For encryption, the best solution is to combine public- and secret-key systems in order to get both the security advantages of public-key systems and the speed advantages of secret-key systems. The public-key system can be used to encrypt a secret key which is used to encrypt the bulk of a file or message. Such a protocol is called a *digital envelope,* which is explained in more detail in Question 16 in the case of RSA.

Public-key cryptography may be vulnerable to impersonation, however, even if users' private keys are not available. A successful attack on a certification authority (see Question 127) will allow an adversary to impersonate whomever the adversary chooses to by using a public-key certificate from the compromised authority to bind a key of the adversary's choice to the name of another user.

In some situations, public-key cryptography is not necessary and secret-key cryptography alone is sufficient. This includes environments where secure secret-key agreement can take place, for example by users meeting in private. It also includes environments where a single authority knows and manages all the keys (e.g., a closed banking system). Since the authority knows everyone's keys already, there is not much advantage for some to be "public" and others "private." Also, public-key cryptography is usually not necessary in a single-user environment. For example, if you want to keep your personal files encrypted, you can do so with any secret-key encryption algorithm using, say, your personal password as the secret key. In general, public-key cryptography is best suited for an open multi-user environment.

Public-key cryptography is not meant to replace secret-key cryptography, but rather to supplement it, to make it more secure. The first use of public-key techniques was for secure key exchange in an otherwise secret-key system [DH76]; this is still one of its primary functions. Secret-key cryptography remains extremely important and is the subject of ongoing study and research. Some secret-key cryptosystems are discussed in the sections on Block Ciphers and Stream Ciphers.

5. Do Digital Signatures Help Detect Altered Documents and Transmission Errors?

A digital signature is superior to a handwritten signature in that it attests to the contents of a message as well as to the identity of the signer. As long as a secure hash function (see Question 94) is used, there is no way to take someone's signature from one document and attach it to another or to alter a signed message in any way. The slightest change in a signed document will cause the digital signature verification process to fail. Thus, public-key authentication allows people to check the integrity of signed documents. If a signature verification fails, however, it is generally difficult to determine whether there was an attempted forgery or simply a transmission error.

6. What is a One-Way Function?

A *one-way function* is a mathematical function that is significantly easier to perform in one direction (the forward direction) than in the opposite direction (the inverse direction). It might be possible, for example, to compute the function in seconds but to compute its inverse could take months or years. A *trap-door one-way function* is a one-way function where the inverse direction is easy, given a certain piece of information (the trap door), but difficult otherwise.

7. What is the Significance of One-Way Functions for Cryptography?

Public-key cryptosystems are based on (presumed) trap-door one-way functions. The public key gives information about the particular instance of the function; the private key gives information about the trap door. Whoever knows the trap door can perform the function easily in both directions, but anyone lacking the trap door can perform the function only in the forward direction. The forward direction is used for encryption and signature verification; the inverse direction is used for decryption and signature generation.

In almost all public-key systems the size of the key corresponds to the size of the inputs to the one-way function. The larger the key, the greater the difference between the efforts necessary to compute the function in the forward and inverse directions (for someone lacking the trap door). For a digital signature to be secure for years, for example, it is necessary to use a trap-door one-way function with inputs large enough that someone without the trap door would need many years to compute the inverse function.

All practical public-key cryptosystems are based on functions that are believed to be one-way, but no function has been proven to be so. This means that it is theoretically possible that an algorithm will be discovered that can compute the inverse function easily without a trap door. This development would render any cryptosystem based on that one-way function insecure and useless. On the other hand, further research in theoretical computer science may result in concrete lower bounds on the difficulty of inverting certain functions, and this would be a landmark event with significant positive ramifications for cryptography.

RSA

8. What is RSA?

RSA is a public-key cryptosystem for both encryption and authentication; it was invented in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman [RSA78]. It works as follows: take two large primes, p and q, and find their product $n = pq$; n is called the modulus. Choose a number, e, less than n and relatively prime to $(p-1)(q-1)$, which means that e and $(p-1)(q-1)$ have no common factors except 1. Find another number d such that $(ed - 1)$ is divisible by $(p-1)(q-1)$. The values e and d are called the public and private exponents, respectively. The public key is the pair (n,e); the private key is (n,d). The factors p and q maybe kept with the private key, or destroyed.

It is difficult (presumably) to obtain the private key d from the public key (n,e). If one could factor n into p and q, however, then one could obtain the private key d. Thus the security of RSA is related to the assumption that factoring is difficult. An easy factoring method or some other feasible attack would “break” RSA (see Question 10 and Question 46).

Here is how RSA can be used for privacy and authentication (in practice, the actual use is slightly different; see Question 16 and Question 17):

> RSA privacy encryption: Suppose Alice wants to send a message m to Bob. Alice creates the ciphertext c by exponentiating: $c = m^e \bmod n$, where e and n are Bob’s public key. She sends c to Bob. To decrypt, Bob also exponentiates: $m = c^d \bmod n$; the relationship between e and d ensures that Bob correctly recovers m. Since only Bob knows d, only Bob can decrypt.
>
> RSA authentication: Suppose Alice wants to send a message m to Bob in such away that Bob is assured that the message is authentic and is from Alice. Alice creates a digital signature s by exponentiating: $s = m^d \bmod n$, where d and n are Alice’s private key. She sends m and s to Bob. To verify the signature, Bob exponentiates and checks that the message m is recovered: $m = s^e \bmod n$, where e and n are Alice’s public key.

Thus encryption and authentication take place without any sharing of private keys: each person uses only other people’s public keys and his or her own private key. Anyone can send an encrypted message or verify a signed message, using only public keys, but only someone in possession of the correct private key can decrypt or sign a message.

9. How Fast is RSA?

An “RSA operation,” whether for encrypting or decrypting, signing or verifying, is essentially a modular exponentiation, which can be performed by a series of modular multiplications.

In practical applications, it is common to choose a small public exponent for the public key; in fact, entire groups of users can use the same public exponent, each with a different modulus. There are some restrictions on the prime factors of the modulus when the public exponent is fixed. This makes encryption faster than decryption and verification faster than signing. With typical modular exponentiation algorithms, public-key operations take $O(k^2)$ steps, private-key operations take $O(k^3)$ steps, and key generation takes $O(k^4)$ steps, where k is the number of bits in the modulus. (O-notation refers to the upper bound on the asymptotic running time of an algorithm [CLR90].) “Fast multiplication” techniques, such as FFT-based methods, require asymptotically fewer steps, though in practice they are not as common due to their great software complexity and the fact that they may actually be slower for typical key sizes.

There are many commercially available software and hardware implementations of RSA, and there are frequent announcements of newer and faster chips. On a 90 MHz Pentium, RSA Data Security’s cryptographic toolkit BSAFE 3.0 (see Question 173) has a throughput for private-key operations of 21.6 Kbits per second with a 512-bit modulus and 7.4 Kbits per second with a 1024-bit modulus. The fastest RSA hardware [SV93] has a throughput greater than 300 Kbits per second with a 512-bit modulus, implying that it performs over 500 RSA private-key operations per second. There is room in that hardware to execute two RSA 512-bit RSA operations in parallel [Sha95], hence the 600 Kbits/s speed reported in [SV93]. For 970-bit keys, the throughput is 185 Kbits/s. It is expected that RSA speeds will reach 1 Mbits/second within a year or so.

By comparison, DES is much faster than RSA. In software, DES is generally at least 100times as fast as RSA. In hardware, DES is between 1,000 and 10,000 times as fast, depending on the implementation. Implementations of RSA will probably narrow the gap a bit in coming years, as there are growing commercial markets, but DES will get faster as well.

For a detailed report on high-speed RSA implementations see [Koc94].

10. What Would it Take to Break RSA?

There are a few possible interpretations of "breaking RSA." The most damaging would be for an attacker to discover the private key corresponding to a given public key; this would enable the attacker both to read all messages encrypted with the public key and to forge signatures. The obvious way to do this attack is to factor the public modulus, n, into its two prime factors, p and q. From p, q, and e, the public exponent, the attacker can easily get d, the private exponent. The hard part is factoring n; the security of RSA depends on factoring being difficult. In fact, the task of recovering the private key is equivalent to the task of factoring the modulus: you can use d to factor n, as well as use the factorization of n to find d (see Question 47 and Question 48 regarding the state of the art in factoring). It should be noted that hardware improvements alone will not weaken RSA, as long as appropriate key lengths are used; in fact, hardware improvements should increase the security of RSA (see Question 47).

Another way to break RSA is to find a technique to compute e-th roots mod n. Since $c= m^e$ mod n, the e-th root of c mod n is the message m. This attack would allow someone to recover encrypted messages and forge signatures even without knowing the private key. This attack is not known to be equivalent to factoring. No general methods are currently known that attempt to break RSA in this way. However, in special cases where multiple related messages are encrypted with the same small exponent, it may be possible to recover the messages.

The attacks just mentioned are the only ways to break RSA in such a way as to be able to recover all messages encrypted under a given key. There are other methods, however, that aim to recover single messages; success would not enable the attacker to recover other messages encrypted with the same key. Some people also studied whether part of the message can be recovered from an encrypted message [ACG84].

The simplest single-message attack is the *guessed plaintext attack*. An attacker sees a ciphertext, guesses that the message might be "Attack at dawn," and encrypts this guess with the public key of the recipient; by comparison with the actual ciphertext, the attacker knows whether or not the guess was correct. This attack can be thwarted by appending some random bits to the message. Another single-message attack can occur if someone sends the same message m to three others, who each have public exponent $e = 3$. An attacker who knows this and sees the three messages will be able to recover the message m; this attack and ways to prevent it are discussed by Hastad [Has88]. Fortunately, this attack can also be defeated by padding the message before each encryption with some random bits. There are also some *chosen ciphertext attacks* (or *chosen message attacks* for signature forgery, in which the attacker creates some ciphertext and gets to see the corresponding plaintext, perhaps by tricking a legitimate user into decrypting a fake message; Davida [Dav82] and Desmedt and Odlyzko [DO86] give some examples.

For a survey of these and other attacks on RSA, see [KR95c].

Of course, there are also attacks that aim not at RSA itself but at a given insecure implementation of RSA; these do not count as "breaking RSA" because it is not any weakness in the RSA algorithm that is exploited, but rather a weakness in a specific implementation. For example, if someone stores a private key insecurely, an attacker may discover it. One cannot emphasize strongly enough that to be truly secure RSA requires a secure implementation; mathematical security measures, such as choosing a long key size, are not enough. In practice, most successful attacks will likely be aimed at insecure implementations and at the key management stages of an RSA system. SeeSection III.A for discussion of secure key management in an RSA system.

11. Are Strong Primes Necessary in RSA?

In the literature pertaining to RSA, it has often been suggested that in choosing a key pair, one should use so-called "strong" primes *p* and *q* to generate the modulus *n*. Strong primes are those with certain properties that make the product *n* hard to factor by specific factoring methods; such properties have included, for example, the existence of a large prime factor of *p*-1 and a large prime factor of *p*+1. The reason for these concerns is that some factoring methods are especially suited to primes *p* such that *p*-1 or *p*+1 has only small factors; strong primes are resistant to these attacks.

However, advances in factoring over the last ten years (see Question 48) appear to have obviated the advantage of strong primes; the elliptic curve factoring algorithm is one such advance. The new factoring methods have as good a chance of success on strong primes as on "weak" primes. Therefore, choosing traditional "strong" primes alone does not significantly increase security. Choosing large enough primes is what matters. However, there is no danger in using strong, large primes, though it may take slightly longer to generate a strong prime than an arbitrary prime.

It is possible that new factoring algorithms may be developed in the future which once again target primes with certain properties; if so, choosing strong primes may once again help to increase security.

12. How Large a Modulus (Key) Should be Used in RSA?

The best size for an RSA modulus depends on one's security needs. The larger the modulus, the greater the security, but also the slower the RSA operations. One should choose a modulus length upon consideration, first, of one's security needs, such as the value of the protected data and how long it needs to be protected, and, second, of how powerful one's potential enemies are.

A good analysis of the security obtained by a given modulus length is given by Rivest [Riv92a], in the context of discrete logarithms modulo a prime, but it applies to RSA as well. A more recent study of RSA key-size security can be found in a paper by Odlyzko [Odl95]. Odlyzko's paper considers the security of RSA key sizes based on factoring techniques available in 1995 and the ability to tap large computational resources via computer networks. A specific assessment of the security of 512-bit RSA keys shows that one may be factored for less than $1,000,000 in cost and eight months of effort in 1997 [Rob95d]. It is believed that 512-bit keys no longer provide sufficient security with the advent of new factoring algorithms and distributed computing. Such keys should not be used after 1997 or 1998. The RSA Laboratories recommended key sizes are now 768 bits for personal use, 1024 bits for corporate use, and 2048 bits for extremely valuable keys like the key pair of a certifying authority (see Question 123). A 768-bit key is expected to be secure until at least the year 2004.

The key of an individual user may expire after a certain time, say, two years (see Question 118). This gives an opportunity to change keys regularly and thus maintain a given level of security. Upon expiration, the user should generate a new key which is at least a few digits longer than the old key to reflect the speed increases of computers and factoring algorithms over the two years. Recommended key length schedules are published by RSA Laboratories on a regular basis.

Users should keep in mind that the estimated times to break RSA are averages only. A large factoring effort, attacking many thousands of RSA moduli, may succeed in factoring at least one in a reasonable time. Although the security of any individual key is still strong, with some factoring methods there is always a small chance that the attacker may get lucky and factor some key quickly.

As for the slowdown caused by increasing the key size (see Question 9), doubling the modulus length will, on average, increase the time required for public-key operations (encryption and signature verification) by a factor of four, and increase the time taken by private-key operations (decryption and signing) by a factor of eight. This assumes typical methods for RSA implementation, not “fast multiplication.” The reason that public-key operations are affected less than private-key operations is that the public exponent can remain fixed when the modulus is increased, whereas the private exponent increases proportionally. Key generation time would increase by a factor of 16 upon doubling the modulus, but this is a relatively infrequent operation for most users. The impact of key size increases other than doubling can be calculated similarly.

13. How Large Should the Primes be?

The two primes, p and q, which compose the modulus, should be of roughly equal length; this will make the modulus harder to factor than if one of the primes was very small. Thus if one chooses to use a 768-bit modulus, the primes should each have length approximately 384 bits. If the two primes are extremely close (identical except for, say, 100 - 200 bits), there is a potential security risk, but the probability that two randomly chosen primes are so close is negligible.

14. Can Users of RSA run out of Distinct Primes?

There are enough prime numbers that RSA users will never run out of them. The Prime Number Theorem states that the number of primes less than or equal to n is asymptotically $n/\log n$. This means that the number of prime numbers of length 512 bits or less is about 10^{150}, which is a number greater than the number of atoms in the known universe.

15. How do You Know if a Number is Prime?

It is generally recommended to use probabilistic primality testing, which is much quicker than actually proving that a number is prime. One can use a probabilistic test that determines whether a number is prime with arbitrarily small probability of error, say, less than 2^{-100}. For further discussion of some primality testing algorithms, see [BBC88]. For some empirical results on the reliability of simple primality tests, see [Riv91a]; one can perform very fast primality tests and be extremely confident in the results. A simple algorithm for choosing probable primes was analyzed by Brandt and Damgard [BD93b].

16. How is RSA used for Encryption in Practice?

RSA is combined with a secret-key cryptosystem, such as DES (see Question 64), to encrypt a message by means of an RSA digital envelope.

Suppose Alice wishes to send an encrypted message to Bob. She first encrypts the message with DES, using a randomly chosen DES key. Then she looks up Bob's public key and uses it to encrypt the DES key. The DES-encrypted message and the RSA-encrypted DES key together form the RSA digital envelope and are sent to Bob. Upon receiving the digital envelope, Bob decrypts the DES key with his private key, then uses the DES key to decrypt to message itself. This combines the high speed of DES with the key-management convenience of RSA.

17. How is RSA Used for Authentication in Practice? What are RSA Digital Signatures?

RSA is usually combined with a hash function (see Question 94) to sign a message.

Suppose Alice wishes to send a signed message to Bob. She applies a hash function to the message to create a message digest, which serves as a "digital fingerprint" of the message. She then encrypts the message digest with her RSA private key; this is the digital signature, which she sends to Bob along with the message itself. Bob, upon receiving the message and signature, decrypts the signature with Alice's public key to recover the message digest. He then hashes the message with the same hash function Alice used and compares the result to the message digest decrypted from the signature. If they are exactly equal, the signature has been successfully verified and he can be confident that the message did indeed come from Alice. If they are not equal, then the message either originated elsewhere or was altered after it was signed, and he rejects the message. With the method just described, anybody can read the message and verify the signature. This may not be applicable to situations where Alice wishes to retain the secrecy of the document. In this case she may wish to sign the document then encrypt it using Bob's public key. Bob will then need to decrypt using his private key and verify the signature on the recovered message using Alice's public key. A third party can also verify the signature at this stage.

In practice, the RSA public exponent is usually much smaller than the RSA private exponent; this means that the verification of a signature is faster than the signing. This is desirable because a message will be signed by an individual only once, but the signature may be verified many times.

It must be infeasible for anyone to either find a message that hashes to a given value or to find two messages that hash to the same value. If either were feasible, an intruder could attach a false message onto Alice's signature. Hash functions such as MD5 and SHA (see Question 99 and Question 100) have been designed specifically to have the property that finding a match is infeasible, and are therefore considered suitable for use in cryptography.

One or more certificates (see Question 123) may accompany a digital signature. A certificate is a signed document that binds the public key to the identity of a party. Its purpose is to prevent someone from impersonating someone else. If a certificate is present, the recipient (or a third party) can check that the public key belongs to a named party, assuming the certifier's public key is itself trusted.

18. What are the Alternatives to RSA?

Many other public-key cryptosystems have been proposed, as a look through the proceedings of the annual Crypto, Eurocrypt, and Asiacrypt conferences quickly reveals. Some of the public-key cryptosystems will be discussed in Question 24 through Question 44. More details can be found in [Sch95b].

A mathematical problem called the knapsack problem was the basis for several systems (see Question 32), but these have lost favor because several versions were broken. Another system, designed by ElGamal (see Question 29), is based on the discrete logarithm problem. The ElGamal system was, in part, the basis for several later signature methods, including one by Schnorr [Sch90], which in turn was the basis for DSS, the Digital Signature Standard (see Question 26).The ElGamal system has been used successfully in applications; it is slower for encryption and verification than RSA and its signatures are larger than RSA signatures.

In 1976, before RSA, Diffie and Hellman proposed a system for key exchange only (see Question 24); it permits secure exchange of keys in an otherwise conventional secret-key system. This system is in use today.

Cryptosystems based on mathematical operations on elliptic curves have also been proposed (see Question 31), as have cryptosystems based on discrete exponentiation in the finite field GF(2^n). The latter are very fast in hardware. There are also some probabilistic encryption methods (see Question 36), which have the attraction of being resistant to a guessed ciphertext attack (see Question 10), but with possible data expansion.

For digital signatures, Rabin (see Question 37) proposed a system which is provably equivalent to factoring; this is an advantage over RSA, where one may still have a lingering worry about an attack unrelated to factoring. Rabin’s method is susceptible to a chosen message attack, however, in which the attacker tricks the user into signing messages of a special form. Another signature scheme, by Fiat and Shamir [FS87], is based on interactive zero-knowledge protocols (see Question 107), but can be adapted for signatures. It is faster than RSA and is provably equivalent to factoring, but the signatures are much larger than RSA signatures. Other variations, however, lessen the necessary signature length; see [BDB92] for references. A system is “equivalent to factoring” if recovering the private key is provably as hard as factoring; forgery may be easier than factoring in some of the systems.

Advantages of RSA over other public-key cryptosystems include the fact that it can be used for both encryption and authentication, and that it has been around for many years and has successfully withstood much scrutiny. RSA has received far more attention, study, and actual use than any other public-key cryptosystem, and thus RSA has more empirical evidence of its security than more recent and less scrutinized systems. In fact, a large number of public-key cryptosystems which at first appeared secure were later broken; see [BO88] for some case histories.

19. Is RSA Currently in Use Today?

The use of RSA is practically ubiquitous today. It is currently used in a wide variety of products, platforms, and industries around the world. It is found in many commercial software products and is planned to be in many more. It is built into current operating systems by Microsoft, Apple, Sun, and Novell. In hardware, RSA can be found in secure telephones, on Ethernet network cards, and on smart cards. In addition, RSA is incorporated into all of the major protocols for secure Internet communications, including SSL, S-HTTP, SET, S/MIME, S/WAN and PCT. It is also used internally in many institutions, including branches of the U.S. government, major corporations, national laboratories, and universities.

As of the time of this publication, RSA technology is licensed by about 150 companies. The estimated installed base of RSA encryption engines is around 20 million, making it by far the most widely used public-key cryptosystem in the world. This figure is expected to grow rapidly as the use of the Internet and the World Wide Web grows at a blistering pace. For a partial list of software and hardware products implementing RSA, see *The 1995 Computer Security Products Buyer's Guide*. A more complete list of RSA licensees and OEM products can be found at `<http://www.rsa.com>`.

20. Is RSA an Official Standard Today?

RSA is part of many official standards worldwide. The ISO (International Standards Organization) 9796 standard lists RSA as a compatible cryptographic algorithm, as does the ITU-T X.509 security standard (see Question 165). RSA is part of the Society for Worldwide Interbank Financial Telecommunications (SWIFT) standard, the French financial industry's ETEBAC 5 standard, and the ANSI X9.31 draft standard for the U.S. banking industry (see Question 160). The Australian key management standard, AS2805.6.5.3, also specifies RSA.

RSA is found in Internet standards and proposed protocols including PEM (Privacy Enhanced Mail), S/MIME, PEM-MIME, S-HTTP and SSL (see Question 130 through Question 134) as well as the PKCS standard (see Question 166) for the software industry. The OSI Implementors' Workshop (OIW) has issued implementors' agreements referring to PKCS and PEM, each of which includes RSA.

A number of other standards are currently being developed and will be announced over the next couple of years; many are expected to include RSA as either an endorsed or a recommended system for privacy and/or authentication. A comprehensive survey of cryptography standards can be found in publications by Kaliski [Kal93b] and Ford [For94] .

21. Is RSA a De Facto Standard?

RSA is the most widely used public-key cryptosystem today and has often been called a de facto standard. Regardless of the official standards, the existence of a de facto standard is extremely important for the development of a digital economy. If one public-key system is used everywhere for authentication, then signed digital documents can be exchanged between users in different nations using different software on different platforms; this interoperability is necessary for a true digital economy to develop. Adoption of RSA has grown to the extent that standards are being written to accommodate RSA. When the U.S. financial industry was developing standards for digital signatures, it first developed ANSI X9.30 to support the federal requirement of using the Digital Signature Standard (see Question 26). They then modified X9.30 to X9.31 with the emphasis on RSA digital signatures to support the defacto standard of financial institutions.

The lack of secure authentication has been a major obstacle in achieving the promise that computers would replace paper; paper is still necessary almost everywhere for contracts, checks, official letters, legal documents, and identification. With this core of necessary paper transaction, it has not been feasible to evolve completely into a society based on electronic transactions. Digital signatures are the exact tool necessary to convert the most essential paper-based documents to digital electronic media. Digital signatures make it possible, for example, to have leases, wills, passports, college transcripts, checks, and voter registration forms that exist only in electronic form; any paper version would just be a “copy” of the electronic original. All of this is enabled by an accepted standard for digital signatures.

22. Is RSA Patented?

RSA is patented under U.S. Patent 4,405,829, issued September 29, 1983 and held by RSA Data Security, Inc. of Redwood City, California; the patent expires 17 years after issue, in 2000. RSA Data Security has a standard, royalty-based licensing policy which can be modified for special circumstances. The U.S. government can use RSA without a license because it was invented at MIT with partial government funding.

In the U.S., a license is needed to "make, use or sell" RSA. However, RSA Data Security usually allows free non-commercial use of RSA, with written permission, for academic or university research purposes. Furthermore, RSA Laboratories has made available (in the U.S. and Canada) at no charge a collection of cryptographic routines in source code, including the RSA algorithm; it can be used, improved and redistributed non-commercially (see Question 174).

23. Can RSA be Exported from the United States?

Export of RSA falls under the same U.S. laws as all other cryptographic products.

RSA used for authentication is more easily exported than when it is used for privacy. In the former case, export is allowed regardless of key (modulus) size, although the exporter must demonstrate that the product cannot be easily converted to use for encryption. In the case of RSA used for privacy (encryption), the U.S. government generally does not allow export if the key size exceeds 512 bits.

Export policy is currently a subject of debate, and the export status of RSA may well change in the next year or two. For example, a Commerce Jurisdiction (basically a general export license per Department of Commerce rather than Department of State approval) has been obtained by Cybercash for 768-bit RSA for financial transactions.

Regardless of U.S. export policy, RSA is available abroad in non-U.S. products.

Diffie-Hellman

24. What is Diffie-Hellman?

The Diffie-Hellman key agreement protocol (also called exponential key agreement) was developed by Diffie and Hellman [DH76] in 1976 and published in the ground-breaking paper "New Directions in Cryptography." The protocol allows two users to exchange a secret key over an insecure medium without any prior secrets.

The protocol has two system parameters p and g. They are both public and may be used by all the users in a system. Parameter p is a prime number and parameter g (usually called a generator) is an integer less than p, which is capable of generating every element from 1 to p-1 when multiplied by itself a certain number of times, modulo the prime p.

Suppose that Alice and Bob want to agree on a shared secret key using the Diffie-Hellman key agreement protocol. They proceed as follows: First, Alice generates a random private value a and Bob generates a random private value b. Then they derive their public values using parameters p and g and their private values. Alice's public value is g^a mod p and Bob's public value is g^b mod p. They then exchange their public values. Finally, Alice computes $k_{ab} = (g^b)^a$ mod p, and Bob computes $k_{ba} = (g^a)^b$ mod p. Since $k_{ab} = k_{ba.} = k$, Alice and Bob now have a shared secret key k.

The protocol depends on the discrete logarithm problem for its security. It assumes that it is computationally infeasible to calculate the shared secret key $k=g^{ab}$ mod p given the two public values g^a mod p and g^b mod p when the prime p is sufficiently large. Maurer [Mau94] has shown that breaking the Diffie-Hellman protocol was equivalent to computing discrete logarithms under certain assumptions.

The Diffie-Hellman key exchange is vulnerable to a *middleperson attack*. In this attack, an opponent, Carol, intercepts Alice's public value and sends her own public value to Bob. When Bob transmits his public value, Carol substitutes it with her own and sends it to Alice. Carol and Alice thus agree on one shared key and Carol and Bob agree on another shared key. After this exchange, Carol simply decrypts any messages sent out by Alice or Bob, and then reads and possibly modifies them before re-encrypting with the appropriate key and transmitting them to the correct party. This vulnerability is due to the fact that Diffie-Hellman key exchange does not authenticate the participants. Possible solutions include the use of digital signatures and other protocol variants (see Question 25).

25. What is Authenticated Diffie-Hellman Key Agreement?

The authenticated Diffie-Hellman key agreement protocol, or Station-to-Station (STS) protocol, was developed by Diffie, van Oorschot, and Wiener in 1992 [DVW92] to defeat the middleperson attack on the Diffie-Hellman key agreement protocol (see Question 24). The immunity is achieved by allowing the two parties to authenticate themselves to each other by the use of digital signatures (see Question 3) and public-key certificates (see Question 123).

Roughly speaking, the basic idea is as follows: Before the protocol, the two parties Alice and Bob each possess a public/private key pair and a certificate for the public key. During the protocol, Alice computes a signature on certain messages and sends Bob the public value g^a mod p together with her signature and her public-key certificate. Bob also proceeds in a similar way. Even though Carol is still able to intercept messages between Alice and Bob, she cannot forge signatures without Alice's private key and Bob's private key. Hence, the enhanced protocol defeats the middleperson attack.

DSA

26. What are DSA and DSS?

The Digital Signature Algorithm (DSA) was published by the National Institute of Standards and Technology (NIST) (see Question 146) in the Digital Signature Standard (DSS), which is a part of the U.S. government's Capstone project (see Question 150). DSS was selected by NIST, in cooperation with the NSA (see Question 148), to be the digital authentication standard of the U.S. government. The standard was issued on May 19, 1994.

DSA is based on the discrete logarithm problem (see Question 52) and derives from cryptosystems proposed by Schnorr [Sch90] and ElGamal (see Question 29). It is for authentication only. For a detailed description of DSA, see [NIS94b] or [NIS92].

In DSA, signature generation is faster than signature verification, whereas in RSA, signature verification is faster than signature generation (if the public and private exponents, respectively, are chosen for this property, which is the usual case). NIST claims that it is an advantage of DSA that signing is faster, but many people in cryptography think that it is better for verification to be the faster operation. Naccache *et al.* [NMR94] have developed some techniques to improve the efficiency of DSA, both for signing and verification.

DSA has been criticized by the computer industry since its announcement. Criticism has focused on a few main issues: it lacks key exchange capability; the underlying cryptosystem is too recent and has been subject to too little scrutiny for users to be confident of its strength; verification of signatures with DSA is too slow; the existence of a second authentication standard will cause hardship to computer hardware and software vendors, who have already standardized on RSA; and the process by which NIST chose DSA was too secretive and arbitrary, with too much influence wielded by NSA. Other criticisms were addressed by NIST by modifying the original proposal. A more detailed discussion of the various criticisms can be found in [NIS92], and a detailed response by NIST can be found in [SB93].

27. Is DSA Secure?

DSA is based on the difficulty of computing discrete logarithm (see Question 52) and is based on schemes proposed by Schnorr [Sch90] and ElGamal (see Question 29). The algorithm is generally considered secure when the key size is large enough. DSS was originally proposed by NIST with a fixed 512-bit key size. After much criticism that this is not secure enough especially for long-term security, NIST revised DSS to allow key sizes up to 1024 bits.

The particular form of the discrete logarithm problem used in DSA is to compute discrete logarithms in certain subgroups in the finite field GF(p) for some prime p. The problem was first proposed for cryptographic use in 1989 by Schnorr [Sch90]. Even though no attacks have been reported on this form of the discrete logarithm problem, further analysis is necessary to fully understand the difficulty of the problem.

Some researchers warned about the existence of “trapdoor” primes in DSA, which could enable a key to be easily broken. These trapdoor primes are relatively rare, however, and are easily avoided if proper key generation procedures are followed [SB93].

28. Is the Use of DSA Covered by Any Patents?

David Kravitz, former member of the NSA, holds a patent on DSA [Kra93]. Claus P. Schnorr has asserted that his patent [Sch91] covers certain implementations of DSA. Public Key Partners (PKP), Caro-Kann Corporation (a subsidiary of Cylink Corporation), and RSA Data Security have also asserted coverage of certain implementations of DSA by the Schnorr patent.

Other Public-Key Encryption and Signature Algorithms

29. What is the ElGamal Cryptosystem?

The ElGamal system [Elg85] is a public-key cryptosystem based on the discrete logarithm problem. It consists of both encryption and signature algorithms. The encryption algorithm is similar in nature to the Diffie-Hellman key agreement protocol (see Question 24).

The system parameters consist of a prime p and an integer g, whose powers modulo p generate a large number of elements, as in Diffie-Hellman. Alice has a private key a and a public key y, where $y = g^a \pmod{p}$. Suppose Bob wishes to send a message m to Alice. Bob first generates a random number k less than p. He then computes

$$y_1 = g^k \pmod{p} \qquad \text{and} \qquad y_2 = m \oplus y^k,$$

where ≈ denotes the bit-wise XOR. Bob sends (y_1, y_2) to Alice. Upon receiving the ciphertext, Alice computes

$$m = (y_1{}^a \, mod \, p) \oplus y_2 .$$

The ElGamal signature algorithm is similar to the encryption algorithm in that the public key and private key have the same form; however, encryption is not the same as signature verification, nor is decryption the same as signature creation as in RSA (see Question 8). DSA (see Question 26) is based in part on the ElGamal signature algorithm.

Analysis based on the best available algorithms for both factoring and discrete logarithms shows that RSA and ElGamal have similar security for equivalent key lengths. The main disadvantage of ElGamal is the need for randomness, and its slower speed (especially for signing). Another potential disadvantage of the ElGamal system is that message expansion by a factor of two takes place during encryption. However, such message expansion is negligible if the cryptosystem is used only for exchange of secret keys.

30. What are Elliptic Curves?

Elliptic curves are mathematical constructions from number theory and algebraic geometry, which in recent years have found numerous applications in cryptography.

An elliptic curve can be defined over any field (e.g., real, rational, complex). However, elliptic curves used in cryptography are mainly defined over finite fields. An elliptic curve consists of elements (x, y) satisfying the equation

$$y^2 = x^3 + ax + b$$

together with a single element denoted O called the "point at infinity," which can be visualized as the point at the top and bottom of every vertical line. Addition of two points on a elliptic curve is defined according to a set of simple rules (e.g., point p_1 plus point p_2 is equal to point $-p_3$ in Figure 1). The addition operation in an elliptic curve is the counterpart to modular multiplication in common public-key cryptosystems, and multiple addition is the counterpart to modular exponentiation. Elliptic curves are covered in more recent texts on cryptography, including a informative text by Koblitz [Kob94].

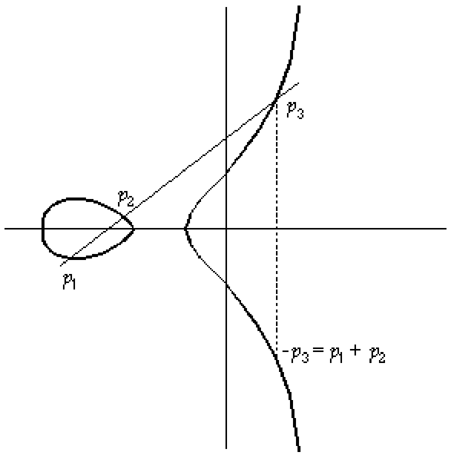

Figure 1. Elliptic Curve Addition.

31. What are Elliptic Curve Cryptosystems?

Elliptic curve cryptosystems [Mil86][Kob87] are analogs of public-key cryptosystems such as RSA (see Question 8) and ElGamal (see Question 29), in which modular multiplication is replaced by the elliptic curve addition operation.

The curves used in elliptic curve analogs of discrete logarithm cryptosystems are normally of the form

$$y^2 = x^3 + ax + b \pmod{p},$$

where p is prime. The problem tapped by the discrete logarithm analogs in elliptic curves is the elliptic curve logarithm problem, defined as follows: given a point G on an elliptic curve with order r (number of points on the curve) and another point Y on the curve, find a unique x ($0 \leq x \leq r - 1$) such that $Y = xG$, i.e., Y is the xth multiple of G.

Until recently, the best attacks on elliptic curve logarithm problems were the general methods applicable to any group. The methods have a running time of about a constant times the square root of r on average, which is much slower than specialized attacks on certain types of groups. The lack of specialized attacks means that shorter key sizes for elliptic cryptosystems give the same security as larger keys in cryptosystems that are based on discrete logarithm problem . However, for certain elliptic curves, Menezes, Okamoto, and Vanstone [MOV90] have been able to reduce the elliptic logarithm problem to a discrete logarithm problem. It is possible that algorithm development in this area will change the security of elliptic curve discrete logarithm cryptosystems to be equivalent to that of general discrete logarithm cryptosystems; this is an open research problem.

Elliptic curve analogs of RSA have been proposed, and they are based on the difficulty of factoring, just as RSA is. The elliptic curve analogs do not seem to offer any significant advantage over RSA, as the underlying problem is the same and the key sizes are similar for equivalent levels of security. Two of their purported advantages; resistance to “low-exponent” attacks, and to signature forgery against a chosen message attack; have recently been shown not to hold (see [KO95] and [Kal95]).

See [Men93] for more information on elliptic curve cryptosystems.

32. What are Knapsack Cryptosystems?

The Merkle-Hellman knapsack cryptosystem [MH78] is a public-key cryptosystem that was first published in 1978. It is commonly referred to as the *knapsack cryptosystem*. It is based on the subset sum problem in combinatorics. The problem involves selecting a number of objects with given weights from a large set such that the sum of the weights is equal to a pre-specified weight. This is considered to be a difficult problem to solve in general, but certain special cases of the problem are relatively easy to solve, which serve as the “trapdoor” of the system. The-single iteration knapsack cryptosystem introduced in 1978 was broken by Shamir [Sha84]. Merkle then published the multiple-iteration knapsack problem which was broken by Brickell [Bri85]. Merkle offered a $100 reward for anybody able to crack the single iteration knapsack and a $1000 reward for anybody able to crack the multiple iteration cipher from his own pocket. When they were cracked, he promptly paid up.

The Chor-Rivest knapsack cryptosystem was first published in 1984, followed by a revised version in 1988 [CR88]. It is the only knapsack-like cryptosystem that does not use modular multiplication. It was also the only knapsack-like cryptosystem that was secure for any extended period of time. Unfortunately, Schnorr and Hörner [SH95] developed an attack on the Chor-Rivest cryptosystem using improved lattice reduction which reduced to hours the amount of time needed to crack the cryptosystem for certain parameter values (though not for those recommended by Chor and Rivest). They also showed how the attack can be extended to attack Damgård’s knapsack hash function [Dam90].

33. What is LUC?

LUC is a public-key cryptosystem [SS95] developed by a group of researchers in Australia and New Zealand. The cipher implements the analogs of ElGamal (see Question 29), Diffie-Hellman (see Question 24), and RSA (see Question 8) over Lucas sequences. LUCELG is the Lucas sequence analog of ElGamal, while LUCDIF and LUCRSA are the Diffie-Hellman and RSA analogs. Lucas sequences used in the cryptosystem are the general second-order linear recurrence relation defined by

$$T_n = PT_{n-1} - QT_{n-2}$$

where P and Q are relatively prime integers. The encryption of the message is computed by iterating the recurrence, instead of by exponentiation as in RSA and Diffie-Hellman.

A recent paper by Bleichenbacher *et al.* [BBL95] shows that many of the supposed security advantages of LUC over cryptosystems based on modular exponentiation are either not present, or not as substantial as claimed.

34. What is the McEliece Cryptosystem?

The McEliece cryptosystem [Mce78] is a public-key encryption algorithm based on algebraic coding theory. The system uses a class of error-correcting codes, known as the Goppa codes, for which fast decoding algorithms exist. The basic idea is to construct a Goppa code as the private key and disguise it as a general linear code, which is the public key. The general linear code cannot be easily decoded unless the corresponding private matrix is known.

The McEliece cryptosystem has a number of drawbacks. These include large public-key size (in excess of one megabit), substantial expansion of data, and possibly a certain similarity to the knapsack cryptosystem. Korzhik and Turkin [KT91] reported a polynomial-time attack on the system, but it is not yet clear whether the attack is effective. Gabidulin, Paramonov, and Tretjakov [GPT91] proposed a modification of theMcEliece cryptosystem by replacing Goppa codes with a different algebraic code and claimed that the new version was more secure than the original system. However, Gibson [Gib93] later showed that there was not really any advantage to the new version.

35. What is Merkle's Tree Signature Scheme?

Merkle proposed a digital signature scheme that was based on both one-time signatures (see Question 43) and a hash function (see Question 94) and that provides an infinite tree of one-time signatures [Mer90b].

One-time signatures normally require the publishing of large amounts of data to authenticate many messages, since each signature can only be used once. Merkle's scheme solves the problem by implementing the signatures via a tree-like scheme. Each message to be signed corresponds a node in a tree, with each node consisting of the verification parameters that are used to sign a message and to authenticate the verification parameters of subsequent nodes. Although the number of messages that can be signed is limited by the size of the tree, the tree can be made arbitrarily large. Merkle's signature scheme is fairly efficient, since it requires only the application of hash functions.

36. What is Probabilistic Encryption?

Probabilistic encryption, discovered by Goldwasser and Micali [GM84], is a design approach for encryption where a message is encrypted into one of many possible ciphertexts (not just a single ciphertext as in deterministic encryption), in such a way that it is provably as hard to obtain partial information about the message from the ciphertext, as it is to solve some hard problem. In previous approaches to encryption, even though it was not always known whether one could obtain such partial information, neither was it proved that one could not do so.

A particular example of probabilistic encryption given by Goldwasser and Micali operates on "bits" rather than "blocks" and is based on the *quadratic residuosity problem*. The problem is to find whether an integer x is a square modulo a composite integer n. (This is easy if the factors of n are known, but presumably hard if they are not.) In their example, a "0" bit is encrypted as a random square, and a "1" bit as a non-square; thus it is as hard to decrypt as it is to solve the quadratic residuosity problem. The scheme has substantial message expansion due to the bit-by-bit encryption of the message. Blum and Goldwasser later proposed an efficient probabilistic encryption scheme with minimal message expansion [BG85].

37. What is the Rabin Signature Scheme?

The Rabin signature scheme [Rab79] is a variant of the RSA signature scheme (see Question 8). It has the advantage over RSA that finding the private key and forgery are both provably as hard as factoring. Verification is faster than signing, as with RSA signatures. In Rabin's scheme, the public key is an integer n where $n = pq$, and p and q are prime numbers which form the private key. The message to be signed must have a square root mod n; otherwise, it has to be modified slightly. Only about 1/4 of all possible messages have square roots mod n.

Signature:	$s = m^{1/2} \bmod n$	where s is the signature
Verification:	$m = s^2 \bmod n$	

The signature is easy to compute if the prime factors of n are known, but probably difficult otherwise – anyone who can forge the signature can also factor n. The provable security has the side-effect that the prime factors can be recovered under a chosen message attack. This attack can be countered by padding a given message with random bits or by modifying the message randomly, at the loss of provable security. (See [GMR86] for a discussion of a way to get around the paradox between provable security and resistance to chosen message attacks.)

Special Digital Signature Schemes

38. What are Special Signature Schemes?

Since the time Diffie and Hellman introduced the concept of digital signatures (see Question 3), many signature schemes have been proposed in cryptographic literature. These schemes can be categorized as either conventional digital signature schemes (e.g., RSA, DSA) or special signature schemes depending on their security features.

In a *conventional signature scheme* (the original model defined by Diffie and Hellman), we generally assume the following situation:

- The signer knows the contents of the message that he has signed.
- Anyone who knows the public key of the signer can verify the correctness of the signature at any time without any consent or input from the signer. (Digital signature schemes with this property are called *self-authenticating signature schemes*.)
- The security of the signature schemes (i.e., hard to forge, non-repudiation, see Question 3) is based on certain complexity-theoretic assumptions.

In some situations, it may be better to relax some of these assumptions, and/or add certain special security features. For example, when Alice asks Bob to sign a certain message, she may not want him to know the contents of the message. In the past decade, a variety of special signature schemes have been developed to fit security needs in different applications. More examples of such special schemes are given in Question 39 through Question 44 in alphabetic order.

39. What is a Blind Signature Scheme?

Blind signature schemes, first introduced by Chaum [Cha83][Cha85], allow a person to get a message signed by another party without revealing any information about the message to the other party.

Chaum demonstrated the implementation of this concept using RSA signatures (seeQuestion 8) as follows: Suppose Alice has a message m that she wishes to have signed by Bob, and she does not want Bob to learn anything about m. Let (n,e) be Bob's public key and (n,d) be his private key. Alice generates a random value r such that $gcd(r, n) = 1$ and sends

$$m' = r^e m \bmod n$$

to Bob. The value m' is "blinded" by the random value r, and hence Bob can derive no useful information from it. Bob returns the signed value,

$$s' = (m')^d = (r^e m)^d \bmod n$$

to Alice. Since $s' = rm^d \bmod n$, Alice can obtain the true signature s of m by computing

$$s = s' r^{-1} \bmod n.$$

Now Alice's message has a signature she could not have obtained on her own. This signature scheme is secure provided that factoring and root extraction remain difficult. However, regardless of the status of these problems the signature scheme is unconditionally "blind" since r is random. The random r does not allow the signer to learn about the message even if the signer can solve the underlying hard problems.

There are potential problems if Alice can give an arbitrary message to be signed, since this effectively enables her to mount a chosen message attack. One way of thwarting this kind of attack is described in [CFN88].

Blind signatures have numerous uses including timestamping (see Question 108), anonymous access control, and digital cash (see Question 138). Thus it is not surprising there are now numerous variations on the blind signature theme. Further work on blind signatures has been carried out in recent years [FY94][SPC95].

40. What is a Designated Confirmer Signature?

A designated confirmer signature [Cha94] strikes a balance between self-authenticating digital signatures (see Question 38) and zero-knowledge proofs (see Question 107). While the former allows anybody to verify a signature, the latter can only convince one recipient at a time of the authenticity of a given document, and only through interaction with the signer. A *designated confirmer signature* allows certain designated parties to confirm the authenticity of a document without the need for the signer's input. At the same time, without the aid of either the signer or the designated parties, it is not possible to verify the authenticity of a given document. Chaum developed implementations of designated confirmer signatures with one or more confirmers using RSA digital signatures (see Question 8).

41. What is a Fail-stop Signature Scheme?

A *fail-stop signature scheme* is a type of signature devised by van Heyst and Pederson [VP92] to protect against the possibility that an enemy may be able to forge a person's signature. It is a variation of the one-time signature scheme (see Question 43), in which only a single message can be signed and protected by a given key at a time. The scheme is based on the discrete logarithm problem. In particular, if an enemy can forge a signature, then the actual signer can prove that forgery has taken place by demonstrating the solution of a supposedly hard problem. Thus the forger's ability to solve that problem is transferred to the actual signer. (The term "fail-stop" refers to the fact that a signer can detect and stop failures, i.e., forgeries. Note that if the enemy obtains an actual copy of the signer's private key, forgery cannot be detected. What the scheme detects are forgeries based on cryptanalysis.)

42. What is a Group Signature?

A *group signature*, introduced by Chaum and van Heijst [CV91], allows any member of a group to digitally sign a document in a manner such that a verifier can confirm that it came from the group, but does not know which individual in the group signed the document. The protocol allows for the identity of the signer to be discovered, in case of disputes, by a designated group authority who has some auxiliary information. Unfortunately, each time a member of the group signs a document, a new key pair has to be generated for the signer. The generation of new key pairs causes the length of both the group members' secret keys and the designated authority's auxiliary information to grow. This tends to cause the scheme to become unwieldy when used by a group to sign numerous messages or when used for an extended period of time. Some improvements [CP94][CP95] have been made in the efficiency of this scheme.

43. What is a One-time Signature Scheme?

A *one-time signature scheme* allows the signature of only a single message using a given piece of private (and public) information. One advantage of such a scheme is that it is generally quite fast. However, the scheme tends to be unwieldy when used to authenticate multiple messages because additional data needs to be generated to both sign and verify each new message. By contrast, with conventional signature schemes like RSA (see Question 8), the same key pair can be used to authenticate multiple documents. There is a relatively efficient implementation of one-time-like signatures by Merkle called the Merkle Tree Signature Scheme (see Question 35), which does not require new key pairs for each message.

44. What is an Undeniable Signature Scheme?

Undeniable signature scheme, devised by Chaum and van Antwerpen [CV90][CV92], are non-self-authenticating signature schemes (see Question 38), where signatures can only be verified with the signer's consent. However, if a signature is only verifiable with the aid of a signer, a dishonest signer may refuse to authenticate a genuine document. Undeniable signatures solve this problem by adding a new component called the *disavowal protocol* in addition to the normal components of signature and verification.

The scheme is implemented using public-key cryptography based on the discrete logarithm problem (see Question 52). The signature part of the scheme is similar to other discrete logarithm signature schemes. Verification is carried out by a challenge-response protocol where the verifier, Alice, sends a challenge to the signer, Bob, and views the answer to verify the signature. The disavowal process is similar: Alice sends a challenge and Bob's response shows that a signature is not his. If Bob does not take part, it may be assumed that the document is authentic. The probability that a dishonest signer is able to successfully mislead the verifier in either verification or disavowal is $1/p$ where p is the prime number in the signer's private key. If we consider the average 768-bit private key, there is only a minuscule probability that the signer will be able to repudiate a document he has signed.

Factoring and Discrete Logarithms

45. What is the Factoring Problem?

Factoring is the act of splitting an integer into a set of smaller integers (factors) which, when multiplied together, form the original integer. For example, the factors of 15 are 3 and 5; the factoring problem is to find 3 and 5 when given 15. Prime factorization requires splitting an integer into factors that are prime numbers; every integer has a unique prime factorization. Multiplying two prime integers together is easy, but as far as we know, factoring the product is much more difficult.

46. What is the Significance of Factoring in Cryptography?

Factoring is the underlying, presumably hard problem upon which several public-key cryptosystems are based, including RSA. Factoring an RSA modulus (see Question 8) would allow an attacker to figure out the private key; thus, anyone who can factor the modulus can decrypt messages and forge signatures. The security of RSA depends on the factoring problem being difficult and the presence of no other types of attack. Unfortunately, it has not been proven that factoring must be difficult, and there remains a possibility that a quick and easy factoring method might be discovered (see Question 49), although factoring researchers consider this possibility remote.

Factoring large numbers takes more time than factoring smaller numbers. This is why the size of the modulus in RSA determines how secure an actual use of RSA is; the larger the modulus, the longer it would take an attacker to factor, and thus the more resistant to attack the RSA modulus is.

47. Has Factoring Been Getting Easier?

Factoring has become easier over the last 15 years for two reasons: computer hardware has become more powerful, and better factoring algorithms have been developed.

Hardware improvement will continue inexorably, but it is important to realize that *hardware improvements make RSA more secure, not less.* This is because a hardware improvement that allows an attacker to factor a number two digits longer than before will at the same time allow a legitimate RSA user to use a key dozens of digits longer than before; a user can choose a new key a dozen digits longer than the old one without any performance slowdown, yet a factoring attack will become much more difficult. Therefore, although the hardware improvement does help the attacker, it helps the legitimate user much more. This general rule may fail in the sense that factoring may take place using fast machines of the future, attacking RSA keys of the past; in this scenario, only the attacker gets the advantage of the hardware improvement. This consideration argues for using a larger key size today than one might otherwise consider warranted. It also argues for replacing one's RSA key with a longer key every few years, in order to take advantage of the extra security offered by hardware improvements. This point holds for other public-key systems as well.

Better factoring algorithms have been more help to the RSA attacker than have hardware improvements. As the RSA system and cryptography in general have attracted much attention, so has the factoring problem, and many researchers have found new factoring methods or improved upon others. This has made factoring easier for numbers of any size, irrespective of the speed of the hardware. However, factoring is still a very difficult problem.

Overall, any recent decrease in security due to algorithm improvement can be offset by increasing the key size. In fact, between general computer hardware improvements and special-purpose RSA hardware improvements, increases in key size (maintaining a constant speed of RSA operations) have kept pace or exceeded increases in algorithm efficiency, resulting in no net loss of security. As long as hardware continues to improve at a faster rate than the rate at which the complexity of factoring algorithms decreases, the security of RSA will increase, assuming RSA users regularly increase their key sizes by appropriate amounts. The open question is how much faster factoring algorithms can get; there could be some intrinsic limit to factoring speed, but this limit remains unknown.

48. What are the Best Factoring Methods in Use Today?

Factoring is a very active field of research among mathematicians and computer scientists. The best factoring algorithms are mentioned below with some references and their big-O asymptotic efficiency. O notation measures how fast an algorithm is; it gives an upper bound on the number of operations (to order of magnitude) in terms of n, the number to be factored, and p, a prime factor of n. For textbook treatment of factoring algorithms, see [Knu81][Kob94][LL90][Bre89], for a detailed explanation of big-O notation, see [CLR90].

Factoring algorithms come in two forms, special purpose and general purpose. The efficiency of the former depends on the unknown factors, whereas the efficiency of the latter depends on the number to be factored. Special-purpose algorithms are best for factoring numbers with small factors, but the numbers used for the modulus in the RSA system do not have any small factors. Therefore, general-purpose factoring algorithms are the more important ones in the context of cryptographic systems and their security.

Special-purpose factoring algorithms include the *Pollard rho method* [Pol75], with expected running time $O(\sqrt{p})$ and the Pollard p - 1 method [Pol74], with running time $O(p')$, where p' is the largest prime factor of $p - 1$. Both of these take an amount of time that is exponential in the size of p, the prime factor that they find; thus these algorithms are too slow for most factoring jobs. The *elliptic curve method* (ECM) [Len87] is superior to these; its asymptotic running time is $O(\exp(\sqrt{2 \ln p \ln \ln p}))$. The ECM is often used in practice to find factors of randomly generated numbers; it is not strong enough to factor a large RSA modulus.

The best general-purpose factoring algorithm today is the *number field sieve* [BLP94] [BLZ94], which runs in time approximately $O(\exp(1.9223(\ln n)^{1/3} (\ln \ln n)^{2/3}))$. Previously, the most widely used general-purpose algorithm was the *multiple polynomial quadratic sieve* (mpqs) [Sil87], which has running time $O(\exp(\sqrt{\ln n \ln \ln n}))$. This was due to initial inefficiencies that made the number field sieve less efficient than the quadratic sieve. Recent improvements to the number field sieve make the number field sieve more efficient in factoring numbers larger than about 115 digits [DL95]. RSA-129 (see Question 51) was factored using a variation of the quadratic sieve method. It is now estimated that if the number field sieve had been used, it would have taken one quarter of the time.

Clearly, the number field sieve will overtake the mpqs as the most widely used factoring algorithm, as the size of the numbers being factored increases from about

130 digits, which is the current threshold of general numbers which can be factored, to 140 or 150 digits. A “general number” is one with no special form that might make it easier to factor; RSA moduli are created to be general numbers. Note that a 512-bit number has about 155 digits.

Numbers with up to 155 digits or more that have a special form can already be factored [LLM93]. The Cunningham Project [BLS88] keeps track of the factorizations of numbers with these special forms and maintains a “10 Most Wanted” list of desired factorizations. Also, a good way to survey current factoring capability is to look at recent results of the RSA Factoring Challenge (see Question 50).

49. What are the Prospects for a Theoretical Factoring Breakthrough?

Factoring is widely believed to be a difficult mathematical problem, but it has not been proved so. Therefore, there remains a possibility that an easy factoring algorithm will be discovered. This development, which could seriously weaken RSA, would be highly surprising and the possibility is considered remote by the researchers most actively engaged in factoring research.

Another possibility is that someone will prove that factoring is difficult. Such a development, while unexpected at the current state of theoretical factoring research, would guarantee the security of RSA beyond a certain key size.

Even if no breakthroughs are discovered in factoring algorithms, both factoring and discrete logarithm problems can be solved efficiently on a quantum computer (see Question 109) if one is ever developed.

50. What is the RSA Factoring Challenge?

The RSA Factoring Challenge was started in March 1991 by RSA Data Security, Inc. to keep abreast of the state of the art in factoring. Since its inception well over a thousand numbers have been factored, with the factorers returning valuable information on the methods they used to complete the factorizations. The Factoring Challenge provides one of the largest test-beds for factoring implementations and provides one of the largest collections of factoring results from many different experts worldwide. In short, this vast pool of information gives us an excellent opportunity to compare the effectiveness of different factoring techniques as they are implemented and used in practice. Since the Security of the RSA public-key cryptosystem relies on the inability to factor large numbers of a special type, the cryptographic significance of these results is self-evident.

The contest is administered by RSA Data Security with quarterly cash prices. Send e-mail to `<challenge-info@rsa.com>` for more information. For an analysis of results from the factoring challenge, see [FR95].

51. What is RSA-129?

RSA-129 is a 129-digit (426-bit) integer published in Martin Gardner's column in *Scientific American* in 1977. A prize of $100 was offered to anybody able to factor the number. The number was factored in March 1994 by Atkins *et al*. [AGL95] using the resources of 1600 computers (which included two fax machines) from the Internet. The factoring took about 4000 to 6000 MIPS years of computation over an eight-month period. It was factored using the quadratic sieve factoring method and, according to Lenstra, will perhaps be the last large number to be factored using the quadratic sieve since the general number field sieve is now more efficient for numbers of this size and larger (see Question 48).

52. What is the Discrete Logarithm Problem?

The *discrete logarithm problem* applies to groups. Given an element g in a group G of order t, and another element y of G, the problem is find x, where $0 \leq x \leq t - 1$, such that y is the result of composing g with itself x times. The element g can typically generate all the elements of G or at least a large subset by exponentiating (i.e., applying the group operation repeatedly) with all the integers from 0 to t - 1. The element g is called a generator if it can generate all the elements in the group.

Like the factoring problem, the discrete logarithm problem is believed to be difficult and also to be the hard direction of a one-way function. For this reason, it has been the basis of several public-key cryptosystems, including the ElGamal system and DSS (see Question 29 and Question 26). The discrete logarithm problem bears the same relation to these systems as factoring does to RSA. The security of these systems rests on the assumption that discrete logarithms are difficult to compute.

The discrete logarithm problem has received much attention in recent years; descriptions of some of the most efficient algorithms for discrete logarithms over finite fields can be found in [Odl84][LL90][COS86][Gor93][GM93]. The best discrete logarithm problems have expected running times similar to those of the best factoring algorithms. Rivest [Riv92a] has analyzed the expected time to solve the discrete logarithm problem both in terms of computing power and cost. The discrete logarithm problem appears to be much harder over arbitrary groups than over finite fields; this is the motivation for cryptosystems based on elliptic curves (see Question 31). In general, the running time for computing discrete logarithms in arbitrary groups is $O(\sqrt{p})$, where p is the order of the group [Pol74].

53. Which is Easier: Factoring or Discrete Logarithm?

The asymptotic running time of the best discrete logarithm algorithm is approximately the same as that of the best general-purpose factoring algorithm. Therefore, it requires about as much effort to solve the discrete logarithm problem modulo a 512-bit prime as it does to factor a 512-bit RSA modulus. One paper [LO91] cites experimental evidence that the discrete logarithm problem is slightly harder than factoring; the authors suggest that the effort necessary to factor a 110-digit integer is the same as the effort to solve discrete logarithms modulo a 100-digit prime. This difference is so slight that it should not be a significant consideration when choosing a cryptosystem.

Historically, it has been the case that an algorithmic advance in either problem, factoring or discrete logs, was then applied to the other. This suggests that the degrees of difficulty of both problems are closely linked, and that any breakthrough, either positive or negative, will affect both problems equally.

BLOCK CIPHERS

General

54. What is a Block Cipher?

A *block cipher* transforms a fixed-length block of plaintext data into a block of ciphertext data of the same length. This transformation takes place under the action of a user-provided secret key. Decryption is performed by applying the reverse transformation to the ciphertext block using the same secret key. The fixed length is called the *block size*, and for many block ciphers, the block size is 64 bits.

Since different plaintext blocks are mapped to different ciphertext blocks (to allow *unique* decryption), a block cipher effectively provides a permutation of the set of all possible messages. The permutation effected during any particular encryption is of course secret, since it is a function of the secret key.

More information about block ciphers and the various available algorithms can be found in almost any book on contemporary cryptography and in RSA Laboratories Technical Report [Rob95a].

55. What is an Iterated Block Cipher?

An *iterated block cipher* is one that encrypts a plaintext block by a process that has several rounds. In each round, the same transformation or *round function* is applied to the data using a *subkey*. The set of subkeys are usually derived from the user-provided secret key by a *key schedule.*

The number of rounds in an iterated cipher depends on the desired security level and the consequent trade-off with performance. In most cases, an increased number of rounds will improve the security offered by a block cipher, but for some ciphers the number of rounds required to achieve adequate security will be too large for the cipher to be practical or desirable.

56. What is a Feistel Cipher?

Feistel ciphers [Fei73] are a special class of iterated block ciphers (see Question 55) where the ciphertext is calculated from the plaintext by repeated application of the same transformation or round function. Feistel ciphers are also sometimes called DES-like ciphers (see Question 64).

In a Feistel cipher (see Figure 2), the text being encrypted is split into two halves. The round function *f* is applied to one half using a subkey and the output of *f* is XORed with the other half. The two halves are then swapped. Each round follows the same pattern except for the last round where there is no swap.

A nice feature of a Feistel cipher is that encryption and decryption are structurally identical, though the subkeys used during encryption at each round are taken in reverse order during decryption.

It is possible to design iterative ciphers that are not Feistel ciphers, yet whose encryption and decryption (after a certain re-ordering or re-calculation of variables) are structurally the same. One such example is IDEA (see Question 77).

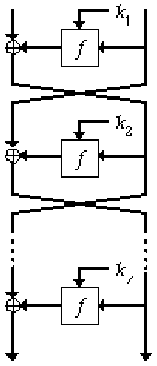

Figure 2. Feistel Cipher

57. What is Exhaustive Key Search?

Exhaustive key search, or *brute-force search*, is the basic technique of trying every possible key in turn until the correct key is identified. To identify the correct key it may be necessary to possess a plaintext and its corresponding ciphertext, or if the plaintext has some recognizable characteristic, ciphertext alone might suffice. Exhaustive key search can be mounted on any cipher and sometimes a weakness in the key schedule (see Question 55) of the cipher can help improve the efficiency of an exhaustive key search attack.

Advances in technology and computing performance will always make exhaustive key search an increasingly practical attack against keys of a fixed length. When DES (see Question 64) was designed, it was generally considered secure against exhaustive key search without a vast financial investment in hardware [DH77]. Over the years, this line of attack will become increasingly attractive to a potential adversary [Wie94].

While the 56-bit key in DES now only offers a few hours of protection against exhaustive search by a modern dedicated machine [Wie94], the current rate of increase in computing power is such that Skipjack's 80-bit key (see Question 80) can be expected to offer the same level of protection against exhaustive key search in 18 years time as DES does today [BDK93].

58. What is Differential Cryptanalysis?

Differential cryptanalysis is a type of attack that can be mounted on iterative block ciphers. These techniques were first introduced by Murphy [Mur90] in an attack on FEAL-4 (see Question 79), but they were later improved and perfected by Biham and Shamir [BS91a][BS93b] who used them to attack DES (see Question 64). Differential cryptanalysis is basically a chosen plaintext attack (see Question 63) and relies on an analysis of the evolution of the differences between two related plaintexts as they are encrypted under the same key. By careful analysis of the available data, probabilities can be assigned to each of the possible keys and eventually the most probable key is identified as the correct one.

Differential cryptanalysis has been used against a great many ciphers with varying degrees of success. In attacks against DES, its effectiveness is limited by what was very careful design of the S-boxes during the design of DES in the mid-1970s [Cop92]. Studies on protecting ciphers against differential cryptanalysis have been conducted by Nyberg and Knudsen [NK95] as well as Lai, Massey and Murphy [LMM92]. Differential cryptanalysis has also been useful in attacking other cryptographic algorithms such as hash functions.

59. What is Linear Cryptanalysis?

Linear cryptanalysis was first devised by Matsui and Yamagishi [MY92] in an attack on FEAL (see Question 79). It was extended by Matsui [Mat93] to attack DES (see Question 64). Linear cryptanalysis is a known plaintext attack (see Question 63) and uses a linear approximation to describe the behavior of the block cipher. Given sufficient pairs of plaintext and corresponding ciphertext, bits of information about the key can be obtained and increased amounts of data will usually give a higher probability of success.

There have been a variety of enhancements and improvements to the basic attack. Langford and Hellman [LH94] introduced an attack called *differential-linear cryptanalysis* which combines elements of differential cryptanalysis (see Question 58) with those of linear cryptanalysis. Also, Kaliski and Robshaw [KR94] showed that a linear cryptanalytic attack using multiple approximations might allow for a reduction in the amount of data required for a successful attack. Other issues such as protecting ciphers against linear cryptanalysis have been considered by Nyberg [Nyb95], Knudsen [Knu93], and O'Conner [Oco95].

60. What is a Weak Key for a Block Cipher?

Weak keys are secret keys with a certain value for which the block cipher in question will exhibit certain regularities in encryption or, in other cases, a poor level of encryption. For instance, with DES (see Question 64) there are four keys for which encryption is exactly the same as decryption. This means that if one were to encrypt twice with one of these weak keys, then the original plaintext would be recovered. For IDEA (see Question 77) there is a class of keys for which cryptanalysis is greatly facilitated and the key can be recovered. However, in both these cases, the number of weak keys is such a small fraction of all possible keys that the chance of picking one at random is exceptionally slight. In such cases, they pose no significant threat to the security of the block cipher when used for encryption.

Of course for other block ciphers, there might well be a large set of weak keys (perhaps even with the weakness exhibiting itself in a different way) for which the chance of picking a weak key is too large for comfort. In such a case, the presence of weak keys would have an obvious impact on the security of the block cipher.

61. What are Algebraic Attacks?

Algebraic attacks are a class of techniques which rely for their success on some block cipher exhibiting a high degree of mathematical structure.

For instance, it is conceivable that a block cipher might exhibit what is termed a *group structure*. If this were the case, then encrypting a plaintext under one key and then encrypting the result under another key would always be equivalent to single encryption under some other single key. If so, then the block cipher would be considerably weaker, and the use of multiple encryption (see Question 85) would offer no additional security over single encryption; see [KRS88] for a more complete discussion. For most block ciphers, the question of whether they form a group is still open. For DES (see Question 64), however, it is known that the cipher is not a group.

There are a variety of other concerns with regards to algebraic attacks. See [Rob95a] for more details.

62. How Can Data Compression be Used With Encryption?

Data compression removes redundant character strings in a file. This means that the compressed file has a more uniform distribution of characters. In addition to providing shorter plaintext and ciphertext, which reduces the amount of time needed to encrypt, decrypt and transmit a file, the reduced redundancy in the plaintext can potentially hinder certain cryptanalytic attacks.

By contrast, compressing a file after encryption is inefficient. The ciphertext produced by a good encryption algorithm should have an almost statistically uniform distribution of characters. As a consequence, a compression algorithm should be unable to find redundant patterns in such text and there will be little, if any, data compression. In fact, if a data compression algorithm is able to significantly compress encrypted text, then this indicates a high level of redundancy in the ciphertext which, in turn, is evidence of poor encryption.

63. At What Point Does an Attack Become Practical?

There is no easy answer to this question since it depends on many distinct factors. Not only must the work and computational resources required by the cryptanalyst be reasonable, but the amount and type of data required for the attack to be successful must also be taken into account.

One classification distinguishes among cryptanalytic attacks according to the data they require in the following way: *chosen plaintext* or *chosen ciphertext*, *known plaintext,* and *ciphertext-only*. This classification is not particular to secret-key ciphers and can be applied to cryptanalytic attacks on any cryptographic function.

A *chosen plaintext* or *chosen ciphertext* attack gives the cryptanalyst the greatest freedom in analyzing a cipher. The cryptanalyst chooses the plaintext to be encrypted and analyzes the plaintext together with the resultant ciphertext to derive the secret key. Such attacks will, in many circumstances, be difficult to mount but they should not be discounted. As Merkle and Hellman have remarked [MH81], a chosen text attack can in some ways be viewed as a "certificational weakness" in a cryptosystem.

A *known plaintext* attack is more useful to the cryptanalyst than a chosen plaintext attack (with the same amount of data) since the cryptanalyst now requires a certain numbers of plaintexts and their corresponding ciphertexts without specifying the values of the plaintexts. This type of information is presumably easier to collect.

The most practical attack, but perhaps the most difficult to actually discover, is a *ciphertext-only* attack. In such an attack, the cryptanalyst merely intercepts a number of encrypted messages and subsequent analysis somehow reveals the key used for encryption. Note that some knowledge of the statistical distribution of the plaintext is required for a ciphertext-only attack to succeed.

An added level of sophistication to the chosen text attacks is to make them *adaptive*. By this we mean that the cryptanalyst has the additional power to choose the text that is to be encrypted or decrypted after seeing the results of previous requests.

The computational effort and resources together with the amount and type of data required are all important features in assessing the practicality of some attack.

DES

64. What is DES?

DES is the Data Encryption Standard, an encryption block cipher defined and endorsed by the U.S. government in 1977 as an official standard; the details can be found in the latest official FIPS (Federal Information Processing Standards) publication concerning DES [NIS93b]. It was originally developed at IBM. DES has been extensively studied since its publication and is the most well-known and widely used cryptosystem in the world.

DES is a symmetric cryptosystem. When used for communication, both sender and receiver must know the same secret key, which is used both to encrypt and decrypt the message. DES can also be used for single-user encryption, such as to store files on a hard disk in encrypted form. In a multi-user environment, secure key distribution may be difficult; public-key cryptography provides an ideal solution to this problem (see Question 4).

DES has a 64-bit block size (see Question 54) and uses a 56-bit key during encryption. It is a 16-round Feistel cipher (see Question 56) and was originally designed for implementation in hardware.

NIST (see Question 146) has recertified DES as an official U.S. government encryption standard every five years; DES was last recertified in 1993, by default. NIST has indicated, however, that it may not recertify DES again.

65. Has DES been Broken?

No easy attack on DES has been discovered, despite the efforts of many researchers over many years. The obvious method of attack is brute-force exhaustive search of the key space; this takes 2^{55} steps on average. Early on it was suggested [DH77] that a rich and powerful enemy could build a special-purpose computer capable of breaking DES by exhaustive search in a reasonable amount of time. Later, Hellman [Hel80] showed a time-memory trade-off that allows improvement over exhaustive search if memory space is plentiful, after an exhaustive precomputation. These ideas fostered doubts about the security of DES. There were also accusations that the NSA (see Question 148) had intentionally weakened DES. Despite these suspicions, no feasible way to break DES faster than exhaustive search (see Question 57) was discovered. The cost of a specialized computer to perform exhaustive search (requiring 3.5 hours on average) has been estimated by Wiener at one million dollars [Wie94].

The first attack on DES that is better than exhaustive search in terms of computational requirements was announced by Biham and Shamir [BS93a] using a new technique known as differential cryptanalysis (see Question 58). This attack requires the encryption of 2^{47} chosen plaintexts (see Question 63), that is, the plaintexts are chosen by the attacker. Although it is a theoretical breakthrough, this attack is not practical because of both the large data requirements and the difficulty of mounting a chosen plaintext attack. Biham and Shamir have stated that they consider DES secure.

More recently Matsui [Mat94] has developed another attack, known as linear cryptanalysis (see Question 59). A DES key can be recovered by the analysis of 2^{43} known plaintexts and the first experimental cryptanalysis of DES was successfully achieved in an attack requiring 50 days on 12 HP 9735 workstations. Clearly, however, this attack is still impractical.

The consensus is that DES, when used properly, is still secure and that triple encryption DES (see Question 72 and Question 85) is far more secure than DES. Both single and triple encryption DES are used extensively in a wide variety of cryptographic systems.

66. How Does One Use DES Securely?

When using DES, there are several practical considerations that can affect the security of the encrypted data. One should change DES keys frequently, in order to prevent attacks that require sustained data analysis. In a communications context, one must also find a secure way of communicating the DES key to both sender and receiver. Use of RSA (see Question 8) or some other public-key technique for key management solves both these issues; a different DES key is generated for each session, and secure key management is provided by encrypting the DES key with the receiver's RSA public key. RSA, in this circumstance, can be regarded as a tool for improving the security of DES (or any other secret-key cipher).

If one wishes to use DES to encrypt files stored on a hard disk, it is not feasible to frequently change the DES keys, as this would entail decrypting and then re-encrypting all files upon each key change. Instead, one should have a master DES key which encrypts the list of DES keys used to encrypt the files; one can then change the master key frequently without much effort. Alternatively the user may wish to use a dedicated file/disk encryption program like RSA Secure (see Question 175).

Aside from the issues mentioned above, DES can be used for encryption in several officially-defined modes (see Question 82 and Question 83), and these modes have a variety of properties. ECB (electronic codebook) mode simply encrypts each 64-bit block of plaintext one after another under the same 56-bit DES key. In CBC (cipher block chaining) mode, each 64-bit plaintext block is XORed with the previous ciphertext block before being encrypted with the DES key. Thus the encryption of eachblock depends on previous blocks and the same 64-bit plaintext block can encrypt to different ciphertext depending on its context in the overall message. CBC mode helps protect against certain attacks, but not against exhaustive search or differential cryptanalysis. CFB (cipher feedback) mode allows one to use DES with block lengths less than 64 bits. Detailed descriptions of the various DES modes can be found in [NIS80].

In practice, CBC is the most widely used mode of DES, and it is specified in several standards. For additional security, one could use triple encryption with CBC (see Question 73).

67. Should One Test for Weak Keys in DES?

DES has four weak keys *k* for which $E_k(E_k(m)) = m$ (see Question 60). There are also twelve semi-weak keys which come in pairs *k1* and *k2* and are such that $E_{k1}(E_{k2}(m)) = m$.

Since there are 2^{56} possible DES keys the chance of picking a weak or semi-weak key at random is 2^{-52}. As long as the user-provided key is chosen entirely at random, they can be safely ignored when DES is used for encryption. Despite this, some users prefer to test whether a key to be used for DES encryption is in fact a weak key. Such a test will have no significant impact on the time required for encryption.

68. Can DES be Exported from the United States?

Export of DES, either in hardware or software, is strictly regulated by the U.S. State Department and the NSA (see Question 148). The government rarely approves export of DES, despite the fact that DES is widely available overseas; financial institutions and foreign subsidiaries of U.S. companies are exceptions.

69. What are the Alternatives to DES?

Over the years, various new block cipher algorithms have been designed as alternatives to DES. One is FEAL (see Question 79), a cipher for which numerous attacks have been discovered. IDEA is a cipher designed by Lai and Massey that seems much more promising (see Question 77) and two more recent designs are RC5 (see Question 76) and SAFER (see Question 78). In addition, the U.S. government announced in 1993 an algorithm called Skipjack (see Question 80) as part of its Capstone project (see Question 150). Skipjack operates on 64-bit blocks of data, as does DES, but uses 80-bit keys, as opposed to the 56-bit keys in DES. However, the details of Skipjack are classified, so Skipjack is only available in hardware from government-authorized manufacturers.

In addition to RC5, Rivest perviously developed the block cipher RC2 (see Question 75) and the stream cipher RC4 (see Question 87). All three algorithms have a variable key size (so that the security they offer can be adjusted) and all three algorithms are faster than DES, at least in software. RC2 and RC4 have the further advantage of special U.S. government status whereby the export approval is simplified and expedited if the key size is limited to 40 bits.

70. Is DES a Group?

No, DES is not a group (see Question 61). This issue was settled only after many years of speculation and circumstantial evidence [CW93] and this result seems to imply that techniques such as triple encryption do in fact increase the security of DES.

71. What is DES with Independent Subkeys?

The DES algorithm derives sixteen 48-bit subkeys, for use in each of the 16 rounds, from the 56-bit secret key supplied by the user. It is interesting to consider the effect of using a 768-bit key (divided into 16 48-bit subkeys) in place of the 16 related 48-bit keys that are generated by the key schedule in the DES algorithm.

While the use of independent subkeys would obviously vastly increase the effort required for exhaustive key search, such a change to the cipher would make it only moderately more secure against differential and linear cryptanalytic attack (see Question 58 and Question 59) than ordinary DES. It is estimated by Biham [Bih95] that 2^{61} chosen plaintexts are required for a differential attack on DES with independent subkeys, while 2^{60} known plaintexts are required for linear cryptanalysis.

72. What is Triple-DES?

For some time it has been common practice to protect and transport a key for DES encryption with triple-DES. This means that the plaintext is, in effect, encrypted three times. There are, of course, a variety of ways of doing this; we will explore these ways below. See Question 85 for a discussion of multiple encryption in general.

A number of modes of triple-encryption have been proposed:

- DES-EEE3: Three DES encryptions with three different keys.
- DES-EDE3: Three DES operations in the sequence *encrypt-decrypt-encrypt* with three different keys.
- DES-EEE2 and DES-EDE2: Same the previous formats except that the first and third operations use the same key.

Attacks on two-key triple-DES have been proposed by Merkle and Hellman [MH81] and Van Oorschot and Wiener [VW91], but the data requirements of these attacks make them impractical. Further information on triple-DES can be obtained from various sources [Bih95][KR96].

The use of double and triple encryption does not always provide the additional security that might be expected. Preneel [Pre94] provides the following comparisons in the security of various versions of multiple-DES and it can be seen that the most secure form of multiple encryption is triple-DES with three *distinct* keys.

# of Encryptions	# of Keys	Computation	Storage	Type of Attack
single	1	2^{56}	-	known plaintext
single	1	2^{38}	2^{38}	chosen plaintext
single	1	-	2^{56}	chosen plaintext
double	2	2^{112}	-	known plaintext
double	2	2^{56}	2^{56}	known plaintext
double	2	-	2^{112}	chosen plaintext
triple	2	2^{112}	-	known plaintext
triple	2	2^{56}	2^{56}	2^{56} chosen plaintext
triple	2	2^{120-t}	2^{t}	2^{t} known plaintext
triple	2	-	2^{56}	chosen plaintext
triple	3	2^{112}	2^{56}	known plaintext
triple	3	2^{56}	2^{112}	chosen plaintext

Table 1: Comparison of Different Forms of DES Multiple Encryption

73. How does One Use Triple-DES in CBC Mode?

Until recently, the most significant use of triple-DES (see Question 72) was for the encryption of single DES keys, and there was really no need to consider how one might implement various block cipher modes (see Question 82 and Question 83) when the block cipher in question is actually one derived from multiple encryption. However, as DES nears the end of its useful lifetime (see Question 65), more thought is being given to an increasingly widespread use of triple-DES.

In particular, there are two obvious ways to implement the CBC mode (see Question 82) for triple-DES. With single-DES in CBC mode, the ciphertext is XORed with the plaintext before encryption. With triple-DES however, we might use feedback around all three DES operations from the ciphertext to the plaintext, something which is called outer-CBC. Alternatively, we might run the feedback around each individual encryption component, thereby making, in effect, triple-(DES-CBC). This is referred to as inner-CBC, since there are internal feedbacks that are never seen by the cryptanalyst.

Performance-wise, there can be some advantages to use the inner-CBC option, but research has established that outer-CBC is in fact more secure [Bih95]. Outer-CBC is the recommended way for using triple-DES in the CBC mode.

74. What are G-DES, DESX?

G-DES was devised by Schaumuller-Bichl to improve on the performance of DES by defining a cipher based on DES with a larger block size, but without an increase in the amount of computation required [Sch83]. It was claimed that G-DES was as secure as DES since the cipher was based on DES. However, Biham and Shamir showed that G-DES with the recommended parameter sizes is easily broken and that any alterations of G-DES parameters that result in a cipher faster than DES are less secure than DES [BS93b].

DESX, another variant of DES, is supported by RSA Data Security's toolkits (see Question 173). The only difference between DES and DESX is that the input plaintext is XORed with 64 bits of key material before encryption with DES and the output is XORed with 64 bits of either related or unrelated key material. The security of DESX against differential and linear attack (see Question 58 and Question 59) is equivalent to that of DES with independent subkeys (see Question 71), while the security against exhaustive search is greatly increased.

Other Block Ciphers

75. What is RC2?

RC2 is a variable key-size block cipher designed by Rivest for RSA Data Security. “RC” stands for “Ron’s Code” or “Rivest’s Cipher.” It is faster than DES and is designed as a “drop-in” replacement for DES (see Question 64). It can be made more secure or less secure than DES against exhaustive key search by using appropriate key sizes. It has a block size of 64 bits and is about two to three times faster than DES in software. The algorithm is confidential and proprietary to RSA Data Security. RC2 can be used in the same modes as DES (see Question 66).

An agreement between the Software Publishers Association (SPA) and the United States government gives RC2 and RC4 (see Question 87) special status by means of which the export approval process is simpler and quicker than the usual cryptographic export process. However, to qualify for quick export approval a product must limit the RC2 and RC4 key sizes to 40 bits; 56 bits is allowed for foreign subsidiaries and overseas offices of United States companies. An additional string (40 to 88 bits long) called a *salt* can be used to thwart attackers who try to precompute a large look-up table of possible encryptions. The salt is appended to the encryption key, and this lengthened key is used to encrypt the message; the salt is then sent, unencrypted, with the message. RC2 and RC4 have been widely used by developers who want to export their products; DES is almost never approved for export.

76. What is RC5?

RC5 [Riv95] is a fast block cipher designed by Rivest for RSA Data Security. It is a parameterized algorithm with a variable block size, a variable key size, and a variable number of rounds. The block size can be 32, 64, or 128 bits long. The number of rounds can range from 0 to 255. The key can range from 0 bits to 2048 bits in size. Such built-in variability provides flexibility in levels of security and efficiency.

There are three routines in RC5: *key expansion*, *encryption*, and *decryption*. In the key-expansion routine, the user-provided secret key is expanded to fill a key table whose size depends on the number of rounds. The key table is then used in both encryption and decryption. The encryption routine consists of three primitive operations: addition, bitwise XOR, and rotation. The exceptional simplicity of RC5 makes it easy to implement and analyze. Indeed, like RSA, RC5 can be written on the "back of the envelope" (except for key expansion).

The security of RC5 is provided by the heavy use of data-dependent rotations and the mixture of different operations. In particular, the use of data-dependent rotations helps defeat differential and linear cryptanalysis (see Question 58 and Question 59), and Kaliski and Yin [KY95] found that RC5 with a block size of 64 bits and 12 or more rounds provides good security against differential and linear cryptanalysis.

RSA Data Security is in the process of patent application for RC5.

77. What is IDEA?

IDEA (International Data Encryption Algorithm) [LMM92] is the second version of a block cipher designed and presented by Lai and Massey [LM91a]. It is a 64-bit iterative block cipher with a 128-bit key and eight rounds. While the cipher is not Feistel (see Question 56), decryption is carried out in the same manner as encryption once the decryption subkeys have been calculated from the encryption subkeys. The cipher structure was designed to be easily implemented in both software and hardware, and the security of IDEA relies on the use of three incompatible types of arithmetic operations on 16-bit words. The speed of IDEA in software is similar to that of DES.

One of the principles during the design of IDEA was to facilitate analysis of its strength against differential cryptanalysis (see Question 58); IDEA is considered to be immune from differential cryptanalysis. In addition, no linear cryptanalytic attacks on IDEA have been reported and there is no known algebraic weakness in IDEA. The most significant cryptanalytic result is due to Daemen [DGV94]. He discovered a large class of 2^{51} weak keys (see Question 60) for which the use of such a key during encryption could be detected and the key recovered. However, since there are 2^{128} possible keys, this result has no impact on the practical security of the cipher for encryption. IDEA is generally considered secure and both the cipher development and its theoretical basis have been openly and widely discussed.

78. What is SAFER?

SAFER (Secure And Fast Encryption Routine) is a non-proprietary block cipher developed by Massey in 1993 for Cylink Corporation [Mas93]. It is a byte-oriented algorithm with a 64-bit block size and, in one version, a 64-bit key size. It has a variable number of rounds (maximum of 10), but a minimum of six rounds is recommended. Unlike most recent block ciphers, SAFER has slightly different encryption and decryption procedures. Only byte-based operations are employed to ensure its utility in smart card-based applications that have limited processing power. When first announced, SAFER was intended to be implemented with a key of length 64 bits and it was accordingly named SAFER K-64. Another version of SAFER was designed that could handle 128-bit keys and was named SAFER K-128.

Early cryptanalysis of SAFER K-64 [Mas93] showed that SAFER K-64 could be considered immune to both differential and linear cryptanalysis (see Question 58 and Question 59) when the number of rounds is greater than six. Knudsen [Knu95] discovered a weakness in the key schedule of SAFER K-64 and a new key schedule for the family of SAFER ciphers soon followed. These new versions of SAFER are denoted SAFER SK-64 and SAFER SK-128 where SK denotes a strengthened key schedule. Most recently, a version of SAFER called SAFER SK-40 was announced, which uses a 40-bit key and has five rounds (thereby increasing the speed of encryption). This reduced-round version is secure against differential and linear cryptanalysis in the sense that any such attack would require more effort than a brute-force search for a 40-bit key.

79. What is FEAL?

The Fast Data Encipherment Algorithm (FEAL) was presented by Shimizu and Miyaguchi [SM88] as an alternative to DES. The original cipher (called FEAL-4) was a four-round cryptosystem with a 64-bit block size and a 64-bit key size and it was designed to give high performance in software. Soon a variety of attacks against FEAL-4 were announced including one attack that required only 20 chosen plaintexts [Mur90]. Several results in the cryptanalysis of FEAL-8 (eight-round version) led the designers to introduce a revised version, FEAL-N, where N denoted the number of rounds. Biham and Shamir [BS91b] developed differential cryptanalytic attacks against FEAL-N for up to 31 rounds. In 1994, Ohta and Aoki presented a linear cryptanalytic attack against FEAL-8 that required 2^{25} known plaintexts [OA94], and other improvements [KR95a] followed. In the wake of these numerous attacks, FEAL and its derivatives should be considered insecure.

80. What is Skipjack?

Skipjack is the encryption algorithm contained in the Clipper chip (see Question 151), and it was designed by the NSA (see Question 148). It uses an 80-bit key to encrypt 64-bit blocks of data. Skipjack can be more secure than DES (see Question 64), since it uses 80-bit keys and scrambles the data for 32 steps, or “rounds." By contrast, DES uses 56-bit keys and scrambles the data for only 16 rounds.

The details of Skipjack are classified. The decision not to make the details of the algorithm publicly available has been widely criticized. Many people are suspicious that Skipjack is not secure, either due to oversight by its designers, or by the deliberate introduction of a secret trapdoor. By contrast, there have been many attempts to find weaknesses in DES over the years, since its details are public. These numerous attempts (and the fact that they have failed) have made people confident in the security of DES. Since Skipjack is not public, the same scrutiny cannot be applied towards it, and thus a corresponding level of confidence may not arise.

Aware of such criticism, the government invited a small group of independent cryptographers to examine the Skipjack algorithm. They issued a report [BDK93] which stated that, although their study was too limited to reach a definitive conclusion, they nevertheless believed that Skipjack was secure.

Another consequence of Skipjack’s classified status is that it cannot be implemented in software, but only in hardware by government-authorized chip manufacturers. An algorithm called S1 was anonymously posted over the Internet in summer 1995, and it was claimed that S1 was the Skipjack algorithm. It is believed, however, to be a hoax.

81. What is Blowfish?

Blowfish is a 64-bit block cipher developed by Schneier [Sch93]. It is a Feistel cipher (see Question 56) and each round consists of a key-dependent permutation and a key-and-data-dependent substitution. All operations are based on XORs and additions on 32-bit words. The key has a variable length (with a maximum length of 448 bits) and is used to generate several subkey arrays. This cipher was designed specifically for 32-bit machines and is significantly faster than DES. There was an open competition for the cryptanalysis of Blowfish supported by *Dr. Dobb's Journal* with a $1000 prize. This contest ended in April 1995 [Sch95a] and among the results were the discoveries of existence of certain weak keys (see Question 60), an attack against a three-round version of Blowfish, and a differential attack against certain variants of Blowfish. However, Blowfish can still be considered secure, and Schneier has invited cryptanalysts to continue investigating his cipher.

Modes and Multiple Encryption

82. What are the ECB and CBC Modes?

When we use a block cipher to encrypt a message of arbitrary length, we use techniques that are known as *modes of operation* for the block cipher. Modes must be at least as secure and as efficient as the underlying cipher. Modes may have properties in addition to those inherent in the basic cipher. The standard DES modes (see Question 66) have been published in FIPS PUB 81 [NIS80] and as ANSI X3.106 [ANS83]. A more general version of the standard [ISO92b] generalized the four modes of DES to be applicable to a block cipher of any block size. The standard modes are Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), and Output Feedback (OFB).

In *ECB mode* (see Figure 3), each plaintext block is encrypted independently with the block cipher.

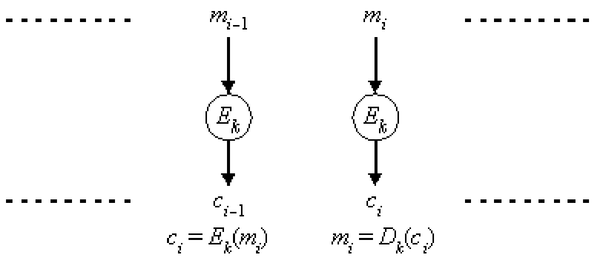

Figure 3. Electronic Code Book Mode

ECB mode is as secure as the underlying block cipher. However, plaintext patterns are not concealed. Each identical block of plaintext gives an identical block of ciphertext. The plaintext can be easily manipulated by removing, repeating, or interchanging blocks. The speed of each encryption operation is identical to that of the block cipher. ECB allows easy parallelization to yield higher performance. Unfortunately, no processing is possible before a block is seen (except for key setup).

In *CBC mode* (see Figure 4), each plaintext block is XORed with previous ciphertext block, then encrypted. An *initialization vector* or value c_0 is used as a "seed" for the process.

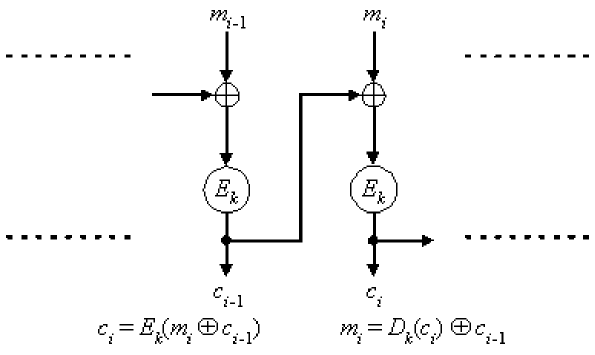
m_{i-1}
m_i
E_k
E_k
c_{i-1}
c_i
$c_i = E_k(m_i \oplus c_{i-1})$
$m_i = D_k(c_i) \oplus c_{i-1}$

Figure 4. Cipher Block Chaining Mode

CBC mode is as secure as the underlying block cipher against standard attacks. In addition, any patterns in the plaintext are concealed by the XORing of the previous ciphertext block with the plaintext block. Security of the ciphertext is enhanced as the plaintext cannot be directly manipulated except by removal of blocks from the beginning or the end of the ciphertext. The speed of encryption is identical to that of the block cipher, but the encryption process cannot be easily parallelized though the decryption process can be parallelized.

83. What are the CFB and OFB modes?

The Cipher Feedback (CFB) mode and the Output Feedback (OFB) mode are two more standard modes of operation (see Question 82) for a block cipher.

In *CFB mode* (see Figure 5), the previous ciphertext block is encrypted and the output produced is combined with the plaintext block using XOR to produce the current ciphertext block. It is possible to define CFB mode so that it uses feedback that is less than one full data block. An initialization vector (see Question 82) or value c_0 is used as a "seed" for the process.

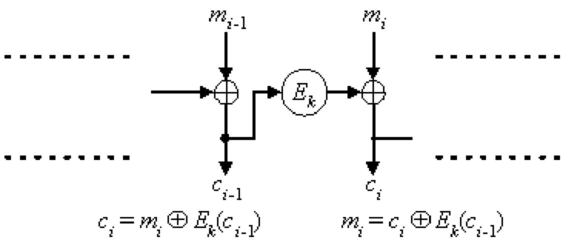
m_{i-1}
m_i
E_k
c_{i-1}
c_i
$c_i = m_i \oplus E_k(c_{i-1})$
$m_i = c_i \oplus E_k(c_{i-1})$

Figure 5. Cipher Feedback Mode

CFB mode is as secure as the underlying cipher and plaintext patterns are concealed in the ciphertext by the use of the XOR operation. Plaintext cannot be manipulated directly except by the removal of blocks from the beginning or the end of the ciphertext. With CFB mode and full feedback, when two ciphertext blocks are identical, the outputs from the DES operation at the next step are also identical. This allows information about plaintext blocks to leak. When using full feedback, the speed of encryption is identical to that of the block cipher, but the encryption process cannot be easily parallelized.

OFB mode (see Figure 6) is similar to the CFB mode except that the quantity XORed with each plaintext block is generated independently of both the plaintext and ciphertext. An initialization vector s_0 is used as a "seed" for a sequence of data blocks

s_i, and each data block s_i is derived from the encryption of the previous data block s_{i-1}. The encryption of a plaintext block is derived by taking the XOR of the plaintext block with the relevant data block.

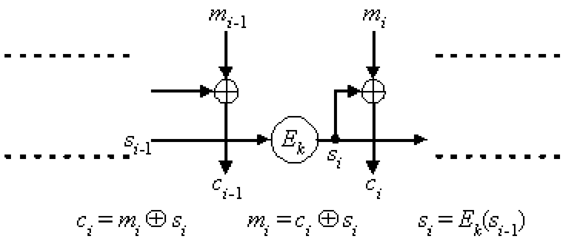

Figure 6. Output Feedback Mode

Feedback widths less than a full block are not recommended for security [DP83][Jue83]. OFB mode has an advantage over CFB mode in that any bit errors that might occur during transmission are not propagated to affect the decryption of subsequent blocks. However, by changing the ciphertext the plaintext can easily be manipulated. The speed of encryption is identical to that of the block cipher. Even though the process cannot easily parallelized, time can be saved by generating the key stream before the data is available for encryption.

84. What are the Counter and PCBC Modes?

Due to shortcomings in OFB mode (see Question 83) Diffie has proposed [Bra88] an additional mode of operation, termed the *counter mode*. It differs from OFB mode in the way the successive data blocks are generated for subsequent encryptions. Instead of deriving one data block as the encryption of the previous data block, Diffie proposed encrypting the quantity $i + IV$ (mod 2^{64}) for the ith data block, where IV is some initialization vector.

The Propagating Cipher Block Chaining (PCBC) mode of encryption is another mode of operation using block ciphers. It is used in protocols such as Kerberos version 4 (see Question 144). The PCBC mode of encryption has not been formally published as a federal or national standard, and it does not have widespread general support. The PCBC mode is a variation on the CBC mode of operation and is designed to extend or propagate a single bit error in the ciphertext. This allows errors in transmission to be captured and the resultant plaintext to be rejected. The method of encryption is given by

$$c_i = E_k(m_i \oplus m_{i-1} \oplus c_{i-1})$$

and decryption is achieved by computing

$$m_i = D_k(c_i) \oplus c_{i-1} \oplus m_{i-1}$$

where $m_0 \oplus c_0$ is the initialization vector.

85. What is Multiple Encryption?

Intuitively, we might expect that by encrypting a message twice with some block cipher (either with the same key or by using two different keys), then we would expect the resultant encryption to be stronger in all but some exceptional circumstances. By using three encryptions, we would expect to achieve a yet greater level of security. While there are some more complicated issues to consider (see Question 61), this is pretty much the case. Triple-DES has been used for a considerable time as a more secure cipher for protecting the keys used with single-DES. However, there are some surprising results when we consider exactly how much additional protection is provided by using double and triple encryption.

For instance, the use of double encryption does not provide the expected increase in security [MH81] when compared with the increased implementation requirements, and it cannot be recommended as a good alternative. Instead, triple-encryption is the point at which multiple encryption gives substantial improvements in security. For a more detailed consideration of the situation with DES, see Question 72; for more information on multiple encryption in general see a survey article by Kaliski and Robshaw [KR96].

STREAM CIPHERS

86. What is a Stream Cipher?

A stream cipher is a symmetric encryption algorithm. Stream ciphers can be designed to be exceptionally fast, much faster in fact than any block cipher (see Question 54). While block ciphers operate on large blocks of data, stream ciphers typically operate on smaller units of plaintext, usually bits. The encryption of any particular plaintext with a block cipher will result in the same ciphertext when the same key is used. With a stream cipher, the transformation of these smaller plaintext units will vary, depending on when they are encountered during the encryption process.

A stream cipher generates what is called a *keystream* and encryption is provided by combining the keystream with the plaintext, usually with the bitwise XOR operation. The generation of the keystream can be independent of the plaintext and ciphertext (yielding what is termed a *synchronous stream cipher*) or it can depend on the data and its encryption (in which case the stream cipher is said to be *self-synchronizing*). Most stream cipher designs are for synchronous stream ciphers.

Current interest in stream ciphers is most commonly attributed to the appealing theoretical properties of the one-time pad (see Question 93), but there have been, as of yet, no attempts to standardize on any particular stream cipher proposal as has been the case with block ciphers (see Question 64). Interestingly, certain modes of operation of a block cipher (see Question 83) effectively transform it into a keystream generator and in this way, any block cipher can be used as a stream cipher. However, stream ciphers with a dedicated design are likely to be much faster.

More information about stream ciphers and the various available algorithms can be found in almost any book on contemporary cryptography and in RSA Laboratories Technical Report [Rob95b].

87. What is RC4?

RC4 is a stream cipher designed by Rivest for RSA Data Security. It is a variable key-size stream cipher with byte-oriented operations. The algorithm is based on the use of a random permutation and analysis shows that the period of the cipher is overwhelmingly likely to be greater than 10^{100}. Eight to sixteen machine operations are required per output byte, and the cipher can be expected to run very quickly in software. While the algorithm is confidential and proprietary to RSA Data Security, Inc., it has been scrutinized under conditions of non-disclosure by independent analysts and it is considered secure. The RC4 stream cipher has a special status by which export from the U.S. can often be facilitated (see Question 75).

88. What is SEAL?

The Software-optimized Encryption Algorithm (SEAL) was designed by Rogaway and Coppersmith in 1993 [RC93] as a fast stream cipher for 32-bit machines. SEAL has a rather involved initialization phase during which a large set of tables is initialized using the Secure Hash Algorithm (see Question 100). However, the use of look-up tables during keystream generation helps to achieve a very fast performance with just five instructions required per byte of output generated.

89. What is a Linear Feedback Shift Register?

A *Linear Feedback Shift Register* (LFSR) is a mechanism for generating a sequence of binary bits. The register (see Figure 7) consists of a series of cells that are set by an initialization vector that is, most often, the secret key. The behavior of the register is regulated by a clock and at each clocking instant, the contents of the cells of the register are shifted right by one position, and the XOR of a subset of the cell contents is placed in the leftmost cell. One bit of output is usually derived during this update procedure.

LFSRs are fast and easy to implement in both hardware and software. With a judicious choice of *feedback taps* the sequences that are generated can have a good statistical appearance. However, the sequences generated by single LFSRs are not secure because a powerful mathematical framework has been developed over the years which allows for their straightforward analysis. However, LFSRs are useful as building blocks in more secure systems (see Question 90, Question 91 and Question 92).

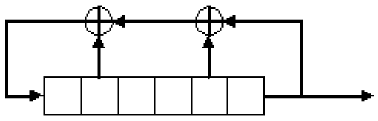

Figure 7. LFSR

90. What are Shift Register Cascades?

A *shift register cascade* is a set of LFSRs (see Question 89) connected together in such a way that the behavior of one particular LFSR depends on the behavior of the previous LFSRs in the cascade. This dependent behavior is usually achieved by using one LFSR to control the clock of the following LFSR. For instance one register might be advanced by one step if the preceding register output is 1 and advanced by two steps otherwise. Many different configurations are possible and certain parameter choices appear to offer very good security. For more detail, see an excellent survey article by Gollman and Chambers [GC89].

91. What are the Shrinking and Self-Shrinking Generators?

The *shrinking generator* was developed by Coppersmith, Krawczyk, and Mansour [CKM94]. It is a stream cipher based on the simple interaction between the outputs from two LFSRs (see Question 89). The bits of one output are used to determine whether the corresponding bits of the second output will be used as part of the overall keystream. The shrinking generator is simple and scaleable, and has good security properties. One drawback of the shrinking generator is that the output rate of the keystream will not be constant unless precautions are taken. A variant of the shrinking generator is the *self-shrinking generator* [MS95], where instead of using one output from one LFSR to “shrink” the output of another (as in the shrinking generator), the output of a single LFSR is used to extract bits from the same output. There are as yet no results on the cryptanalysis of either technique.

92. What Other Stream Ciphers Are There?

There are a vast number of alternative stream ciphers that have been proposed in cryptographic literature as well as an equally vast number that appear in implementations and products world-wide. Many are based on the use of LFSRs (see Question 89) since such ciphers tend to be more amenable to analysis and it is easier to assess the security that they offer.

Rueppel suggests that there are essentially four distinct approaches to stream cipher design [Rue92]. The first is termed the *information-theoretic* approach as exemplified by Shannon's analysis of the one-time pad (see Question 93). The second approach is that of *system-theoretic* design. In essence, the cryptographer designs the cipher along established guidelines which ensure that the cipher is resistant to all known attacks. While there is, of course, no substantial guarantee that future cryptanalysis will be unsuccessful, it is this design approach that is perhaps the most common in cipher design. The third approach is to attempt to relate the difficulty of breaking the stream cipher (where "breaking" means being able to predict the unseen keystream with a success rate better than can be achieved by guessing) to solving some difficult problem (see [BM84][BBS86]). This *complexity-theoretic* approach is very appealing, but in practice the ciphers that have been developed tend to be rather slow and impractical. The final approach highlighted by Rueppel is that of designing a *randomized cipher*. Here the aim is to ensure that the cipher is resistant to any practical amount of cryptanalytic work rather than being secure against an unlimited amount of work, as was the aim with Shannon's information-theoretic approach.

See Rueppel's article [Rue92] or any book on contemporary cryptography for examples of ciphers in each of these categories. More details are also provided in RSALaboratories Technical Report [Rob95b].

93. What is a One-time Pad?

A *one-time pad*, sometimes called the *Vernam cipher* [Ver26], uses a string of bits that is generated completely at random. The keystream is the same length as the plaintext message and the random string is combined using bitwise XOR with the plaintext to produce the ciphertext. Since the entire keystream is random, an opponent with infinite computational resources can only guess the plaintext if he sees the ciphertext. Such a cipher is said to offer perfect secrecy and the analysis of the one-time pad is seen as one of the cornerstones of modern cryptography [Sha49].

While the one-time pad saw use during wartime, over diplomatic channels requiring exceptionally high security, the fact that the secret key (which can be used only once) is as long as the message introduces severe key-management problems. While perfectly secure, the one-time pad is impractical.

Stream ciphers (see Question 86) were developed as an approximation to the action of the one-time pad, and while contemporary stream ciphers are unable to provide the satisfying theoretical security of the one-time pad, they are at least practical.

HASH FUNCTIONS

94. What is a Hash Function?

A hash function H is a transformation that takes a variable-size input m and returns a fixed-size string, which is called the hash value h (that is, $h = H(m)$). Hash functions with just this property have a variety of general computational uses, but when employed in cryptography the hash functions are usually chosen to have some additional properties.

The basic requirements for a cryptographic hash function are:

- the input can be of any length,
- the output has a fixed length,
- $H(x)$ is relatively easy to compute for any given x,
- $H(x)$ is one-way,
- $H(x)$ is collision-free.

A hash function H is said to be *one-way* if it is hard to invert, where "hard to invert" means that given a hash value h, it is computationally infeasible to find some input x such that $H(x) = h$.

If, given a message x, it is computationally infeasible to find a message y ı x such that $H(x) = H(y)$ then H is said to be a *weakly collision-free* hash function.

A *strongly collision-free* hash function H is one for which it is computationally infeasible to find any two messages x and y such that $H(x) = H(y)$.

The hash value represents concisely the longer message or document from which it was computed; one can think of a message digest as a "digital fingerprint" of the larger document. Examples of well-known hash functions are MD2 and MD5 (see Question 99) and SHA (see Question 100).

Perhaps the main role of a cryptographic hash function is in the provision of digital signatures. Since hash functions are generally faster than digital signature algorithms, it is typical to compute the digital signature to some document by computing the signature on the document's hash value, which is small compared to the document itself. Additionally, a digest can be made public without revealing the contents of the document from which it is derived. This is important in digital timestamping (see Question 108) where, using hash functions, one can get a document timestamped without revealing its contents to the timestamping service.

For more information and a particularly thorough study of hash functions, see Preneel [Pre93].

95. What is a Birthday Attack?

A *birthday attack* is a name used to refer to a class of brute-force attacks. It gets its name from the surprising result that the probability that two or more people in a group of 23 share the same birthday is greater than 1/2; such a result is called a *birthday paradox*.

If some function, when supplied with a random input, returns one of k equally-likely values, then by repeatedly evaluating the function for different inputs, we expect to obtain the same output after about $1.2k^{1/2}$. For the above birthday paradox, replace k with 365.

Birthday attacks are often used to find collisions of hash functions (see Question 96).

96. How Does the Length of a Hash Value Affect Security?

The essential cryptographic properties of a hash function are that it is both one-way and collision-free (see Question 94). The most basic attack we might mount on a hash function is to choose inputs to the hash function at random until either we find some input that will give us the target output value we are looking for (thereby contradicting the one-way property), or we find two inputs that produce the same output (thereby contradicting the collision-free property).

Suppose the hash function produces an *n*-bit long output. If we are trying to find some input which will produce some target output value *y*, then since each output is equally likely we expect to have to try 2^n possible input values.

If we are trying to find a collision, then by the birthday paradox (see Question 95) we would expect that after trying $2^{n/2}$ possible input values we would have some collision. Van Oorschot and Wiener [VW94] showed how such a brute-force attack might be implemented.

With regard to the use of hash functions in the provision of digital signatures, Yuval [Yuv79] proposed the following strategy based on the birthday paradox, where *n* is the length of the message digest:

- Adversary selects the target message to be signed and an innocuous message that Alice is likely to want to sign.
- The adversary generates $2^{n/2}$ variations of the innocuous message (by making, for instance, minor editorial changes), all of which convey the same meaning, and their corresponding message digests. He then generates an equal number of variations of the target message to be substituted.
- The probability that one of the variations of the innocuous message will match one of the variations of the target message is greater than 1/2 according to the birthday paradox.
- The adversary then obtains Alice's signature on the variation of the innocuous message.
- The signature from the innocuous message is removed and attached to the variation of the target message that generates the same message digest. The adversary has successfully forged the message without discovering the enciphering keys.

To avoid an attack that depends on brute-force methods, the output from the hash function must be sufficiently long.

97. What is a Compression Function?

Damgård and Merkle [Dam90][Mer90a] greatly influenced cryptographic hash function design by defining a hash function in terms of what is called a compression function. A *compression function* takes a fixed length input and returns a shorter, fixed-length output. Then a hash function can be defined by means of repeated applications of the compression function until the entire message has been processed. In this process, a message of arbitrary length is broken into blocks of a certain length which depends on the compression function, and "padded" (for security reasons) so that the size of the message is a multiple of the block size. The blocks are then processed sequentially, taking as input the result of the hash so far and the current message block, with the final output being the hash value for the message (see Figure 8).

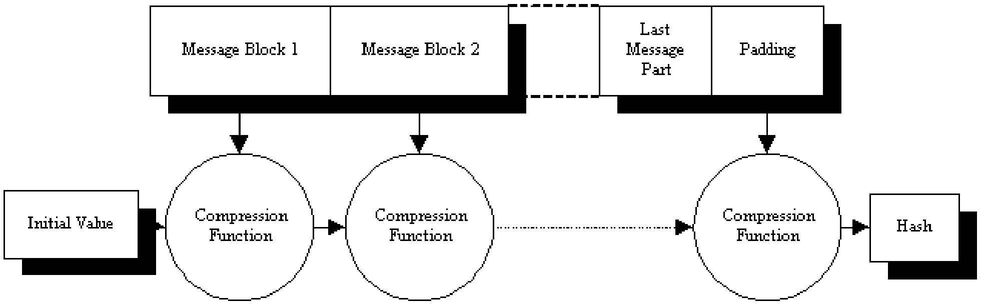

Figure 8. Damgård/Merkle iterative structure for hash functions

98. What are Pseudo-Collisions?

Pseudo-collisions are collisions for the compression function (see Question 97) that lies at the heart of an iterative hash function. While collisions for the compression function of a hash function might be useful in constructing collisions for the hash function itself,this is not normally the case. While pseudo-collisions might be viewed as an unfortunate property of a hash function, a pseudo-collision is not equivalent to a collision, and the hash function can still be secure. MD5 (see Question 99) is an example of a hash function for which pseudo-collisions have been discovered and yet is still considered secure.

99. What are MD2, MD4 and MD5?

MD2 [Kal92], MD4 [Riv91b][Riv92b], and MD5 [Riv92c] are message-digest algorithms developed by Rivest. They are meant for digital signature applications where a large message has to be "compressed" in a secure manner before being signed with the private key. All three algorithms take a message of arbitrary length and produce a 128-bit message digest. While the structures of these algorithms are somewhat similar, the design of MD2 is quite different from that of MD4 and MD5 and MD2 was optimized for 8-bit machines, whereas MD4 and MD5 were aimed at 32-bit machines. Description and source code for the three algorithms can be found as Internet RFCs 1319 - 1321 [Kal92] [Riv92b][Riv92c].

MD2 was developed by Rivest in 1989. The message is first padded so that its length in bytes is divisible by 16. A 16-byte checksum is then appended to the message, and the hash value is computed on this resulting message. Rogier and Chauvaud have found that collisions for MD2 can be constructed if the calculation of the checksum is omitted [RC95]. This is the only cryptanalytic result known for MD2.

MD4 was developed by Rivest in 1990. The message is padded to ensure that its length in bits plus 448 is divisible 512. A 64-bit binary representation of the original length of the message is then concatenated to the message. The message is processed in 512-bit blocks in the Damgård/Merkle iterative structure (see Question 97), and each block is processed in three distinct rounds. Attacks on versions of MD4 with either the first or the last rounds missing were developed very quickly by Den Boer and Bosselaers [DB92] and others. Dobbertin [Dob95] has shown how collisions for the full version of MD4 can be found in under a minute on a typical PC. Clearly, MD4 should now be considered broken.

MD5 was developed by Rivest in 1991. It is basically MD4 with "safety-belts" and while it is slightly slower than MD4, it is more secure. The algorithm consists of four distinct rounds, which have a slightly different design from that of MD4. Message-digest size, as well as padding requirements, remains the same. Den Boer and Bosselaers [DB94] have found pseudo-collisions for MD5 (see Question 98), but there are no other known cryptanalytic results.

Van Oorschot and Wiener [VW94] have considered a brute-force search for collisions (see Question 96) in hash functions, and they estimate that a collision search machine designed specifically for MD5 (costing $10 million in 1994) could find a collision for MD5 in 24 days on average. The general techniques can be applied to other hash functions.

More details on MD2, MD4, and MD5 can be found in [Pre93] and [Rob95c].

100. What is the Secure Hash Algorithm (SHA and SHA-1)?

The Secure Hash Algorithm (SHA), the algorithm specified in the Secure Hash Standard (SHS), was developed by NIST (see Question 146) and published as a federal information processing standard (FIPS PUB 180) [NIS93a]. SHA-1 [NIS94c] was a revision to SHA that was published in 1994. The revision corrected an unpublished flaw in SHA. Its design is very similar to the MD4 family of hash functions developed by Rivest (see Question 99).

The algorithm takes a message of less than 2^{64} bits in length and produces a 160-bit message digest. The algorithm is slightly slower than MD5 (see Question 99), but the larger message digest makes it more secure against brute-force collision and inversion attacks (see Question 96). SHA is part of the Capstone project (see Question 150). For further information on SHA, see [Pre93] and [Rob95c].

101. What Other Hash Functions Are There?

The best review of hash function techniques is provided by Preneel [Pre93].

For a brief overview here, we note that hash functions are often divided into three classes according to their design:

- Those built around block ciphers.
- Those which use modular arithmetic.
- Those which have what is termed a "dedicated" design.

By building a hash function around a block cipher, it is intended that by using a well-trusted block cipher such as DES (see Question 64) a secure and well-trusted hash function can be obtained. The so-called Davies-Meyer hash function [Pre93] is an example of a hash function built around the use of DES.

The use of modular arithmetic in the second class of hash functions is intended to save on implementation costs when the hash function is used in conjunction with a digital signature algorithm that also uses modular arithmetic. Unfortunately, the track record of such hash functions is not good from a security perspective and there are no hash functions in this second class that can be recommended today.

The hash functions in the third class, with their so-called "dedicated" design, tend to be fast, giving a considerable advantage over algorithms that are based around the use of a block cipher. Along with MD5 and SHA-1 (see Question 99 and Question 100) RIPE-MD [DBP96] and HAVAL [ZPS93] are dedicated hash functions that also have a similar design to MD4.

OTHER TECHNIQUES

Message Authentication Codes

102. What are Message Authentication Codes (MACs)?

A *message authentication code* (MAC) is an authentication tag (also called a checksum) derived by application of an authentication scheme, together with a secret key, to a message. MACs are computed and verified with the same key so they can only be verified by the intended receiver, unlike digital signatures. MACs can be categorized as unconditionally secure, hash function-based, stream cipher-based, or block cipher-based.

Simmons and Stinson [Sti95] proposed an unconditionally secure MAC that is based on encryption with a one-time pad. The ciphertext of the message authenticates itself, as nobody else has access to the one-time pad. However, there has to be some redundancy in the message. An unconditionally secure MAC can also be obtained by use of a one-time secret key.

Hash function-based MACs use a key or keys in conjunction with a hash function (see Question 94) to produce a checksum that is appended to the message. An example is the keyed-MD5 (see Question 99) method of message authentication [KR95b].

Lai, Rueppel, and Woolven [LRW92] proposed a MAC based on stream ciphers (see Question 86). In their algorithm, a provably secure stream cipher is used to split a message into two substreams and each substream is fed into a LFSR (see Question 89); the checksum is the final state of the two LFSRs.

MACs can also be derived from block ciphers (see Question 54). The DES-CBC MAC is a widely used US and international standard [NIS85]. The basic idea is to encrypt the message blocks using DES CBC (see Question 82) and output the final block in the ciphertext as the checksum. Bellare *et al.* give an analysis of the security of this MAC [BKR94].

Secret Sharing

103. What are Secret Sharing Schemes?

Secret sharing schemes were discovered independently by Blakley [Bla79] and Shamir [Sha79]. The motivation for secret sharing is secure key management. In some situations, there is usually *one* secret key that provides access to many important files. If such a key is lost (e.g., the person who knows the key becomes unavailable, or the computer which stores the key is destroyed), then all the important files become inaccessible. The basic idea in *secret sharing* is to divide the secret key into pieces and distribute the pieces to different persons so that certain subsets of the persons can get together to recover the key.

The general model for secret sharing is called an *m-out-of-n scheme* (or *(m,n)-threshold scheme*) for integers $1 \leq m \leq n$. In the scheme, there is a *sender* (or *dealer*) and n *participants*. The sender divides the secret into n parts and gives each participant one part so that any m parts can be put together to recover the secret, but any $m - 1$ parts reveal no information about the secret. The pieces are usually called *shares* or *shadows.* Different choices for the values of m and n reflect the trade-off between security and reliability. A secret sharing scheme is *perfect* if any group of at most $m - 1$ participants (insiders) has no advantage in guessing the secret over the outsiders.

Both Shamir's scheme (see Question 104) and Blakley's scheme (see Question 105) are m-out-of-n secret sharing schemes. They represent two different ways of constructing such schemes, based on which more advanced secret sharing schemes can be designed. For further information on secret sharing schemes, see [Sim92].

104. What is Shamir's Secret Sharing Scheme?

Shamir's secret sharing scheme [Sha79] is an *interpolating scheme* based on polynomial interpolation. An (m – 1)-degree polynomial over the finite field GF(q)

$$F(x) = a_0 + a_1x + \dots + a_{m-1}\, x^{m-1}$$

is constructed such that the coefficient a_0 is the secret and all other coefficients are random elements in the field. (The field is known to all participants.) Each of the n shares is a point (x_i, y_i) on the curve defined by the polynomial, where $x_i \neq 0$. Given any m shares, the polynomial is uniquely determined and hence the secret a_0 can be computed. However, given m – 1 or fewer shares, the secret can be any element in the field. Therefore, Shamir's scheme is a perfect secret sharing scheme (see Question 103).

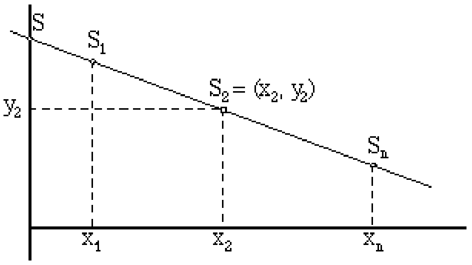

Figure 9. Shamir's secret sharing scheme

A special case where m = 2 (i.e., two shares are required for retrieval of the secret) is given in Figure 9. The polynomial is a line and the secret is the point where the line intersects with the y-axis. Each share is a point on the line. Any two shares (two points) determine the line and hence the secret. With just a single share (point), the line can be any line that passes the point, and hence the secret can be any point on the y-axis.

105. What is Blakley's Secret Sharing Scheme?

Blakley's secret sharing scheme [Bla79] is geometric in nature. The secret is a point in an *m*-dimensional space. *n* shares are constructed with each share defining a hyperplane in this space. By finding the intersection of any *m* of these planes, the secret (or point of intersection) can be obtained. This scheme is not perfect, as the person with a share of the secret knows that the secret is a point on his hyperplane. Nevertheless, this scheme can be modified to achieve perfect security [Sim92].

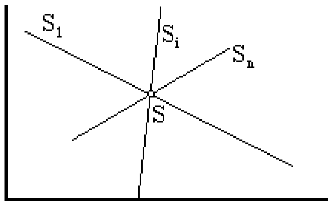

Figure 10. Blakley's scheme

A special case of Blakley's scheme is shown in Figure 10. This is based on the scenario where two shares are required to recover the secret. A two-dimensional plane is used as only two shares are required to recover the secret. The secret is a point in the plane. Each share is a line that passes through the point. If any two of the shares are put together, the point of intersection, which is the secret, can be easily derived.

106. What are Visual Secret Sharing Schemes?

Naor and Shamir [NS94] developed what they called *visual secret sharing schemes*, which are an interesting visual variant of the ordinary secret sharing schemes.

Roughly speaking, the problem can be formulated as follows: There is a secret picture to be shared among *n* participants. The picture is divided into *n* transparencies (shares) such that if any *m* transparencies are placed together, the picture becomes visible, but if fewer than *m* transparencies are placed together, nothing can be seen. Such a scheme is constructed by viewing the secret picture as a set of black and white pixels and handling each pixel separately. See [NS94] for more details. The schemes are perfectly secure and easily implemented without any cryptographic computation. A further improvement allows each transparency (share) to be an innocent picture (e.g. a picture of a landscape or a picture of a building), thus concealing the fact that secret sharing is taking place.

Zero Knowledge

107. What are Interactive Proofs and Zero-Knowledge Proofs?

Informally, an *interactive proof* is a protocol between two parties in which one party, called the *prover,* tries to prove a certain fact to the other party, called the *verifier.* An interactive proof usually takes the form of a challenge-response protocol, in which the prover and the verifier exchange messages and the verifier outputs either "accept" or "reject" at the end of the protocol. Besides their theoretical interests, interactive proofs have found applications in cryptography and computer security such as identification and authentication. In these situations, the fact to be proved is usually related to the prover's identity (e.g., the prover's private key).

The following properties of interactive proofs are quite useful, especially in cryptographic applications:

- *Completeness: T*he verifier always accepts the proof if the prover knows the fact and both the prover and the verifier follow the protocol.
- *Soundness:* The verifier always rejects the proof if the prover does not know the fact, as long as the verifier follows the protocol.
- *Zero knowledge:* The verifier learns nothing about the fact being proved (except that it is correct) from the prover that he could not already learn without the prover. In a zero-knowledge proof, the verifier cannot even later prove the fact to anyone else.

A typical round in a zero-knowledge proof consists of a "commitment" message from the prover, followed by a challenge from the verifier, and then a response to the challenge from the prover. The protocol may be repeated for many rounds. Based on the prover's responses in all the rounds, the verifier decides whether to accept or reject the proof.

Let us consider an intuitive example called Ali Baba's Cave [QG90] (see Figure 11).

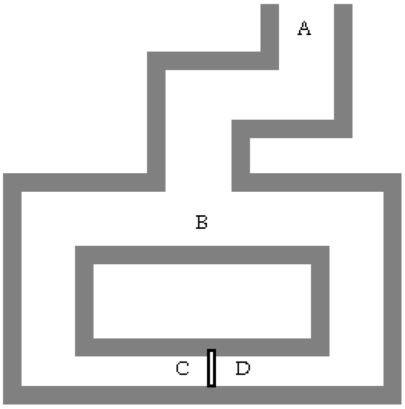

Figure 11. Ali Baba's Cave

Alice wants to prove to Bob that she knows the secret words that will open the portal at CD in the cave, but she does not wish to reveal the secret to Bob. In this scenario, Alice's commitment is to go to C or D. A typical round in the proof proceeds as follows: Bob goes to A and waits there while Alice goes to C or D. Bob then goes to Band shouts to ask Alice to appear from either the right side or the left side of the tunnel. If Alice does not know the secret words (e.g., "Open Sesame"), there is only a 50 percent chance that she will come out from the right tunnel. Bob will repeat this round as many times as he desires until he is certain that Alice knows the secret words. No matter how many times that the proof repeats, Bob does not learn the secret words.

There are a number of zero-knowledge protocols in use today as identification schemes. The Fiat-Shamir protocol [FS87] is the first practical zero-knowledge protocol with cryptographic applications and is based on the difficulty of factoring. A more common variation of the Fiat-Shamir protocol is the Feige-Fiat-Shamir scheme [FFS88]. Guillou and Quisquater [GQ88] further improved Fiat-Shamir's protocol in terms of memory requirements and interaction (the number of rounds in the protocol).

Zero-knowledge identification schemes can usually be modified into digital signature schemes (see Question 3 and [FS87]).

Digital Timestamping

108. How do Digital Timestamps Support Digital Signatures?

Consider two questions that may be asked by a computer user as he or she views a digital document or on-line record. (1) Who is the author of this record — who wrote it, approved it, or consented to it? (2) When was this record created or last modified?

In both cases, the question is one about exactly *this* record — exactly this sequence of bits — whether it was first stored on this computer or was created somewhere else and then copied and saved here. An answer to the first question tells *who & what*; who approved exactly what is in this record? An answer to the second question tells *when & what*; when exactly did the contents of this record first exist?

Both of the above questions have good solutions. A system for answering the first question is called a *digital signature scheme* (see Question 3). Such a system was first proposed in [DH76] and there is a wide variety of accepted designs for an implementation of this kind of system [NIS94b][RSA78].

A system for answering the second question is called a *digital timestamping scheme*. Such systems were described in [BHS93][HS91], and an implementation is commercially available from Surety Technologies (`<http://www.surety.com/>`).

Any system allowing users to answer these questions reliably for all their records must include two different sorts of procedures. First, there must be a *certification* procedure with which (1) the author of a record can "sign" the record, or (2) any user can fix a record in time. The result of this procedure is a small certifying file, a *certificate* if you will, that captures the result of this procedure. Second, there must be a *verification* procedure by which any user can check a record and its accompanying certificate to make sure it correctly answers (1) who and what? or (2) when and what? about the record in question.

The "certificate" returned by the certification procedure of a digital signature system is usually called a *signature*; it is a signature for a particular signer (specifying whom) and for a particular record (specifying what). In order to be able to "sign" documents, a user registers with the system by using special software to compute a pair of numbers called keys — a *public key* and a corresponding *private key.* The private key should only be available to the user to whom it belongs, and is used (by the certification or "signing" procedure) in order to sign documents; it is by employing the user's private key that the signature and the record are tied to that particular user. The public key may be available to many users of the system, and is used by the verification procedure. That is, the verification procedure takes a particular record, a particular user's public key, and a putative signature for that record and that user, and uses this information to check whether the would-be signature was correctly computed using that record and the corresponding private key.

Special computational methods are employed for signing documents and for verifying documents and signatures; when these methods are carefully implemented, they have the remarkable property that the knowledge of a user's public key does not enable an attacker or hacker to figure out the user's corresponding private key. Of course, if, either through carelessness or deliberate intent, someone else — a hacker, for example — gains access to the user's private key, then this person will be able to "forge" the legitimate user's signatures on documents of the hacker's choice. At that point, even the value of legitimately signed records can be called into question.

The "certificate" returned by the certification procedure of a digital timestamping system is a certificate for a particular record (specifying what) at a particular time (specifying when). The procedure works by mathematically linking the bits of the record to a "summary number" that is widely witnessed by and widely available to members of the public — including, of course, users of the system. The computational methods employed ensure that only the record in question can be linked, according to the "instructions" contained in its timestamp certificate, to this widely witnessed summary number; this is how the particular record is tied to a particular moment in time. The verification procedure takes a particular record and a putative timestamp certificate for that record and a particular time, and uses this information to validate whether that record was indeed certified at the time claimed by checking it against the widely available summary number for that moment.

Two features of a digital timestamping system are particularly helpful in enhancing the integrity of a digital signature system. First, a timestamping system cannot be compromised by the disclosure of a key. This is because digital timestamping systems do not rely on keys, or any other secret information, for that matter. Second, following the technique introduced in [BHS93], digital timestamp certificates can be *renewed* so as to remain valid indefinitely.

With these features in mind, consider the following situations.

It sometimes happens that the connection between a person and his or her public signature key must be revoked — for example, if the user's secure access to the private key is accidentally compromised; or when the key belongs to a job or role in an organization that the person no longer holds. Therefore the person-key connection must have time limits, and the signature verification procedure should check that the record was signed at a time when the signer's public key was indeed in effect. And thus when a user signs a record that may be checked some time later — perhaps after the user's key is no longer in effect — the combination of the record *and* its signature should be certified with a secure digital timestamping service.

There is another situation in which a user's public key may be revoked. Consider the case of the signer of a particularly important document who later wishes to repudiate his signature. By dishonestly reporting the compromise of his private key, so that all his signatures are called into question, the user is able to disavow the signature he regrets. However, if the document in question was digitally timestamped together with its signature (and key-revocation reports are timestamped as well), then the signature cannot be disavowed in this way. This is the recommended procedure, therefore, in order to preserve the *non-repudiability* desired of digital signatures for important documents.

The statement that private keys cannot be derived from public keys is an over-simplification of a more complicated situation. In fact, this claim depends on the computational difficulty of certain mathematical problems. As the state of the art advances — both the current state of algorithmic knowledge, as well as the computational speed and memory available in currently available computers — the maintainers of a digital signature system will have to make sure that signers use longer and longer keys. But what is to become of documents that were signed using key lengths that are no longer considered secure? If the signed document is digitally timestamped, then its integrity can be maintained even after a particular key length is no longer considered secure.

Of course, digital timestamp certificates also depend for their security on the difficulty of certain computational tasks concerned with one-way hash functions (see Question 94). (All practical digital signature systems depend on these functions as well.) The maintainers of a secure digital timestamping service will have to remain abreast of the state of the art in building and in attacking one-way hash functions. Over time, they will need to upgrade their implementation of these functions, as part of the process of renewal [BHS93]. This will allow timestamp certificates to remain valid indefinitely.

NEW TECHNOLOGIES

109. What is Quantum Computing?

Quantum computing [Ben82][Fey82][Fey86][Deu92] is a new field in computer science that has been developed with our increased understanding of quantum mechanics. It holds the key to computers that are exponentially faster than conventional computers (for certain problems). A quantum computer is based on the idea of a quantum bit or *qubit*. In classical computers, the bit has a discrete range and can represent either a *zero* or a *one*. A qubit can be in a linear *superposition* of the two states. A quantum *register* consists of *n* qubits. Because of superposition, a phenomenon known as *quantum parallelism* allows exponentially many such computations to take place simultaneously, thus vastly increasing the speed of computation.

Quantum interference, the analog of Young's double-slit experiment which demonstrated constructive and destructive interference phenomena of light, is one of the most significant characteristics of quantum computing. Quantum interference improves the probability of obtaining a desired result by constructive interference and diminishes the probability of obtaining an erroneous result by destructive interference. Thus, among the exponentially many computations, the correct answer can theoretically be identified with appropriate quantum "algorithms."

It has been proven [Sho94] that a quantum computer will be able to factor (see Question 45) and compute discrete logarithms (see Question 52) in polynomial time. Unfortunately, the development of a practical quantum computer still seems far away because of a phenomenon called *quantum decoherence*, which is due to the influence of the outside environment on the quantum computer. Brassard has written a number of helpful texts in this field [Bra95a][Bra95b][Bra95c].

Quantum cryptography (see Question 110) is quite different from, and currently more viable than, quantum computing.

110. What is Quantum Cryptography?

Quantum cryptography [BBB92][Bra93] is a method for secure key exchange over an insecure channel based on the nature of *photons*. Photons have a polarization, which can be measured in any basis, where a basis consists of two directions orthogonal to each other, as shown in Figure 12. If a photon's polarization is read in the same basis twice,

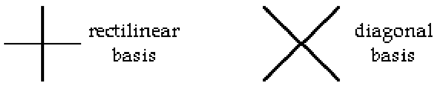

Figure 12. Bases

the polarization will be read correctly and will remain unchanged. If it is read in two different bases, a random answer will be obtained in the second basis, and the polarization in the initial basis will be changed randomly, as shown in Figure 13.

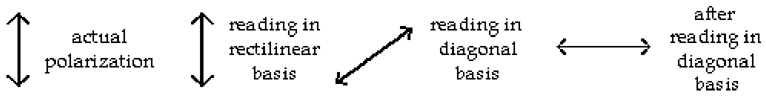
actual polarization
reading in rectilinear basis
reading in diagonal basis
after reading in diagonal basis

Figure 13. Polarization readings

The following protocol can be used by Alice and Bob to exchange secret keys:

- Alice sends Bob a stream of photons, each with a random polarization, in a random basis. She records the polarizations.
- Bob measures each photon in a randomly chosen basis and records the results.
- Bob announces, over an authenticated but not necessarily private channel (e.g., by telephone), which basis he used for each photon.
- Alice tells him which choices of bases are correct.
- The shared secret key consists of the polarization readings in the correctly chosen bases.

Quantum cryptography has a special defense against eavesdropping: If an enemy measures the photons during transmission, he will use the wrong basis half the time, and thus will change some of the polarizations. That will result in Alice and Bob having different values for their secret keys. As a check, they can exchange some random bits of their key using an authenticated channel. They will therefore detect the presence of eavesdropping, and can start the protocol over.

There has been experimental work in developing such systems by IBM and British Telecom. For information on quantum computing, which is quite different from quantum cryptography, see Question 109.

111. What is DNA Computing?

DNA computing, also known as *molecular computing*, is a new approach to massively parallel computation based on ground-breaking work by Adleman. He used DNA to solve a seven-node Hamiltonian path problem, a special case of an NP-complete problem that attempts to visit every node in a graph exactly once. (This special case is trivial to solve with a conventional computer, or even by hand, but illustrates the potential of DNA computing.)

A DNA computer is basically a collection of specially selected DNA strands whose combinations will result in the solution to some problem. Technology is currently available both to select the initial strands and to filter the final solution. The promise of DNA computing is massive parallelism; with a given setup and enough DNA, one can potentially solve huge problems by parallel search. This can be much faster than a conventional computer, for which massive parallelism would require large amounts of hardware, not simply more DNA.

Research on DNA computing is ongoing; Lipton [Lip94] and Adleman [Adl95] have extended on Adleman's original work with more efficient designs of possible DNA computers.

The impact of DNA computing on cryptography remains to be determined. Beaver [Bea95] has shown that to factor a 1000-bit number following Adleman's original approach of solving an NP-complete problem, the required amount of solution would be 10^{200000} liters, which is not practical. However, Adleman has observed that a DNA computer sufficient to search for 2^{56} DES keys would occupy only a small set of test tubes [Adl96]. In any case, DNA computing is just classical computing, albeit highly parallelized, so with a large enough key, one should be able to thwart any DNA computer that can be built. With quantum computing (see Question 109), on the other hand, factoring can theoretically be done in (quantum) polynomial time, so quantum computing seems to be much more of a concern than DNA computing.

Part Two

SECURITY PROTOCOLS AND SERVICES

KEY MANAGEMENT

General

112. How Does One Find Random Numbers for Keys?

One needs a good source of random numbers to generate keys, both for secret-key algorithms and public-key algorithms. Random numbers obtained from a physical process are in principle the best. One could use a hardware device, such as a noisy diode; some are sold commercially on computer add-in boards for this purpose. Another idea is to use physical movements of the computer user, such as keystroke timings measured in microseconds. However, it was recently proved that techniques that used the spinning of disks to generate random data are not truly random, as the movement of the disk platter is not truly random. A negligible-cost alternative is available. Davis *et al.* designed a random number generator based on the variation of a disk drive motor's speed [DIP94]. This variation is caused by air turbulence, which has been shown to be unpredicatable. By whichever method they are generated, the random numbers may still contain some correlations, thus preventing sufficient statistical randomness. Therefore, it is best to run them through a good hash function (see Question 94) before actually using them [ECS94].

Another approach is to use a pseudorandom number generator fed by a random seed. Since these are deterministic algorithms, it is important to find one that is cryptographically secure and also to use a truly random seed; the generator effectively acts as an "expander" from the seed to a larger amount of pseudorandom data. The seed must be sufficiently variable to deter attacks based on trying all possible seeds.

It is not sufficient for a pseudorandom number generator just to pass a variety of statistical tests, as described in Knuth [Knu81] and elsewhere, because the output of such generators may still be predictable. Rather, it must be computationally infeasible for an attacker to determine any bit of the output sequence, even if all the others are known, with probability better than 1/2. Blum and Micali's generator based on the discrete logarithm problem [BM84] satisfies this stronger definition, assuming that computing discrete logarithm is difficult(see Question 52). Other generators, perhaps based on DES (see Question 64) or a hash function (see Question 94), can also be considered to satisfy this definition, under reasonable assumptions.

A summary of different methods for generating random numbers in software can be found in [Mat96].

Note that one does not need random numbers to determine the public and private exponents in RSA (see Question 8), after generating the primes and hence the modulus. One can simply choose an arbitrary value for the public exponent, which then determines the private exponent.

113. What is the Life Cycle of a Key?

Keys have limited lifetimes for a number of reasons. The most important reason is protection against cryptanalysis. Each time the key is used, it generates a number of ciphertexts. Using a key repetitively allows an attacker to build up a store of ciphertexts (and possibly plaintexts) which may prove sufficient for a successful cryptanalysis of the key value. If you suspect that an attacker may have obtained your key, the your key is considered *compromised*.

Research in cryptanalysis can lead to possible attacks against either the key or the algorithm. For example, RSA key lengths are increased every few years to ensure that the improved factoring algorithms do not compromise the security of messages encrypted with RSA.

Another reason for limiting the lifetime of a key is to minimize the damage from a compromised key. It is unlikely that a user will discover that his key has been compromised by an attacker if the attacker remains “passive.” Relatively frequent key changes will limit any potential damage from compromised keys. Ford [For94] describes the life cycle of a key as follows:

- Key generation and possibly registration for a public key.
- Key distribution.
- Key activation/deactivation.
- Key replacement or key update.
- Key revocation.
- Key termination, involving destruction and possibly archival.

Public-Key Issues

114. What Key Management Issues are Involved in Public-Key Cryptography?

Secure methods of key management are extremely important. In practice, most attacks on public-key systems will probably be aimed at the key management level, rather than at the cryptographic algorithm itself. The key management issues mentioned here are discussed in detail in later questions.

Users must be able to securely obtain a key pair suited to their efficiency and security needs. There must be a way to look up other people's public keys and to publicize one's own key. Users must have confidence in the legitimacy of others' public keys; otherwise, an intruder can either change public keys listed in a directory, or impersonate another user. Certificates are used for this purpose (see Question 123). Certificates must be unforgeable, obtainable in a secure manner, and processed in such a way that an intruder cannot misuse them. The issuance of certificates must proceed in a secure way, impervious to attack.

If someone's private key is lost or compromised, others must be made aware of this, so that they will no longer encrypt messages under the invalid public key nor accept messages signed with the invalid private key. Users must be able to store their private keys securely, so that no intruder can find them, yet the keys must be readily accessible for legitimate use. Keys need to be valid only until a specified expiration date. The expiration date must be chosen properly and publicized in an authenticated channel. Some documents need to have verifiable signatures beyond the time when the key used to sign them has expired.

115. Who Needs a Key Pair?

Anyone who wishes to sign messages or to receive encrypted messages must have a key pair. People may have more than one key pair. For example, someone might have a key pair affiliated with his or her work and a separate key pair for personal use. Other entities may also have key pairs, including electronic entities such as modems, workstations and printers; as well as organizational entities such as a corporate department, a hotel registration desk, or a university registrar's office.

Corporations may require more than one key pair for communication. They may use one or more key pairs for encryption (with the keys stored under key escrow to safeguard the key in event of loss) and use a single key pair for authentication purposes. The lengths of the encryption and authentication key pairs may be varied according to the desired security.

116. How Does One Get a Key Pair?

A user can generate his or her own key pair, or, depending on local policy, a security officer may generate key pairs for all users. There are tradeoffs between the two approaches; in the former, the user needs some way to trust his or her copy of the key generation software, and in the latter, the user must trust the security officer, and the private key must be transferred securely to the user. Typically, each node on a network should be capable of local key generation. Secret-key authentication systems, such as Kerberos (see Question 144), often do not allow local key generation, but instead use a central server to generate keys.

Once generated, a user must register his or her public key with some central administration, called a Certifying Authority. The Certifying Authority returns to the user a certificate attesting to the validity of the user's public key along with other information (see Question 123 through Question 125). If a security officer generates the key pair, then the security officer can request the certificate for the user. Most users should not obtain more than one certificate for the same key, in order to simplify various bookkeeping tasks associated with the key.

117. Should a Public Key or Private Key be Shared Among Users?

Users who share a key can impersonate one another (i.e., sign messages as one another and decrypt messages intended for one another), so in general keys should not be shared among users. However, some parts of a key may be shared, depending on the algorithm.

In RSA, while each person should have a unique modulus and private exponent (i.e., a unique private key), the public exponent can be common to a group of users without security being compromised. Some public exponents in common use today are *3* and $2^{16}+1$; because these numbers are small, the public-key operations (encryption and signature verification) are fast relative to the private-key operations (decryption and signing). If one public exponent becomes standard, software and hardware can be optimized for that value. However, the modulus should not be shared.

In public-key systems based on discrete logarithms, such as Diffie-Hellman, DSA, and ElGamal (see Question 24, Question 26, and Question 29), a group of people can share a prime, which can lead to simpler implementations. It is worth noting, however, that this would make breaking a key more attractive to an attacker because it is possible to break every key with a given prime with only slightly more effort than it takes to break a single key. To an attacker, therefore, the average cost to break a key is much lower with a common prime than if every key has a distinct prime. Thus, if a common prime is chosen in a discrete-logarithm system, it should be large.

118. What Happens When a Key Expires?

In order to guard against a long-term cryptanalytic attack, every key must have an expiration date after which it is no longer valid (see Question 113). The time to expiration must therefore be much shorter than the expected time for cryptanalysis, or equivalently, the key length must be long enough to make the chances of cryptanalysis before key expiration extremely small. The validity period for a key pair may also depend on the circumstances in which the key is used, although there will also be a standard validity period. The validity period, together with the value of the key and the estimated strength of an expected attacker, determines the appropriate key size. In a certificate (see Question 123), the expiration date of a key is typically the same as the expiration date of the certificate, though it need not be.

A signature verification program should check for expiration and should not accept a message signed with an expired key. This means that when one's own key expires, everything signed with it will no longer be considered valid. Of course, there will be cases where it is important that a signed document be considered valid for a much longer period of time; Question 108 discusses ways to achieve this.

After expiration, the user should typically choose a new key, which should be longer than the old key to reflect both the performance increase of computer hardware and any recent improvements in factoring algorithms. Recommended key length schedules are published regularly by RSA Laboratories. However, if a key is sufficiently long and has not been compromised, the user can continue to use the same key. In this case, the certifying authority would issue a new certificate for the same key, and all new signatures would point to the new certificate instead of the old. However, the fact that computer hardware continues to improve argues for replacing expired keys with new, longer keys every few years. Key replacement enables one to take advantage of the hardware improvements to increase the security of the cryptosystem. Faster hardware has the effect of increasing security, perhaps vastly, but only if key lengths are increased regularly (see Question 47).

119. What Happens if I Lose my Private Key?

If your private key is lost or destroyed, but not compromised, you can no longer sign or decrypt messages, but anything previously signed with the lost key is still valid. This can happen, for example, if you lose the smart card used to store your key, or if the disk on which the key is stored is damaged. You need to obtain a new key right away, to minimize the number of messages people send you that are encrypted under your old key, messages which you can no longer read.

120. What Happens if my Private Key is Compromised?

If your private key is compromised, that is, if you suspect an attacker may have obtained your private key, then you must assume that some enemy can read encrypted messages sent to you and forge your signature on documents. The seriousness of these consequences underscores the importance of protecting your private key with extremely strong mechanisms (see Question 121).

You must immediately notify your certifying authority and have your old key placed on a certificate revocation list (see Question 129); this will inform people that the key has been revoked. Then generate a new key and obtain the proper certificates for it. You may wish to use the new key to re-sign documents that you had signed with the compromised key; documents that had been timestamped as well as signed might still be valid. You should also change the way you store your private key, to prevent compromise of the new key.

121. How Should I Store my Private Key?

Private keys must be stored securely, since forgery and loss of privacy could result from compromise (see Question 120). The measures taken to protect the private key must be at least equal to the security of the messages encrypted with the key. The private key should never be stored anywhere in plaintext form. The simplest storage mechanism is to encrypt the private key under a password and store the result on a disk. However, since passwords are sometimes easily guessed, such a password should be chosen very carefully.

Storing the encrypted key on a disk that is not accessible through a computer network, such as a floppy disk or a local hard disk, will make some attacks more difficult. It might be best to store the key in a computer that is not accessible to other users or on removable media that the user can remove and take with her when she has finished using a particular computer. Private keys may be stored on portable hardware, such as a smart card. Users with extremely high security needs, such as certifying authorities, should use special hardware devices to protect their keys (see Question 126).

122. How Do I Find Someone Else's Public Key?

Suppose Alice wants to find Bob's public key. There are several possible ways. She could call him up and ask him to send her his public key via e-mail; she could request it via e-mail as well. Certifying authorities may provide directory services; if Bob works for company Z, she could look in the directory kept by Z's certifying authority. Directories must be secure against tampering, so that users can be confident that a public key listed in the directory actually belongs to the person listed. Otherwise, they might send private encrypted information to the wrong person.

Eventually, full-fledged directories will arise, serving as on-line white or yellow pages. If they are compliant with ITU-T X.509 standards (see Question 165), the directories will contain certificates as well as public keys; the presence of certificates will lower the directories' security needs.

Certificates

123. What are Certificates?

Certificates are digital documents attesting to the binding of a public key to an individual or other entity. They allow verification of the claim that a given public key does in fact belong to a given individual. Certificates help prevent someone from using a phony key to impersonate someone else.

In their simplest form, certificates contain a public key and a name. As commonly used, a certificate also contains an expiration date, the name of the certifying authority that issued the certificate, a serial number, and perhaps other information. Most importantly, it contains the digital signature of the certificate issuer. The most widely accepted format for certificates is defined by the ITU-T X.509 international standard (see Question 165); thus, certificates can be read or written by any application complying with X.509. A detailed discussion of certificate formats can be found in [Ken93].

124. How are Certificates Used?

Certificates are typically used to generate confidence in the legitimacy of a public key. Someone verifying a signature can also verify the signer's certificate, to ensure that no forgery or false representation has occurred. These steps can be performed with greater or lesser rigor depending on the context.

The most secure use of authentication involves enclosing one or more certificates with every signed message. The receiver of the message would verify the certificate using the certifying authority's public key and, now confident of the public key of the sender, verify the message's signature. There may be two or more certificates enclosed with the message, forming a hierarchical chain, wherein one certificate testifies to the authenticity of the previous certificate. At the end of a certificate hierarchy is a top-level certifying authority, which is trusted without a certificate from any other certifying authority. The public key of the top-level certifying authority must be independently known, for example, by being widely published.

The more familiar the sender is to the receiver of the message, the less need there is to enclose, and to verify, certificates. If Alice sends messages to Bob every day, Alice can enclose a certificate chain, which Bob verifies on the first day. Bob thereafter stores Alice's public key and no more certificates or certificate verifications are necessary. A sender whose company is known to the receiver may need to enclose only one certificate (issued by the company), whereas a sender whose company is unknown to the receiver may need to enclose two certificates. A good rule of thumb is to enclose just enough of a certificate chain so that the issuer of the highest level certificate in the chain is well-known to the receiver. If there are multiple recipients, then enough certificates should be included to cover what each recipient might need.

In the Public-Key Cryptography Standards (PKCS) (see Question 166), as well as Privacy-Enhanced Mail (PEM) (see Question 130), every signature points to a certificate that validates the public key of the signer. Specifically, each signature contains the name of the issuer of the certificate and the serial number of the certificate. Thus, even if no certificates are enclosed with a message, a verifier can still use the certificate chain to check the status of the public key.

125. Who Issues Certificates and How?

Certificates are issued by a certifying authority (CA), which can be any trusted central administration willing to vouch for the identities of those to whom it issues certificates and their association with a given key. A company may issue certificates to its employees, a university to its students, a town to its citizens. In order to prevent forged certificates, the CA's public key must be trustworthy. A CA must either publicize its public key or provide a certificate from a higher-level CA attesting to the validity of its public key. The latter solution gives rise to hierarchies of CAs (see Figure 14 for an example).

Certificate issuance proceeds as follows. Alice generates her own key pair and sends the public key to an appropriate CA with some proof of her identification. The CA checks the identification and takes any other steps necessary to assure itself that the request really did come from Alice, and then sends her a certificate attesting to the binding between Alice and her public key, along with a hierarchy of certificates verifying the CA's public key. Alice can present this certificate chain whenever desired in order to demonstrate the legitimacy of her public key.

Since the CA must check for proper identification, organizations find it convenient to act as a CA for its own members and employees. There are also CAs that issue certificates to unaffiliated individuals.

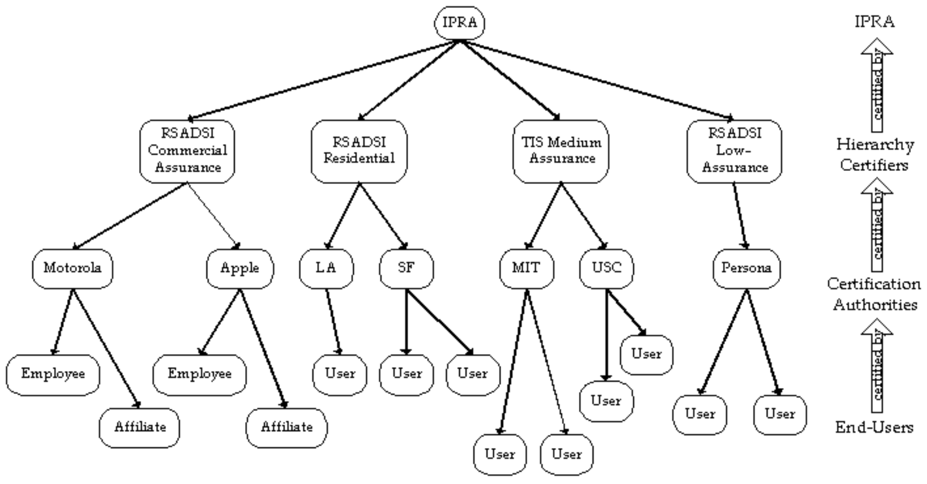

Figure 14. Example of a Certification Hierarchy.

Different CAs may issue certificates with varying levels of identification requirements. One CA may insist on seeing a driver's license, another may want the certificate request form to be notarized, yet another may want fingerprints of anyone requesting a certificate. Each CA should publish its own identification requirements and standards, so that verifiers can attach the appropriate level of confidence in the certified name-key bindings. CA's with lower levels of identification requirements produce certificates with lower "assurance." CA's can thus be considered to be of high, medium, and low assurance. One type of CA is the *persona* CA. This type of CA creates certificates that bind only e-mail addresses and their corresponding public-keys. It is designed for users who wish to remain anonymous yet want to be able to participate in secure electronic services.

An example of a certificate-issuing protocol is found in Apple Computer's System 7.5 for the Macintosh. System 7.5 users can generate a key pair and then request and receive a certificate for the public key; the certificate request must be notarized.

VeriSign, Inc. provides a complete line of certificate-related products and services; see `<http://www.verisign.com/>` for more information.

126. How Do Certifying Authorities Store their Private Keys?

It is extremely important that the private keys of certifying authorities (see Question 125) are stored securely because compromise would enable undetectable forgeries. One way to achieve the desired security is to store the key in a tamper-resistant device. The device should preferably destroy its contents if ever opened, and be shielded against attacks using electromagnetic radiation. Not even employees of the certifying authority should have access to the private key itself, but only the ability to use the private key in the process of issuing certificates.

There are many possible designs for controlling the use of a certifying authority's private key. BBN's SafeKeyper, for instance, is activated by a set of data keys, which are physical keys capable of storing digital information. The data keys use secret sharing technology so that several people must use their data keys to activate the SafeKeyper. This prevents a disgruntled CA employee from producing phony certificates.

Note that if the certificate-signing device is destroyed, say in a fire, no security is compromised. Certificates signed by the device are still valid, as long as the verifier uses the correct public key. Moreover, some devices are manufactured so that a lost private key can be restored into a new device (see Question 128).

127. How Are Certifying Authorities Susceptible to Attack?

One can think of many attacks aimed at certifying authorities (see Question 125), all of which can be defended against.

For instance, an attacker may attempt to discover the private key of a certifying authority by reverse engineering the device in which it is stored. For this reason, a certifying authority must take extreme precautions to prevent illegitimate access to its private key (see Question 126 for discussion).

The certifying authority's key pair might be the target of an extensive cryptanalytic attack. For this reason, CAs should use long keys, and should also change keys regularly. Top-level certifying authorities need especially long keys, as it may not be practical for them to change keys frequently because the public key may be written into software used by a large number of verifiers.

What if an attacker breaks a CA's key, but the CA is no longer using it? Though the key has long since expired, the attacker, say Alice, can now forge a certificate dated 15 years ago attesting to a phony public key of some other person, say Bob; Alice can then forge a document with a signature of Bob dated 15 years ago, perhaps a will leaving everything to Alice. The underlying issue raised by this attack is how to authenticate a signed document dated many years ago; this issue is discussed in Question 108.

There are other attacks to consider that do not involve the compromise of a CA's private key. For instance, suppose Bob wishes to impersonate Alice. If Bob can convincingly sign messages as Alice, he can send a message to Alice's bank saying "I wish to withdraw $10,000 from my account. Please send me the money." To carry out this attack, Bob generates a key pair and sends the public key to a certifying authority saying "I'm Alice. Here is my public key. Please send me a certificate." If the CA is fooled and sends him such a certificate, he can then fool the bank, and his attack will succeed. In order to prevent such an attack, the CA must verify that a certificate request did indeed come from its purported author, i.e., it must require sufficient evidence that it is actually Alice who is requesting the certificate. The CA may, for example, require Alice to appear in person and show a birth certificate. Some CAs may require very little identification, but the bank should not honor messages authenticated with such low-assurance certificates. Every CA must publicly state its identification requirements and policies; others can then attach an appropriate level of confidence to the certificates.

In another attack, Bob bribes someone who works for the CA to issue to him a certificate in the name of Alice. Now Bob can send messages signed in Alice's name and anyone receiving such a message will believe it is authentic because a full and verifiable certificate chain will accompany the message. This attack can be hindered by requiring the cooperation of two (or more) employees to generate a certificate; the attacker now has to bribe two or more employees rather than one (see Question 126).

Unfortunately, there may be other ways to generate a forged certificate by bribing only one employee. If each certificate request is checked by only one employee, that one employee can be bribed and slip a false request into a stack of real certificate requests. Note that a corrupt employee cannot reveal the certifying authority's private key, as long as it is properly stored.

A CA should also be certain that a user possesses the private key corresponding to the public key that is certified; otherwise certain attacks become possible where the user attaches a certificate to a message signed by someone else (see [Kal93b]). (See also [MQV95] for discussion of this issue in the context of key agreement protocols.)

Note that these attacks on certifying authorities do not directly threaten the privacy of messages between users, as might result from an attack on a secret-key distribution center in a secret-key cryptosystem.

128. What if a Certifying Authority's Key is Lost or Compromised?

If the certifying authority's key is lost or destroyed but not compromised, certificates signed with the old key are still valid, as long as the verifier knows to use the old public key to verify the certificate.

In some designs for certificate-signing devices, encrypted backup copies of the CA's private key are kept, so a CA that loses its key can then restore it by loading the encrypted backup into the device. If the device itself is destroyed, the manufacturer may be able to supply another one with the same internal information, thus allowing recovery of the key.

A compromised CA key is a much more dangerous situation. An attacker who discovers a certifying authority's private key can issue phony certificates in the name of the certifying authority, which would enable undetectable forgeries; for this reason, all precautions must be taken to prevent compromise, including those outlined in Question 126 and Question 127.

If a compromise does occur, the CA must immediately cease issuing certificates under its old key and change to a new key. If it is suspected that some phony certificates were issued, all certificates should be recalled and then reissued with a new CA key. These measures could be relaxed somewhat if the certificates were registered with a digital timestamping service (see Question 108). Note that compromise of a CA key does not invalidate users' keys, but only the certificates that authenticate them. Compromise of a top-level CA's private key should be considered catastrophic, since the public key may be built into applications that verify certificates.

129. What are Certificate Revocation Lists (CRLs)?

A *certificate revocation list* (CRL) is a list of certificates that have been revoked before their scheduled expiration date. There are several reasons why a certificate might need to be revoked and placed on a CRL. For instance, the key specified in the certificate might have been compromised, or, the user specified in the certificate may no longer have authority to use the key. For example, suppose the user name associated with a key is “Alice Avery, Vice President, Argo Corp.” If Alice were fired, her company would not want her to be able to sign messages with that key, and therefore, the company would place the certificate on a CRL.

When verifying a signature, one can check the relevant CRL to make sure the signer’s certificate has not been revoked. Whether it is worth the time to perform this check depends on the importance of the signed document.

CRLs are maintained by CAs and provide information about revoked certificates that were issued by the CA. CRLs only list current certificates, since expired certificates should not be accepted in any case; when a revoked certificate is past its original expiration date, it is removed from the CRL. Although CRLs are maintained in a distributed manner, there may be central repositories for CRLs, that is, network sites containing the latest CRLs from many organizations. An institution like a bank might want an in-house CRL repository to make CRL searches on every transaction feasible.

The original CRL proposals often required a list, per issuer, of all revoked certificates; new certificate revocation methods (e.g., in X.509 version 3, see Question 165) are more flexible.

SECURITY ON THE INTERNET

Secure Electronic Mail

130. What is PEM?

PEM is the draft Internet Privacy-Enhanced Mail standard, designed, proposed, but not yet officially adopted, by the Internet Activities Board to provide secure electronic mail over the Internet. Designed to work with RFC 822 e-mail formats, PEM includes encryption, authentication, and key management, and allows use of both public-key and secret-key cryptosystems. Multiple cryptographic tools are supported; for each mail message, the specific encryption algorithm, digital signature algorithm, hash function, and so on are specified in the header. PEM explicitly supports only a few cryptographic algorithms; others may be added later. DES (see Question 64) in CBC mode is currently the only message encryption algorithm supported, and both RSA (see Question 8) and DES are supported for key management. Public-key management in PEM is based on X.509 certificates (see Question 165).

The details of PEM can be found in Internet RFCs 1421 through 1424. PEM has been on the draft track for more than two years and it seems that the standard may be superseded by S/MIME and PEM-MIME. Trusted Information Systems has developed a free non-commercial implementation of PEM, and other implementations have been developed such as RIPEM (see Question 178).

131. What is S/MIME?

S/MIME (Secure/ Multipurpose Internet Mail Extensions) is a protocol that adds digital signatures and encryption to Internet MIME (Multipurpose Internet Mail Extensions) messages described in RFC 1521. MIME is the official proposed standard format for extended Internet electronic mail. Internet e-mail messages consist of two parts, the header and the body. The header forms a collection of field/value pairs structured to provide information essential for the transmission of the message. The structure of the headers can be found in RFC 822. The body is normally unstructured unless the e-mail is in MIME format. MIME defines how the body of an e-mail message is structured. The MIME format permits e-mail to include enhanced text, graphics, audio, and more in a standardized manner via MIME-compliant mail systems. However, MIME itself does not provide any security services. The purpose of S/MIME is to define such services, following the syntax given in PKCS #7 (see Question 166) for digital signatures and encryption. The MIME body part carries a PKCS #7 message, which itself is the result of cryptographic processing on other MIME body parts.

S/MIME has been recently endorsed by a number of leading networking and messaging vendors, including ConnectSoft, Frontier, FTP Software, Qualcomm, Microsoft, Lotus, Wollongong, Banyan, NCD, SecureWare, VeriSign, Netscape, and Novell. For more information on S/MIME, check `<http://www.rsa.com>`.

132. What is PEM-MIME, or What is MOSS?

PEM-MIME, also known as MIME Object Security Standard or MOSS, is a proposed Internet Draft [CFG95] that is designed to be a successor to PEM (see Question 130). It proposes adding PEM- based security services to MIME messages in much the same manner as S/MIME (see Question 131). Due to the nature of MIME, it is possible to apply different security services to each part of the body. For example, the MIME body may contain two copies of a message, with one copy digitally signed and the other copy not modified in any way. This will allow a recipient to read the message even if the recipient does not have a MIME-compliant mail reader. If the recipient has a privacy-enhanced MIME compliant mail reader, the recipient will be able to verify the digital signature as well. Another possibility would be to encrypt different blocks of the message body using different keys and algorithms, or to sign some blocks and not others need not b.

The proposed standard has come under criticism because the requirements for PEM-MIME mailers are extremely flexible. This flexibility can result in two different PEM-MIME-compliant mailers where one mailer can produce PEM-MIME messages that the other PEM-MIME mailer is unable to read. This flexibility is in part a reaction to the rigidity in the structure of PEM, which was not very popular among users. S/MIME treads a somewhat middle ground between the rigid standards of PEM and the loosely defined requirements of PEM-MIME.

General Security Protocols

133. What is S-HTTP?

S-HTTP (Secure Hypertext Transfer Protocol) is an extension to HTTP (Hypertext Transfer Protocol) that provides security services [RS95]. It was originally developed by Enterprise Integration Technologies, and further development continues at Terisa Systems. HTTP is the protocol that forms the basis of the World Wide Web, allowing the exchange of multimedia documents on the Web. S-HTTP is designed to provide confidentiality, authenticity, integrity, and non-repudiability while supporting multiple key management mechanisms and cryptographic algorithms via option negotiation between the parties involved in each transaction.

S-HTTP can use any of four methods to exchange data-encrypting keys. The possible methods are RSA (see Question 8), out-band, in-band, and Kerberos (see Question 144). If RSA is used, data-encrypting keys are exchanged by the RSA public-key cryptosystem. Out-band refers to an external key agreement, while in-band refers to a key transported in a S-HTTP-protected message in another session. In the Kerberos method, the key is obtained from a Kerberos server. Cryptographic algorithms supported by S-HTTP include DES (see Question 64), two-key and three-key triple-DES (see Question 72), DESX (see Question 74), IDEA (see Question 77), RC2 (see Question 75), and CDMF [JML93].

More information about S-HTTP can be obtained from Terisa Systems at `<http://www.terisa.com/>`.

134. What is SSL?

The SSL (Secure Socket Layer) Handshake Protocol [Hic95] was developed by Netscape Communications Corporation to provide security and privacy over the Internet. The protocol supports server and client authentication. The SSL protocol is application independent, allowing protocols like HTTP (see Question 133), FTP (File Transfer Protocol), and Telnet to be layered on top of it transparently. The SSL protocol is able to negotiate encryption keys as well as authenticate the server before data is exchanged by the higher-level application. The SSL protocol maintains the security and integrity of the transmission channel by using encryption, authentication and message authentication codes.

The SSL Handshake Protocol consists of two phases, server authentication and client authentication, with the second phase being optional. In the first phase, the server, in response to a client's request, sends its certificate and its cipher preferences. The client then generates a master key, which it encrypts with the server's public key, and transmits the encrypted master key to the server. The server recovers the master key and authenticates itself to the client by returning a message encrypted with the master key. Subsequent data is encrypted with keys derived from this master key. In the optional second phase, the server sends a challenge to the client. The client authenticates itself to the server by returning the client's digital signature on the challenge, as well as its public-key certificate.

A variety of cryptographic algorithms are supported by SSL. During the "handshaking" process, the RSA public-key cryptosystem (see Question 8) is used. After the exchange of keys, a number of ciphers are used. These include RC2 (see Question 75), RC4 (see Question 87), IDEA (see Question 77), DES (see Question 64), and triple-DES (see Question 72). The MD5 message-digest algorithm (see Question 99) is also used. The public-key certificates follow the X.509 syntax (see Question 165).

The SSL protocol has been submitted as an Internet Draft. Questions can be addressed to `<standards@netscape.com>`.

135. What is the Difference Between SSL and S-HTTP?

The main difference between the protocols is the layer at which they operate. SSL (see Question 134) operates at the transport layer and mimics the “socket library,” while S-HTTP (see Question 133) operates at the application layer. Encryption of the transport layer allows SSL to be application-independent, while S-HTTP is limited to the specific software implementing it. The protocols adopt different philosophies towards encryption as well, with SSL encrypting the entire communications channel and S-HTTP encrypting each message independently. S-HTTP allows a user to produce digital signatures on any messages (not just specific messages during an authentication protocol), a feature SSL lacks. Terisa Systems is developing toolkits to support both protocols.

136. What is PCT?

PCT stands for Private Communication Technology, a protocol developed by Microsoft and Visa International for secure communication on the Internet [BLS95]. It is a counterpart to Netscape's SSL protocol (see Question 134). Like SSL, PCT is intended for Internet standardization.

The protocol is quite similar to SSL in many respects, and in fact the message formats are similar enough so that a server can interact with clients supporting SSL as well as client supporting PCT. According to the specification, PCT "corrects or improves on several weaknesses of SSL." The following are the main differences:

- PCT involves fewer messages between the client and the server than SSL, and the messages themselves are shorter.
- PCT has more choices in the negotiation of algorithm and data formats than SSL, and the negotiation has additional cryptographic protection so that the client and server can verify that their choices have not been modified.
- Message authentication and encryption in PCT uses different keys. In SSL, both involve the same keys. This means in particular that in PCT, authentication can involve longer keys than encryption (encryption key length may be limited by export restrictions), and can thus be more secure.
- In the PCT authentication protocol, the client's response depends on the negotiated encryption algorithm, where as in SSL it is independent of the algorithm. This provides a kind of "firewall" so that an opponent who recovers the encryption key in a session with one choice of algorithm (e.g., a weak algorithm) cannot subsequently compromise a session with another choice of algorithm (e.g., a strong one). SSL does not have this "firewall."

For key establishment, PCT supports RSA (see Question 8), Diffie-Hellman (see Question 24), and Fortezza (see Question 156); encryption algorithms include DES (see Question 64), triple-DES (see Question 72), RC2 (see Question 75), and RC4 (see Question 87). Both DSA (see Question 26) and RSA signatures are supported.

137. What is S/WAN?

The S/WAN (Secure Wide Area Network, pronounced “swan”) initiative designates specifications for implementing IPSec, the security architecture for the Internet Protocol [Atk95a][Atk95b][Atk95c][KMS95][MS95], to ensure interoperability among firewall and TCP/IP products. S/WAN’s goal is to use IPSec to allow companies to mix-and-match the best firewall and TCP/IP stack products to build Internet-based Virtual Private Networks (VPNs). Currently, users and administrators are often locked in to single-vendor solutions network-wide, because vendors have been unable to agree upon the details of IPSec implementation. The S/WAN effort should therefore remove a major obstacle to the widespread deployment of secure VPNs.

S/WAN supports encryption at the IP level, which provides more fundamental, lower-level security than higher-level protocols, such as SSL (see Question 134) and S-HTTP (see Question 133). It is expected that higher-level security specifications, including SSL and S-HTTP, will be routinely layered on top of S/WAN implementations, and these security specifications will work together synergistically.

To guarantee IPsec interoperability, S/WAN defines a common set of algorithms, modes, and options. S/WAN uses RC5 (see Question 76) at key sizes ranging from 40 bits (for exportability) to 128 bits. S/WAN can also be implemented using DES (see Question 64).

Electronic Commerce

138. What is Electronic Money?

Electronic money (also called *electronic cash* or *digital cash*) is a term that is still fairly vague and undefined. It refers to transactions carried out electronically with a net result of funds transferred from one party to another. Electronic money may be either debit or credit. Digital cash per se is basically another currency, and digital cash transactions can be visualized as a foreign exchange market. This is because we need to convert an amount of money to digital cash before we can spend it, and the conversion process is analogous to purchasing foreign currency.

Pioneer work on the theoretical foundations of digital cash was carried out by Chaum [Cha83][Cha85]. Digital cash in its precise definition may be anonymous or identified in nature. Anonymous schemes do not reveal the identity of the customer and are based on blind signature schemes (see Question 39). Identified spending schemes always reveal the identity of the customer and are based on more general forms of signature schemes. Anonymous schemes are the electronic analog of cash, while identified schemes are the electronic analog of a debit or credit card. There are other approaches - payments can be anonymous with respect to the merchant but not the bank; or anonymous to everyone, but traceable (a sequence of purchases can be related, but not linked directly to the spender’s identity).

Since digital cash is merely an electronic representation of funds, it is possible to easily duplicate and spend a certain amount of money more than once. Therefore, digital cash schemes have been structured so that it is not possible to spend the same money more than once without getting caught immediately or within a short period of time. Another approach is to have the digital cash stored in a secure device, which prevents the user from double spending.

Electronic money also encompasses payment systems that are analogous to traditional credit cards and checks. Here, cryptography protects conventional transaction data such as an account number and amount; a digital signature can replace a handwritten signature or a credit-card authorization, and public-key encryption can provide confidentiality. There are a variety of systems for this type of electronic money, ranging from those that are strict analogs of conventional paper transactions with a typical value of several dollars or more, to those (not digital cash per se) that offer a form of “micropayments” where the transaction value may be a few pennies or less. The main difference is that for extremely low-value transactions even the limited overhead of public-key encryption and digital signatures is too much, not to mention the cost of "clearing" the transaction with a bank. As a result, "batching" of transactions is required, with the public-key operations done only occasionally.

Several Web pages surveying payment systems and other forms of electronic money are available, including the following:

- `<http://ganges.cs.tcd.ie/mepeirce/Project/oninternet.html>`, by Michael Peirce
- `<http://www.w3.org/hypertext/WWW/Payments/roadmap.html>`, by Phillip Hallam-Baker
- `<http://nii.isi.edu/info/netcheque/related.html>`, part of the NetCheque project at the Information Sciences Institute (University of Southern California)

The following questions cover a few of the more recent and prominent of these efforts, including IBM's iKP (see Question 139), Visa-MasterCard's SET (see Question 140), and Netscape's Secure Courier (see Question 141).

139. What is iKP?

iKP is the Internet Keyed Payments Protocol, an architecture for secure payments involving three or more parties [BGH95]. Developed at IBM's T.J. Watson Research Center and Zurich Research Laboratory, the protocol defines transactions of a "credit card" nature, where a buyer and seller interact with a third party "acquirer," such as a credit-card system or a bank, to authorize transactions. The protocol is based on public-key cryptography.

A typical iKP transaction involves six flows, with the following simplified descriptions:

- INITIATE, which the buyer sends to the seller to begin the transaction.
- INVOICE, the seller's response, which optionally contains the seller's signature on the transaction data (amount, description, etc.).
- PAYMENT, the buyer's response, which contains a payment "slip" including the buyer's account number and possibly a Personal Identification Number (PIN), encrypted with the acquirer's public key; and optionally, the buyer's signature on the transaction data.
- AUTH-REQUEST, which the seller sends to the acquirer, containing the encrypted payment slip.
- AUTH-RESPONSE, the acquirer's response to the seller, containing the acquirer's signature on the transaction data.
- CONFIRM, a confirmation from the seller to the buyer that the transaction has been authorized.

Messages between the buyer and the seller are assumed to be sent over the Internet; the messages between the seller and the acquirer may be sent over the Internet, or over a private financial network. The main cryptographic protections are the encryption of the payment slip with the acquirer's public key and the acquirer's signature on the authorization. Neither the seller nor anyone else can obtain the payment slip, so the buyer's account number is kept private. (This is a significant improvement over conventional credit-card systems.) The buyer's and seller's signatures are optional, providing additional protection against repudiation.

As in the conventional credit-card case, the acquirer separately reconciles the transaction with the buyer's issuing bank; this is outside the scope of iKP.

iKP currently specifies RSA (see Question 8) as its public-key encryption and signature algorithm; the encryption technique is called "RSA encryption with integrity" and is based on Bellare and Rogaway's "Optimal Asymmetric Encryption" [BR94].

The protocol can be adapted to a variety of environments, including HTTP transactions on the World Wide Web (see Question 133).

Additional information on iKP is available from `<http://www.zurich.ibm.com/Technology/Security/extern/ecommerce/iKP.html>`. iKP has been submitted for consideration as an Internet standard.

140. What is SET?

Visa and MasterCard have jointly developed the Secure Electronic Transaction (SET) protocol as a method for secure, cost-effective bankcard transactions over open networks. SET includes messages for purchasing goods and services electronically, requesting authorization of payment, and requesting "credentials" (i.e., certificates) binding public keys to identities, among other services.

SET supports DES (see Question 64) for bulk data encryption and RSA (see Question 8) for signatures and public-key encryption of data encryption keys and bank card numbers. The RSA public-key encryption employs Optimal Asymmetric Encryption Padding [BR94]. RSA Data Security plans to fully support the SET specification throughout the RSA product line.

SET is being published as open specifications for the industry, which may be used by software vendors to develop applications. More information can be found at `<http://www.visa.com>` and `<http://www.mastercard.com>`.

141. What is Secure Courier?

Secure Courier is Netscape's proposed protocol for secure electronic commerce on the Internet [Elg95]. It is intended to layer on top of SSL (see Question 134) or a similar protocol.

Like iKP (see Question 139), Secure Courier is based on a credit-card model involving a third party acquiring bank; messages are sent between the customer and the merchant, and between the merchant and the acquirer gateway. It is simpler in some respects from other payment protocols in that it leaves some of the message integrity and confidentiality services to SSL or another underlying layer. For instance, a "transaction ID" field in the message is sufficient for linking messages in a transaction and preventing replay, since the underlying layer assures that the field is not modified. In other protocols, the transaction ID would need explicit cryptographic protection.

Secure Courier supports RSA public-key encryption and digital signatures (see Question 8), as well as DES (see Question 64); "payment slips" from the customer to the acquirer gateway can be sent with one of the PKCS #7 message formats (see Question 166).

Authentication Systems

142. What are Identification Schemes and Authentication Protocols?

Identification schemes are methods by which a user may prove his or her identity to somebody else, without revealing essential knowledge that may be used by either an eavesdropper or the recipient to impersonate the user. If Alice authenticates herself to Bob, with Carol eavesdropping, neither Bob nor Carol should be able to impersonate Alice in the future.

A number of such schemes have been described in cryptographic literature. The traditional form of identification is by use of a secret key or password. Alice identifies herself to a server by entering the password for her account. Unfortunately, this scheme is insecure, as an eavesdropper may *replay* previous messages (e.g. the password) from Alice to the server to impersonate Alice. This is an example of a *unilateral identification scheme,* as one person in the scheme identifies herself without obtaining identification from the other person.

A more common scheme is the use of digital signatures (see Question 3) and public-key cryptography using challenge-response protocols: Bob keeps challenging Alice with different questions and assesses the answers until he is satisfied that Alice is who she claims to be. In one such scheme, Bob generates a random number and transmits it to Alice. Alice generates a new random number and then digitally signs a message containing both her random number and Bob's. She then sends the signed message, together with her random number, to Bob. Bob verifies the signature to ensure that he is communicating with Alice (one such challenge is enough). This scheme is secure against an eavesdropper; it also prevent Bob from impersonating Alice later. Until this point, the protocol is a unilateral authentication scheme.

To identify himself, Bob may also sign the message containing both random numbers and send the signature back to Alice. Alice verifies the signature to assure herself of Bob's identity. This is an example of a *mutual identification scheme,* as both parties identify themselves to each other.

See [BDB92] for further information on identification schemes and zero knowledge protocols (see Question 107), and see [Oka93] for some examples of identification and signature schemes.

143. What is Kerberos?

Kerberos [KN93][KNT94] is an authentication service developed by the Project Athena team at MIT, based on a 1978 paper by Needham and Schroeder [NS78]. The first general use version was version 4. Version 5, which addressed certain shortfalls in version 4, was released in 1994. Kerberos uses secret-key ciphers (see Question 1) for encryption and authentication. Version 4 could only use DES (see Question 64). Unlike a public-key authentication system, it does not produce digital signatures (see Question 3); Kerberos was designed to authenticate requests for network resources rather than to authenticate authorship of documents. Thus, Kerberos does not provide for future third-party verification of documents.

In a Kerberos system, there is a designated site on each network, called the Kerberos server, which performs centralized key management and administrative functions. The server maintains a database containing the secret keys of all users, authenticates the identities of users, and distributes session keys to users and servers who wish to authenticate one another. Kerberos requires trust in a third party, in this case the Kerberos server. If the server were compromised, the integrity of the whole system would be lost. Public-key cryptography was designed precisely to avoid the necessity to trust third parties with secrets (see Question 3). Kerberos is generally considered adequate within an administrative domain; across domains, the more robust functions and properties of public-key systems are often preferred. There has been some developmental work in incorporating public-key cryptography into Kerberos [Gan95].

CRYPTOGRAPHY APIS

144. What are CAPIs?

A CAPI, or *cryptographic application programming interface,* is an interface to a library of functions that software developers can call upon for security and cryptography services. The goal of a CAPI is to make it easy for developers to integrate cryptography into applications. Separating the cryptographic routines from the software may also allow the export of software without any security services implemented. The software can later be linked by the user to the local security services. CAPIs can be targeted at different levels of abstraction, ranging from cryptographic module interfaces to authentication service interfaces. The International Cryptography Experiment (ICE) is an informally structured program for testing NSA's export restrictions (see Question 148 and Question 149) on CAPIs. More information can be obtained about this program by e-mail to `<ice@tis.com>`. Some examples of CAPIs include RSA Laboratories' Cryptoki (PKCS #11) [RSA95], NSA's Fortezza (see Question 156), Internet GSS-API [Lin93], and the X/Open GCS-API [Xop95]. NSA has prepared a helpful report [NSA95]that surveys some of the current CAPIs.

Part Three

THE REAL WORLD

GOVERNMENT INVOLVEMENT

NIST & NSA

145. What is NIST?

NIST is an acronym for the National Institute of Standards and Technology, a division of the U.S. Department of Commerce; it was formerly known as the National Bureau of Standards (NBS). Through its Computer Systems Laboratory it aims to promote open systems and interoperability that will spur development of computer-based economic activity. NIST issues standards and guidelines that it hopes will be adopted by all computer systems in the U.S., and also sponsors workshops and seminars. Official standards are published as FIPS (Federal Information Processing Standards) publications.

In 1987 Congress passed the Computer Security Act, which authorized NIST to develop standards for ensuring the security of sensitive but unclassified information in government computer systems. It encouraged NIST to work with other government agencies and private industry in evaluating proposed computer security standards. However, it seems that NIST-standard cryptography such as the Escrowed Encryption Standard [NIS94a] and DSS [NIS94b] are hardly used in government, and virtually not at all by industry.

146. What Role Does NIST Play in Cryptography?

NIST issues standards for cryptographic routines; U.S. government agencies are required to use them, and the private sector often adopts them as well. In January 1977, NIST declared DES (see Question 64) the official U.S. encryption standard and published it as FIPS Publication 46; DES soon became a de facto standard throughout the U.S.

A few years ago, NIST was asked to choose a set of cryptographic standards for the U.S.; this has become known as the Capstone project (see Question 150). After a few years of rather secretive deliberations, and in cooperation with the NSA (see Question 148), NIST issued proposals for various standards in cryptography, including digital signatures (DSS, see Question 26) and data encryption (the Clipper chip, see Question 151); these are pieces of the overall Capstone project.

NIST has been criticized for allowing the NSA too much power in setting cryptographic standards, since the interests of the NSA conflict with that of the Commerce Department and NIST. Yet, the NSA has much more experience with cryptography, and many more qualified cryptographers and cryptanalysts, than does NIST; it would be unrealistic to expect NIST to forego such available assistance.

147. What is the NSA?

The NSA is the National Security Agency, a highly secretive agency of the U.S. government that was created by Harry Truman in 1952; its very existence was kept secret for many years. For a history of the NSA, see Bamford [Bam82]. The NSA has a mandate to listen to and decode all foreign communications of interest to the security of the United States. It has also used its power in various ways (see Question 149) to slow the spread of publicly available cryptography in order to prevent national enemies from employing encryption methods too strong for the NSA to break.

As the premier cryptographic government agency, the NSA has huge financial and computer resources and employs a host of cryptographers. Developments in cryptography achieved at the NSA are not made public; this secrecy has led to many rumors about the NSA's ability to break popular cryptosystems like DES (see Question 64), as well as rumors that the NSA has secretly placed weaknesses, called "trap doors," in government-endorsed cryptosystems. These rumors have never been proved or disproved, and the criteria used by the NSA in selecting cryptography standards have never been made public.

Recent advances in the computer and telecommunications industries have placed NSA actions under unprecedented scrutiny, and the agency has become the target of heavy criticism for hindering U.S. industries that wish to use or sell strong cryptographic tools. The two main reasons for this increased criticism are the collapse of the Soviet Union and the development and spread of commercially available public-key cryptographic tools. Under pressure, the NSA may be forced to change its policies.

148. What Role Does the NSA Play in Commercial Cryptography?

The NSA's charter limits its activities to foreign intelligence. However, the NSA is concerned with the development of commercial cryptography because the availability of strong encryption tools through commercial channels could impede the NSA's mission of decoding international communications; in other words, the NSA is worried lest strong commercial cryptography fall into the wrong hands.

The NSA has stated that it has no objection to the use of secure cryptography by U.S. industry. It also has no objection to cryptographic tools used for authentication, as opposed to privacy. However, the NSA is widely viewed to be following policies that have the practical effect of limiting and/or weakening the cryptographic tools used by law-abiding U.S. citizens and corporations; see Barlow [Bar92] for a discussion of NSA's effect on commercial cryptography.

The NSA exerts influence over commercial cryptography in several ways. First, it controls the export of cryptography from the U.S.; the NSA generally does not approve export of products used for encryption unless the key size is strictly limited. It does, however, approve for export any products used for authentication only, no matter how large the key size, so long as the product cannot be converted to be used for encryption. The NSA has also blocked encryption methods from being published or patented, citing a national security threat; see [Lan88] for a discussion of this practice. Additionally, the NSA serves an "advisory" role to NIST in the evaluation and selection of official U.S. government computer security standards; in this capacity, it has played a prominent and controversial role in the selection of DES and in the development of the group of standards known as the Capstone project (see Question 150), which includes DSS and the Clipper chip. The NSA can also exert market pressure on U.S. companies to produce (or refrain from producing) cryptographic goods, since the NSA itself is often a large customer of these companies. Examples of NSA-supported goods include Fortezza (see Question 156), the Defense Messaging System (DMS), and MISSI, the Multilevel Information System Security Initiative.

Cryptography is in the public eye as never before and has become the subject of national public debate. The status of cryptography, and the NSA's role in it, will probably continue to change over the next few years.

Clipper and Capstone

149. What is Capstone?

Capstone is the U.S. government's long-term project to develop a set of standards for publicly available cryptography, as authorized by the Computer Security Act of 1987. The primary agencies responsible for Capstone are NIST and the NSA (see Question 146 and Question 148). The plan calls for the elements of Capstone to become official U.S. government standards, in which case both the government itself and all private companies doing business with the government would be required to use Capstone.

There are four major components of Capstone: a bulk data encryption algorithm, a digital signature algorithm, a key exchange protocol, and a hash function. The data encryption algorithm is called Skipjack (see Question 80), but is often referred to as Clipper (see Question 151), which is the encryption chip that includes Skipjack. The digital signature algorithm is DSA (see Question 26) and the hash function is SHA (see Question 100). The key exchange protocol is not published, but is generally considered to be related to Diffie-Hellman (see Question 24).

The Skipjack algorithm and the concept of a Law Enforcement Access Field have been accepted as FIPS 185; DSS has been published as FIPS 186, and finally SHS has been published as FIPS 180.

All the parts of Capstone have 80-bit security: all the keys involved are 80 bits long and other aspects are also designed to withstand anything less than an "80-bit" attack, that is, an effort of 2^{80} operations. A Capstone chip is available that implements all the algorithms.

150. What is Clipper?

Clipper is an encryption chip developed and sponsored by the U.S. government as part of the Capstone project (see Question 150). Announced by the White House in April 1993 [OPS93], Clipper was designed to balance the competing concerns of federal law-enforcement agencies with those of private citizens and industry. The law-enforcement agencies wish to have access to the communications of suspected criminals, for example by wire-tapping; these needs are threatened by secure cryptography. Industry and individual citizens, however, want secure communications, and look to cryptography to provide it.

Clipper technology attempts to balance these needs by using escrowed keys. The idea is that communications would be encrypted with a secure algorithm, but the keys would be kept by one or more third parties (the "escrow agencies"), and made available to law-enforcement agencies when authorized by a court-issued warrant. Thus, for example, personal communications would be impervious to recreational eavesdroppers, and commercial communications would be impervious to industrial espionage, and yet the FBI could listen in on suspected terrorists or gangsters.

Clipper has been accepted as a U.S. government standard, but the use of Clipper for business with the government as well as communications within the government is voluntary. A number of vendors, including AT&T, have announced products based on the Clipper chip.

151. How does the Clipper Chip Work?

The Clipper chip contains an encryption algorithm called Skipjack (see Question 80), whose details have not been made public. Each chip contains a unique 80-bit unit key *U*, which is escrowed in two parts at two escrow agencies; both parts must be known in order to recover the key. Also present is a serial number and an 80-bit "family key" *F*; the latter is common to all Clipper chips. The chip is manufactured so that it cannot be reverse engineered; this means that the Skipjack algorithm and the keys cannot be read off the chip.

As specified by the Escrowed Encryption Standard, when two devices wish to communicate, they first agree on an 80-bit "session key" *K*. The method by which they choose this key is left up to the implementor's discretion; a public-key method such as RSA or Diffie-Hellman seems a likely choice. The message is encrypted with the key *K* and sent; note that the key *K* is not escrowed. In addition to the encrypted message, another piece of data, called the law-enforcement access field (LEAF), is created and sent. It includes the session key *K* encrypted with the unit key *U*, then concatenated with the serial number of the sender and an authentication string, and then, finally, all encrypted with the family key. The exact details of the law-enforcement field are classified.

The receiver decrypts the law-enforcement field, checks the authentication string, and decrypts the message with the key *K*.

Now suppose a law-enforcement agency wishes to tap the line. It uses the family key to decrypt the law-enforcement field; the agency now knows the serial number and has an encrypted version of the session key. It presents an authorization warrant to the two escrow agencies along with the serial number. The escrow agencies give the two parts of the unit key to the law-enforcement agency, which then decrypts to obtain the session key *K*. Now the agency can use *K* to decrypt the actual message.

Further details on the Clipper chip operation, such as the generation of the unit key, are sketched by Denning [Den93].

152. Who are the Escrow Agencies?

The current escrow agents are the NIST and the automated systems division of the department of treasury [DB95]. It is possible that the escrow agents may be changed to include at least one commercial organization.

It is essential that the escrow agencies keep the key databases extremely secure, since unauthorized access to both escrow databases could allow unauthorized eavesdropping on private communications. In fact, the escrow agencies are likely to be one of the major targets for anyone trying to compromise the Clipper system; the Clipper chip factory is another likely target.

153. Why is Clipper Controversial?

The Clipper chip proposal has aroused much controversy and has been the subject of much criticism. Unfortunately two distinct issues have become confused in the large volume of public comment and discussion.

First there is controversy about the whole idea of escrowed keys. Those in favor of escrowed keys see it as a way to provide secure communications for the public at large while allowing law-enforcement agencies to monitor the communications of suspected criminals. Those opposed to escrowed keys see it as an unnecessary and ineffective intrusion of the government into the private lives of citizens. They argue that escrowed keys infringe their rights of privacy and free speech. It will take a lot of time and much public discussion for society to reach a consensus on what role, if any, escrowed keys should have.

The second area of controversy concerns various objections to the specific Clipper proposal, that is, objections to this particular implementation of escrowed keys, as opposed to the idea of escrowed keys in general. Common objections include: the Skipjack algorithm is not public (see Question 80) and may not be secure; the key escrow agencies will be vulnerable to attack; there are not enough key escrow agencies; the keys on the Clipper chips are not generated in a sufficiently secure fashion; there will not be sufficient competition among implementors, resulting in expensive and slow chips; software implementations are not possible; and the key size is fixed and cannot be increased if necessary.

Micali [Mic93] has proposed an alternative system that also attempts to balance the privacy concerns of law-abiding citizens with the investigative concerns of law-enforcement agencies. Called fair public-key cryptography, it is similar in function and purpose to the Clipper chip proposal but users can choose their own keys, which they register with the escrow agencies. Also, the system does not require secure hardware, and can be implemented completely in software. Desmedt [Des95] has also developed a secure software-based key escrow system that could be a viable alternative. There have been numerous other proposals in the cryptographic community over the last few years; Denning and Branstad give a nice survey [DB95].

154. What is the Current Status of Clipper?

Clipper has been accepted as FIPS 185 [NIS94a] by the federal government. There were some suggestions that Clipper might be abandoned when Vice President Gore invited the industry to look into alternative ciphers that implemented key escrow technology. However, both Clipper and Capstone are being produced and sold with extensions planned. Already 9000 units of the Clipper chip has been sold to federal agencies and about 1000 units have been sold to commercial users.

155. What is Fortezza?

Fortezza, formerly called Tessera, is a PCMCIA card developed by NSA that implements the Capstone algorithms. It is meant for use with the Defense Messaging Service (DMS). A number of vendors have announced support for the Fortezza card; NSA has also built and demonstrated a PKCS #11-based library (see Question 166) that interfaces to the card.

Digital Signature Law

156. What is the Law Concerning Digital Signatures?

Just as traditional handwritten (holographic) signatures link people to the content of their agreements in a legally recognized manner, digital signatures can provide similar (but not identical) functions for electronic commerce and other purposes. Perhaps most importantly, digital signatures contribute to non-repudiation — a security service that is increasingly appreciated within the legal and business communities to provide important benefits.

The legal status of digital signatures for many, diverse applications has meaningfully advanced during the past few years. Even undigitally signed messages and records, such as those utilizing traditional electronic data interchange (EDI) or simple e-mail, have gained considerable legal recognition. The lack of litigation is, arguably, testament to the practical use and legal effectiveness of digital practices. The following developments support this assessment.

In 1989, electronic funds transfer laws, such as Article 4A of the *Uniform Commercial Code* and later the United Nations Commission on International Trade Law's (UNCITRAL's) *Model Law on International Credit Transfers*, adopted *authentication procedures* rather than *traditional signatures* as the basis for verifying transactions and apportioning liability. In 1990, the U.S. Department of Justice issued its *Guidelines on the Admissibility of Electronically Filed Federal Records as Evidence*, which emphasized the reliability and trustworthiness of computer-based data for evidentiary purposes. In 1991, the comptroller general of the United States issued a decision entitled "Use Of Electronic Data Interchange Technology to Create Valid Obligations" that authorized EDI for government contractual obligations "using properly secured EDI systems" and considered the permissible uses of digital signatures. The comptroller general's decision is only one effort, albeit an especially important one, to resolve information security and signature issues.

In 1992, the House of Delegates of the American Bar Association (ABA) went on record supporting government action, to "encourage the use of appropriate and properly implemented security techniques, procedures and practices to assure the authenticity and integrity of information in electronic form." It also recognized that "information in electronic form, where appropriate, may be considered to satisfy legal requirements regarding a written signature to the same extent as information on paper or in other conventional forms when appropriate security techniques, practices, and procedures have been adopted." In 1994, the first comprehensive legal study of digital signature infrastructure was published, *Federal Certification Authority Liability and Policy*, under the auspices of the U.S. government. The study urged the government to forge ahead with implementations and recognized that liability and other legal concerns could be appropriately controlled.

One of the longest and most notorious legal efforts (concerning signatures) has been to reform *statutes of frauds*, which require traditional writings and signatures to make certain transactions enforceable. The ongoing revision process of Article 2 of the *Uniform Commercial Code* (addressing commercial sales law) now contemplates the statutes' revision or elimination.

And, of course, 1995 commenced the adoption or consideration of digital signature legislation in various U.S. states. The first Digital Signature Act became law in Utah in May 1995, followed shortly thereafter by California, and other states are contemplating various forms of digital signature legislation. Such legislative efforts generally seek to make digital signatures at least as legally effective as traditional handwritten signatures (for certain purposes). Most recently, draft *Digital Signature Guidelines* developed by the Information Security Committee, Section of Science and Technology, American Bar Association, have been released for comment — the *Guidelines* place digital signatures *at least* on a par with holographic signatures.

Although further law reform is both inevitable and necessary, these developments present a very encouraging picture — indeed one that supports wide-scale adoption of digital signatures by business and government and their corresponding recognition in the law.

STANDARDS

ANSI Standards

157. What is ANSI X9.9?

ANSI X9.9 [ANS86a] is a United States national wholesale banking standard for authentication of financial transactions. ANSI X9.9 addresses two issues: message formatting and the particular message authentication algorithm. The algorithm defined by ANSI X9.9 is the so-called DES-MAC (see Question 102) based on DES (see Question 64) in either CBC or CFB modes (see Question 82 and Question 83). A more detailed standard for retail banking was published as X9.19 [ANS86b].

The equivalent international standards are ISO 8730 [ISO87] and ISO 8731 for ANSI X9.9, and ISO 9807 for ANSI X9.19. The ISO standards differ slightly in that they do not limit themselves to DES to obtain the message authentication code but allow the use of other message authentication codes and block ciphers.

158. What is ANSI X9.17?

ANSI X9.17 [ANS85] is the Financial Institution Key Management (Wholesale) standard. It defines the protocols to be used by financial institutions such as banks to transfer encryption keys. This protocol is aimed at the distribution of secret keys using symmetric (secret-key) techniques (see Question 1).

Financial institutions need to change their bulk encryption keys on a daily or per-session basis due to the volume of encryptions performed. This does not permit the costs and other inefficiencies associated with manual transfer of keys. The standard therefore defines a three-level hierarchy of keys:

- The highest level is the master key (KKM), which is always manually distributed.
- The next level consists of key-encrypting keys (KKs), which are distributed on-line.
- The lowest level has data keys (KDs), which are also distributed on-line.

The data keys are used for bulk encryption and are changed on a per-session or per-day basis. New data keys are encrypted with the key-encrypting keys and distributed to the users. The key-encrypting keys are changed periodically and encrypted with the master key. The master keys are changed less often but are always distributed manually in a very secure manner.

ANSI X9.17 defines a format for messages to establish new keys and replace old ones called CSM (cryptographic service messages). ANSI X9.17 also defines two-key triple-DES encryption (see Question 72) as a method by which keys can be distributed. ANSI X9.17 is gradually being supplemented by public-key techniques such as Diffie-Hellman (see Question 24).

One of the major limitations of ANSI X9.17 is the inefficiency of communicating in a large system since each pair of terminal systems that need to communicate with each other will need to have a common master key. To resolve this problem, ANSI X9.28 was developed to support the distribution of keys between terminal systems that do not share a common key center. The protocol defines a multiple-center group as two or more key centers that implement this standard. Any member of the multiple-center group is able to exchange keys with any other member.

159. What are ANSI X9.30 and ANSI X9.31?

ANSI X9.30 [ANS93a] is the United States financial industry standard for digital signatures based on the federal Digital Signature Algorithm (DSA), and ANSI X9.31 [ANS93b] is the counterpart standard for digital signatures based on the RSA algorithm. ANSI X9.30 requires the SHA1 hash algorithm (see Question 100); ANSI X9.31 requires the MDC-2 hash algorithm [ISO92c].

A related document, X9.57 [ANS95], covers certificate management (see Question 123).

160. What are ANSI X9.42 and ANSI X9.44?

ANSI X9.42 [ANS94a] is a draft standard for key agreement based on the Diffie-Hellman algorithm, and ANSI X9.44 [ANS94b] is a draft standard for key transport based on the RSA algorithm. The former is intended to specify techniques for deriving a shared secret key; techniques currently being considered include basic Diffie-Hellman (see Question 24), authenticated Diffie-Hellman (see Question 25), and the MQV protocols [MQV95]. Some work to unify the various approaches is currently in progress. ANSI X9.44 will specify techniques for transporting a secret key with the RSA algorithm. It is currently based on IBM's Optimal Asymmetric Encryption Padding, a "provably secure" padding technique related to work by Bellare and Rogaway [BR94].

ANSI X9.42 was previously part of ANSI X9.30, and ANSI X9.44 was previously part of ANSI X9.31 (see Question 160).

ITU-T (CCITT) Standards

161. What are Distinguished Names?

Distinguished names are the standard form of naming in an ITU-T X.500 directory [CCI88b] and in X.509 certificates (see Question 165). A distinguished name is comprised of one or more relative distinguished names, and each relative distinguished name is comprised of one or more attribute-value assertions. Each attribute-value assertion consists of an attribute identifier and its corresponding value information, e.g. “CountryName = US.”

Distinguished names were intended to identify entities in the X.500 directory tree. A relative distinguished name is the path from one node to a subordinate node. The entire distinguished name traverses a path from the root of the tree to an end node that represents a particular entity. A goal of the directory was to provide an infrastructure to uniquely name every communications entity everywhere (hence the “distinguished” in “distinguished name”). As a result of the directory’s goals, names in X.509 certificates are perhaps more complex than one might like (e.g., compared to an e-mail address). Nevertheless, for business applications, distinguished names are worth the complexity, as they are closely coupled with legal name registration procedures, something that simple names such as e-mail addresses do not offer.

162. What is X.400?

ITU-T Recommendation X. 400 [CCI88a], also known as the Message Handling System (MHS), is one of the two standard e-mail architectures used for providing e-mail services and interconnecting proprietary e-mail systems. The other is the Simple Mail Transfer Protocol (SMTP) used by the Internet. MHS allows e-mail and other store-and-forward message transferring such as Electronic business Data Interchange (EDI, see Question 164) and voice messaging. The MHS and Internet mail protocols are different but based on similar underlying architectural models. The noteworthy fact of MHS is that it has supported secure messaging since 1988 while the Internet is still considering a variety of draft standards such PEM (see Question 130), S/MIME (see Question 131), and PEM-MIME (see Question 132). The MHS message structure is similar to the MIME (see Question 133) message structure. It has both a header and a body. The body can be broken up into multiple parts, with each part being encoded differently. For example, one part of the body may be text, the next part a picture, and a third part encrypted information.

163. What is X.435?

ITU-T Recommendation X.435 [CCI91] and its equivalent F.435 are X.400-based standards (see Question 163) designed to support electronic data interchange (EDI) messaging. EDI needs more stringent security than typical e-mail because of its business nature: not only does an EDI message need protection against fraudulent or accidental modification in transit, but it also needs to be immune to repudiation after it has been sent and received.

In support of these security requirements, X.435 defines, in addition to normal EDI messages, a set of EDI “notifications.” Positive notification implies that the recipient has received the document and accepts the responsibility for it; negative notification means that the recipient refused to accept the document due to a specified reason; and forwarding notification means that the document had been forwarded to another recipient. Together, these notifications form the basis for a system that can provide security controls comparable to those in the paper-based system that EDI replaces.

164. What is X.509?

ITU-T Recommendation X.509 [CCI88c] specifies the authentication service for X.500 directories, as well as the widely adopted X.509 certificate syntax. The initial version of X.509 was published in 1988, version 2 was published in 1993, and version 3 was proposed in 1994 and considered for approval in 1995. Version 3 addresses some of the security concerns and limited flexibility that were issues in versions 1 and 2.

Directory authentication in X.509 can be carried out using either secret-key techniques or public-key techniques; the latter is based on public-key certificates. The standard does not specify a particular cryptographic algorithm, although an informative annex of the standard describes the RSA algorithm (see Question 8).

An X.509 certificate consists of the following fields:

- version
- serial number
- signature algorithm ID
- issuer name
- validity period
- subject (user) name
- subject public key information
- issuer unique identifier (version 2 and 3 only)
- subject unique identifier (version 2 and 3 only)
- extensions (version 3 only)
- signature on the above fields

This certificate is signed by the issuer to authenticate the binding between the subject (user's) name and the user's public key. The major difference between versions 2 and 3 is the addition of the extensions field. This field grants more flexibility as it can convey additional information beyond just the key and name binding. Standard extensions include subject and issuer attributes, certification policy information, and key usage restrictions, among others.

X.509 also defines a syntax for certificate revocation lists (CRLs) (see Question 129).

The X.509 standard is supported by a number of protocols, including PEM (see Question 130), PKCS (see Question 166), S-HTTP (see Question 133), and SSL (see Question 134).

PKCS

165. What is PKCS?

The Public-Key Cryptography Standards (PKCS) is a set of standards for public-key cryptography, developed by RSA Laboratories in cooperation with an informal consortium, originally including Apple, Microsoft, DEC, Lotus, Sun and MIT. PKCShas been cited by the OIW (OSI Implementors' Workshop) as a method for implementation of OSI standards. PKCS is compatible with PEM (see Question 130) but extends beyond PEM. For example, where PEM can only handle ASCII data, PKCS is designed for binary data as well. PKCS is also compatible with the ITU-T X.509 standard (see Question 165). The published standards are PKCS #1, #3, #5, #6, #7, #8, #9, #10 and #11.

PKCS includes both algorithm-specific and algorithm-independent implementation standards. Algorithms supported include RSA (see Question 8) and Diffie-Hellman key exchange (see Question 24), among many others. However, only RSA and Diffie-Hellman are specifically detailed. It also defines an algorithm-independent syntax for digital signatures (see Question 3), digital envelopes (for encryption, see Question 16), and extended certificates; this enables someone implementing any cryptographic algorithm whatsoever to conform to a standard syntax, and thus achieve interoperability. Documents detailing the PKCS standards can be obtained at RSA Data Security's FTP server (accessible from `<http://www.rsa.com>` or via anonymous ftp to `<ftp.rsa.com>` or by sending e-mail to `<pkcs@rsa.com>`).

- PKCS #1 defines mechanisms for encrypting and signing data using RSA public-key cryptosystem.
- PKCS #3 defines a Diffie-Hellman key agreement protocol.
- PKCS #5 describes a method for encrypting a string with a secret key derived from a password.
- PKCS #6 describes a format for extended certificates. An extended certificate consists of a X.509 certificate (see Question 165) together with a set of attributes signed by the issuer of the certificate. PKCS #6 is being phased out in favor of version 3 of X.509.
- PKCS #7 defines a general syntax for messages that include cryptographic enhancements such as digital signatures and encryption.

- PKCS #8 describes a format for private-key information. This information includes a private key for some public-key algorithm, and optionally a set of attributes.
- PKCS #9 defines selected attribute types for use in the other PKCS standards.
- PKCS #10 describes a syntax for certification requests.
- PKCS #11 defines a technology-independent programming interface, calledCryptoki, for cryptographic devices such as smart cards and PCMCIAcards.

It is RSA Laboratories' intention to revise the PKCS documents from time to time to keep track of new developments in cryptography and data security, as well as to transition the documents into open standards development efforts as opportunities arise.

Other Standards

166. What is IEEE P1363?

P1363 is the IEEE working group that is developing standards for public-key cryptography based on RSA (see Question 8), Diffie-Hellman (see Question 24), and related algorithms. The scope of the standards includes encryption algorithms such as RSA, Diffie-Hellman, ElGamal (see Question 29), and elliptic curve cryptosystems (see Question 31), as well as random number generation and hardware support. A first version of the standard covers elliptic curve cryptosystems and is nearly complete.

More information on P1363 can be obtained by anonymous ftp to `<ftp.rsa.com>` in the pub/p1363 directory [IEE95].

167. What is ISO/IEC 9798?

ISO/IEC 9798 [ISO92a] is an emerging international standard for entity authentication techniques. It consists of three parts. Part 1 is introductory, and Parts 2 and 3 define protocols for entity authentication using secret-key techniques and public-key techniques. Parts 4 and 5 are currently in preparation. Part 4 defines protocols based on cryptographic checksums, and part 5 is planned to address zero-knowledge techniques.

168. What is ISO/IEC 9979?

ISO/IEC 9979 [ISO91] is a standard that defines the procedures for a service that registers cryptographic algorithms. Registering a cryptographic algorithm results in a unique identifier being assigned to it. The registration is achieved via a single organization called the registration authority. The registration authority does not evaluate or make any judgment on the quality of the protection provided.

169. What are the Orange and Red Books?

The Department of Defense (DOD) publication *Trusted Computer System Evaluation Criteria* (TCSEC) is also called the Orange Book. It specifies the criteria the DOD uses in evaluating the security of a product. The assessed features include the security policy, marking, identification, accountability, assurance, and continuous protection of the system. Based on the assessment, the security of the system is classified into one of four hierarchies, with A providing the most security and D providing minimal or non-existent security. Each hierarchy has a number of levels as well.

The Red Book was published to provide subsidiary information to enable the Orange Book principles to be applied in a network environment. The Red Book was initially published as the *Trusted Network Interpretation (TNI) of the Trusted Computer System Evaluation Criteria*. Acceptance of these criteria has grown to the extent that some commercial companies require their purchases to satisfy a specific level of security as described in the Orange and Red Books.

170. What is TEMPEST?

TEMPEST is a standard for electromagnetic shielding for computer equipment. It was devised to address the threat that information might be easily read from a distance viaemanation of radiation from computer systems. For instance, an attacker might intercept the radiation from the CRT of a terminal that is not protected against emission of stray radiation, thwarting any cryptographic protection that might otherwise be in place. TEMPEST describes techniques for countering these threats. Formore details about TEMPEST, see [RG91].

PRODUCTS

RSA Data Security, Inc. Products

171. What is BCERT?

BCERT 1.0 is RSA’s first general-purpose toolkit for issuing X.509 certificates (see Question 165) and certificate revocation lists (see Question 129). BCERT fully supports the standard X.509 version 3 Certificate and CRL Extensions. It also offers additional features that allow developers to define their own extensions and to customize the supported standard extensions.

BCERT contains all the cryptographic support necessary for generating certificate requests and preparing, signing, and verifying certificates and CRLs.

172. What is BSAFE?

BSAFE is a general purpose, low-level cryptographic toolkit that offers developers the tools to add privacy and authentication features to their applications.

BSAFE is designed to provide the security tools for a wide range of applications, such as digitally signed electronic forms, virus detection, or virtual private networks. It is compatible with various industry standards, including S/MIME, S-HTTP, SSL, PCT, S/WAN and SET (see Question 131, Question 133, Question 134, Question 136, Question 137 and Question 140). BSAFE also fully supports PKCS (see Question 166).

The 3.0 release of BSAFE contains the following algorithms:

Public-Key Algorithms:

- RSA Public Key Cryptosystem
- Diffie-Hellman Key Negotiation
- U.S. Digital Signature Algorithm (DSA)

Secret-Key Algorithms:

- Data Encryption Standard (DES)
- Triple-DES
- Extended Data Encryption Standard (DESX)
- RC2 Variable-Key-Size Symmetric Block Cipher (exportable)
- RC4 Variable-Key-Size Symmetric Stream Cipher (exportable)
- RC5 Variable-Key-Size Symmetric Block Cipher

Cryptographic Hashing Algorithms:

- MD Hashing Algorithm
- MD2 Hashing Algorithm
- MD5 Hashing Algorithm
- Secure Hashing Algorithm (SHA1)

Other Cryptographic Functions:

- Bloom-Shamir Secret Sharing Algorithm
- Pseudo-Random Number Generation

The 3.0 release of BSAFE offers greater security by supporting public-key operations with up to 2048-bit keys, and better performance by enhancing the throughput of both the public-key and secret-key algorithms. BSAFE is written in portable C, and is available on a wide variety of platforms.

For more information on BSAFE, see `<http://www.rsa.com>`.

173. What is RSAREF?

RSAREF is a free, portable software developer's library of popular encryption and authentication algorithms. The name "RSAREF" means "RSA reference." RSA Laboratories intends RSAREF to serve as a free, educational reference implementation of modern public- and secret-key cryptography.

The current version of RSAREF is 2.0, which supports the following algorithms:

- RSA encryption and key generation, as defined by PKCS
- MD2 and MD5 message digests
- DES (Data Encryption Standard) in cipher-block chaining mode
- Diffie-Hellman key agreement
- DESX, RSA Data Security's efficient, secure DES enhancement
- Triple-DES, three DES operations for added security

Version 2.0 offers three other improvements over RSAREF 1.0: the ability to process messages of arbitrary length in parts; the option to process either binary data, or data encoded in printable ASCII; and support for encrypting messages for more than one recipient.

RSAREF is written in the C programming language as a library that can be called from an application program. A simple Internet Privacy-Enhanced Mail (PEM) implementation can be built directly on top of RSAREF, together with message parsing and formatting routines and certificate-management routines. RSAREF is distributed with a demonstration program that shows how one might build such an implementation.

For information on downloading RSAREF over the Internet, send e-mail to `<rsaref@rsa.com>`.

174. What is RSA Secure?

RSA Secure is a commercial software package for Windows and Macintosh computers, created by RSA Data Security, Inc. Its primary function is to encrypt the files on a user's computer or local area network. The key features of RSA Secure include ease of use, strong encryption, emergency access with key splitting, and file sharing. In more detail:

- Ease of use: RSA Secure is not a separate application; on a PC, it actually becomes part of the Windows File Manager. RSA Secure also provides a "set it and forget it" utility which automatically encrypts a user's files when the user exits Windows, shuts down his/her Macintosh, or chooses the "encrypt now" function. Individual files or entire directories can be chosen in advance for this "AutoCrypt" function, or can be encrypted on an ad hoc basis.

- Strong encryption: Individual files are encrypted using RSA's RC4 stream cipher, using 128-bit encryption for versions sold in the USA and Canada and 40-bit encryption for versions sold overseas. The emergency access RSA public/private key pair is set at 1024 bits domestically and 512 bits overseas.

- Emergency access with key splitting: RSA Secure implements a corporate-based key escrow system whereby every file that is encrypted has the random RC4 key for that file, in turn encrypted by both the user's secret key and the organization's public key. The organization's private key might then be used to decrypt a single user's files. However, to prevent potential abuse, RSA Secure implements Bloom-Shamir secret splitting, which divides the private key into discrete "shares" which are held by a number of trustees.

- File sharing: Other than protecting files on a user's computer, RSA Secure implements an additional option of encryption, allowing for ad hoc password-based encryption. A file encrypted in this manner can be transferred to another RSA Secure user for decryption. If the second user does not have RSA Secure, the encrypted file can be made a self-extracting executable file, in which case the second user runs the file, enters the password, and thus decrypts the file.

A free evaluation copy of RSA Secure can be downloaded from RSA Data Security at `<http://www.rsa.com>`.

175. What is TIPEM?

TIPEM is a cryptographic Toolkit for Interoperable Privacy-Enhanced Messaging. It is a library that provides the tools for developing secure messaging applications. In addition to electronic mail, TIPEM can be used for secure messaging-based applications, such as electronic commerce, EDI, and electronic forms routing. The Certificate Management Extensions (CME) in TIPEM facilitate issuing, management, and tracking of certificates from within the application.

TIPEM supports the X.509 certificate standards and secure messaging industry standards, such as S/MIME, PKCS#7, PEM, and X.400. It provides the following algorithms:

- RSA Public Key Cryptosystem
- Data Encryption Standard (DES)
- RC2 Variable-Key Size Symmetric Block Cipher (exportable)
- MD2 Hashing Algorithm
- MD5 Hashing Algorithm

TIPEM is written entirely in portable C, and is available on a wide variety of platforms. For more information on TIPEM, see `<http://www.rsa.com>`.

Other Products

176. What is PGP?

Pretty Good Privacy (PGP) is a software package originally developed by Phil Zimmerman that provides cryptographic routines for e-mail and file storage applications. Zimmerman took existing cryptosystems and cryptographic protocols and developed a freeware program that can run on multiple platforms. It provides message encryption, digital signatures, data compression, and e-mail compatibility.

The algorithms used for message encryption are RSA (see Question 8) for key transport and IDEA (see Question 77) for bulk encryption of messages. Digital signatures are achieved by the use of RSA (see Question 8) for signing and MD5 (see Question 99) for computing the message digest. The freeware program ZIP is used to compress messages for transmission and storage. E-mail compatibility is achieved by the use of Radix-64 conversion.

MIT PGP versions 2.6 and later are legal freeware for non-commercial use based on RSAREF (see Question 174). Viacrypt PGP versions 2.7 and later are legal commercial versions of the same software. PGP is bound by Federal export laws due to the use of the RSA public key cryptosystem.

177. What is RIPEM?

RIPEM is a program developed by Mark Riordan and enhanced by Jeff Thompson that enables secure Internet e-mail; it provides both encryption and digital signatures, using RSA and DES routines from RSAREF (see Question 174). RIPEM is compatible with PKCS #7 and PKCS #10 (see Question 166) in support of S/MIME and other PKCS-based messaging. RIPEM implements certificates, certification hierarchies and CRLs (see Question 123 through Question 129). RIPEM is also PEM-compatible and provides a convenient application programming interface which lets e-mail handlers link to RIPEM's message-handling functions. RIPEM is available free for non-commercial use in the U.S. and Canada. To get RIPEM, read `<ftp://ripem.msu.edu/pub/crypt/ripem/GETTING_ACCESS>`.

REFERENCES

[ACG84] W. Alexi, B. Chor, O. Goldreich, and C.P. Schnorr. RSA and Rabin functions: Certain parts are as hard as the whole. *SIAM Journal of Computing*, October 1984.

[Adl94] L.M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266: 1021–1024, November 1994.

[Adl95] L.M. Adleman. On constructing a molecular computer, University of Southern California, draft, January 1995.

[Adl96] L.M. Adleman. Statement, Cryptographer's Expert Panel, RSA Data Security Conference, San Francisco, CA, January 17, 1996.

[AGL95] D. Atkins, M. Graff, A.K. Lenstra and P.C. Leyland. The magic words are squeamish ossifrage. In *Advances in Cryptology — Asiacrypt '94*, pages 263–277, Springer-Verlag, 1995.

[ANS83] American National Standards Institute. *American National Standard X3.106: Data Encryption Algorithm, Modes of Operations*, 1983.

[ANS85] American National Standards Institute. *American National Standard X9.17: Financial Institution Key Management (Wholesale)*, 1985.

[ANS86a] American National Standards Institute. *American National Standard X9.9: Financial Institution Message Authentication (Wholesale)*, 1986.

[ANS86b] American National Standards Institute. *American National Standard X9.19: Financial Institution Retail Message Authentication*, 1986.

[ANS93a] American National Standards Institute. *Draft: American National Standard X9.30-199X: Public-Key Cryptography Using Irreversible Algorithms for the Financial Services Industry: Part 1: The Digital Signature Algorithm (DSA).* American Bankers Association, March 1993.

[ANS93b] American National Standards Institute. *American National Standard X9.31-1992: Public Key Cryptography Using Reversible Algorithms for the Financial Services Industry: Part 1: The RSA Signature Algorithm*, March 1993.

[ANS93c] American National Standards Institute. *American National Standard X9.31-1992: Public Key Cryptography Using Reversible Algorithms for the Financial Services Industry: Part 2: The MDC-2 Hash Algorithm*, June 1993.

[ANS94a] American National Standards Institute. *Accredited Standards Committee X9 Working Draft: American National Standard X9.42-1993: Public Key Cryptography for the Financial Services Industry: Management of Symmetric Algorithm Keys Using Diffie-Hellman*, American Bankers Association, September 21, 1994.

[ANS94b] American National Standards Institute. *Accredited Standards Committee X9 Working Draft: American National Standard X9.44: Public Key Cryptography Using Reversible Algorithms for the Financial Services Industry: Transport of Symmetric Algorithm Keys Using RSA*, American Bankers Association, September 21, 1994.

[ANS95] American National Standards Institute. *Accredited Standards Committee X9 Working Draft: American National Standard X9.57: Certificate Management*, American Bankers Association, 1995.

[Atk95a] R. Atkinson. *RFC 1825: Security Architecture for the Internet Protocol*. Naval Research Laboratory, August 1995.

[Atk95b] R. Atkinson. *RFC 1826: IP Authentication Header*. Naval Research Laboratory, August 1995.

[Atk95c] R. Atkinson. *RFC 1827: IP Encapsulating Security Payload (ESP).* Naval Research Laboratory, August 1995.

[Bam82] J. Bamford. *The Puzzle Palace.* Houghton Mifflin, Boston, 1982.

[Bar92] J.P. Barlow. Decrypting the puzzle palace. *Communications of the ACM*, 35(7): 25–31, July 1992.

[BBB92] C. Bennett, F. Bessette, G. Brassard, L. Savail, and J. Smolin. Experimental quantum cryptography. *Journal of Cryptology*, 5(1): 3–28, 1992.

[BBC88] P. Beauchemin, G. Brassard, C. Crepeau, C. Goutier, and C. Pomerance. The generation of random numbers that are probably prime. *Journal of Cryptology*, 1: 53–64, 1988.

[BBL95] D. Bleichenbacher, W. Bosma, and A. Lenstra. Some remarks on Lucas-based cryptosystems. In *Advances in Cryptology – Crypto '95*, pages 386–396, Springer-Verlag, 1995.

[BBS86] L. Blum, M. Blum, and M. Shub. A simple unpredicatable random number generator. *SIAM Journal on Computing*, 15: 364–383, 1986.

[BD93b] J. Brandt and I. Damgard. On generation of probable primes by incremental search. In *Advances in Cryptology — Crypto '92*, pages 358–370, Springer-Verlag, 1993.

[BDB92] M.V.D. Burmester, Y.G. Desmedt, and T. Beth. Efficient zero-knowledge identification schemes for smart cards. *Computer Journal*, 35: 21–29, 1992.

[BDK93] E.F. Brickell, D.E. Denning, S.T. Kent, D.P. Maher, and W. Tuchman. *Skipjack Review, Interim Report: The Skipjack Algorithm*. July 28, 1993.

[Bea95] D. Beaver. Factoring: The DNA solution. In *Advances in Cryptology — Asiacrypt '94*, pages 419–423, Springer-Verlag, 1995.

[Ben82] P. Benioff. Quantum mechanical Hamiltonian models of Turing machines. *Journal of Statistical Physics*, 29(3): 515–546, 1982.

[BG85] M. Blum and S. Goldwasser. An efficient probabilistic public-key encryption scheme which hides all partial information. In *Advances in Cryptology — Crypto '84*, pages 289–299, Springer-Verlag, 1985.

[BGH95] M. Bellare, J.A. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner. *iKP* - A Family of Secure Electronic Payment Protocols. Usenix Electronic Commerce Workshop, July 1995.

[BHS93] D. Bayer, S. Haber, and W.S. Stornetta. Improving the efficiency and reliability of digital timestamping. In *Proceedings Sequences II: Methods in Communication, Security, and Computer Science*, pages 329–334, Springer-Verlag, 1993.

[Bih95] E. Biham. Cryptanalysis of Multiple Modes of Operation. In *Advances in Cryptology — Asiacrypt '94*, pages 278–292, Springer-Verlag, 1995.

[BKR94] M. Bellare, J. Killian and P. Rogaway. The security of cipher block chaining. In *Advances in Cryptology — Crypto '94*, pages 341–358, Springer-Verlag, 1994.

[Bla79] G.R. Blakley. Safeguarding cryptographic keys. *AFIPS Conference Proceedings*, 48: 313–317, 1979.

[BLP94] J.P. Buhler, H.W. Lenstra, and C. Pomerance. The development of the number field sieve. Volume 1554 of *Lecture Notes in Computer Science,* Springer-Verlag, 1994.

[BLS88] J. Brillhart, D.H. Lehmer, J.L. Selfridge, B. Tuckerman, and S.S. Wagstaff Jr. *Factorizations of* $b^n \pm 1$, $b =$ 2,3,5,6,7,10,11,12 *up to High Powers*. Volume 22 of *Contemporary Mathematics,* American Mathematical Society, 2nd edition, 1988.

[BLS95] J. Benaloh, B. Lampson, D. Simon, T. Spies, and B. Yee. *The Private Communication Technology Protocol*. Version 1.00, Microsoft Corporation, Redmond, WA, October 1995. `<http://www.microsoft.com/windows/ie/PCT.htm/>`

[BLZ94] J. Buchmann, J. Loho, and J. Zayer. An implementation of the general number field sieve. In *Advances in Cryptology — Crypto '93*, pages 159–166, Springer-Verlag, 1994.

[BM84] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4): 850–863, 1984.

[BO88] E.F. Brickell and A.M. Odlyzko. Cryptanalysis: A survey of recent results. *Proceedings of the IEEE*, 76: 578–593, 1988.

[BR94] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology — Eurocrypt '94*, pages 92–111, Springer-Verlag, 1994.

[Bra88] G. Brassard. *Modern Cryptology*. Volume 325 of *Lecture Notes in Computer Science*, Springer-Verlag, 1988.

[Bra93] G. Brassard. Cryptography column — Quantum cryptography: A bibliography. *Sigact News*, 24(3): 16–20, 1993.

[Bra95a] G. Brassard. The computer in the 21st Century. *Scientific American*. March 1995.

[Bra95b] G. Brassard. The impending demise of RSA? *CryptoBytes,* 1(1): 1–4, Spring 1995.

[Bra95c] G. Brassard. A quantum jump in computer science. *Current Trends in Computer Science*, LNCS 1000, Springer-Verlag, 1995.

[Bre89] D.M. Bressoud. *Factorization and Primality Testing*. Springer-Verlag, 1989.

[Bri85] E.F. Brickell. Breaking iterated knapsacks. In *Advances in Cryptology — Crypto '84*, pages 342–358, Springer-Verlag, 1985.

[BS91a] E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. In *Advances in Cryptology – Crypto '90*, pages 2–21, Springer-Verlag, 1991.

[BS91b] E. Biham and A. Shamir. Differential cryptanalysis of FEAL and N-Hash. In *Advances in Cryptology – Eurocrypt '91*, pages 156–171, Springer-Verlag, 1991.

[BS93a] E. Biham and A. Shamir. Differential cryptanalysis of the full 16-round DES. In *Advances in Cryptology — Crypto '92*, pages 487–496, Springer-Verlag, 1993.

[BS93b] E. Biham and A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, 1993.

[CCI88a] CCITT. Recommendation X.400: *Message Handling System and Service Overview*. 1988.

[CCI88b] CCITT. Recommendation X.500: *The Directory — Overview of Concepts, Models and Services*. 1988.

[CCI88c] CCITT. Recommendation X.509: *The Directory — Authentication Framework*. 1988.

[CCI91] CCITT. Recommendation X.435: *Message Handling Systems: EDI Messaging System*. 1991.

[CFG95] S. Crocker, N. Freed, J. Galvin, and S. Murphy. *RFC 1848: MIME Object Security Services*. CyberCash, Inc., Innosoft International, Inc., and Trusted Information Systems, October 1995.

[CFN88] D. Chaum, A. Fiat and M. Naor. Untraceable electronic cash. In *Advances in Cryptology — Crypto '88,* pages 319–327, Springer-Verlag, 1988.

[Cha83] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology — Crypto '82,* pages 199–203, Springer-Verlag, 1983.

[Cha85] D. Chaum. Security without identification: transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10): 1030–1044, October 1985.

[Cha94] D. Chaum. Designated confirmer signatures. In *Advances in Cryptology — Eurocrypt '94*, pages 86–91, Springer-Verlag, 1994.

[CKM94] D. Coppersmith, H. Krawczyz and Y. Mansour. The shrinking generator. In *Advances in Cryptology — Crypto '93*, pages 22–38, Springer-Verlag, 1994.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, Massachusetts, 1990.

[Cop92] D. Coppersmith. The data encryption standard and its strength against attacks. *IBM Research Report* RC 18613 (81421), T. J. Watson research center, December 1992.

[COS86] D. Coppersmith, A.M. Odlyzko, and R. Schroeppel. Discrete logarithms. In *GF(p). Algorithmica*, 1: 1–15, 1986.

[CP94] L. Chen and T.P. Pederson. New group signature schemes. In *Advances in Cryptology — Eurocrypt '94*, pages 171–181, Springer-Verlag, 1994.

[CP95] L. Chen and T.P. Pedersen. On the efficiency of group signatures: providing information-theoretic anonymity. In *Advances in Cryptology — Eurocrypt '95*, pages 39–49, Springer-Verlag, 1995.

[CR88] B. Chor and R.L. Rivest. A knapsack-type public-key cryptosystem based on arithmetic in finite fields. *IEEE Transactions on Information Theory*, 34(5): 901–909, 1988.

[CV90] D. Chaum and H. van Antwerpen. Undeniable signatures. In *Advances in Cryptology — Crypto '89*, pages 212–216, Springer-Verlag, 1990.

[CV91] D. Chaum and E. van Heijst. Group signatures. In *Advances in Cryptology — Eurocrypt '91*, pages 257–265, Springer-Verlag, 1991.

[CV92] D. Chaum and H. van Antwerpen. Cryptographically strong undeniable signatures, unconditionally secure for the signer. In *Advances in Cryptology — Crypto '91*, pages 470–484, Springer-Verlag, 1992.

[CW93] K.W. Campbell and M.J. Wiener. DES is not a group. In *Advances in Cryptology — Crypto '92*, pages 512–520, Springer-Verlag, 1993.

[Dam90] I. Damgård. A design principle for hash functions. In *Advances in Cryptology — Crypto '89*, pages 416–427, Springer-Verlag, 1990.

[Dav82] G. Davida. *Chosen signature cryptanalysis of the RSA public key cryptosystem*. Technical Report TR-CS-82-2, Department of EECS, University of Wisconsin, Milwaukee, 1982.

[DB92] B. den Boer and A. Bosselaers. An attack on the last two rounds of MD4. In *Advances in Cryptology — Crypto '91*, pages 194–203, Springer-Verlag, 1992.

[DB94] B. den Boer and A. Bosselaers. Collisions for the compression function of MD5. In *Advances in Cryptology — Eurocrypt '93*, pages 293-304, Springer-Verlag, 1994.

[DB95] D.E. Denning and D.K. Branstad. *A taxonomy for key escrow encryption systems*. January, 1995.

[DBP96] H. Dobbertin, A. Bosselaers, and B. Preneel. RIPEMD-160: A strengthened version of RIPEMD. To appear in *3rd Workshop on Fast Software Encryption,* 1996.

[Den93] D.E. Denning. The Clipper encryption system. *American Scientist*, 81(4): 319–323, July-August 1993.

[Den95] D.E. Denning. The Case for "Clipper." *Technology Review*, pages 48-55, July 1995.

[Des95] Y. Desmedt. Securing traceability of ciphertexts–Towards a secure software key escrow system. In *Advances in Cryptology — Eurocrypt '95*, pages 147–157, Springer-Verlag, 1995.

[Deu92] D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society*, London, A439: 553–558, 1992.

[DGV94] J. Daemen, R. Govaerts, and J. Vandewalle. Weak keys for IDEA. In *Advances in Cryptology — Crypto '93*, pages 224–231, Springer-Verlag, 1994.

[DH76] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22: 644–654, 1976.

[DH77] W. Diffie and M.E. Hellman. Exhaustive cryptanalysis of the NBS Data Encryption Standard. *Computer*, 10: 74–84, 1977.

[Dif88] W. Diffie. The first ten years of public-key cryptography. *Proceedings of the IEEE*, 76: 560–577, 1988.

[DIP94] D. Davies, R. Ihaka, and P. Fenstermacher. Cryptographic randomness from air turbulence in disk drives. In *Advances in Cryptology — Crypto '94*, pages 114–120, Springer-Verlag, 1994.

[Div95] D.P. DiVincenzo. Two-bit gates are universal for quantum computation. *Physical Review A*, 51: 1015–1022, 1995.

[DL95] B. Dodson and A.K. Lenstra. NFS with four large primes: An explosive experiment. In *Advances in Cryptology – Crypto '95,* pages 372–385, Springer-Verlag, 1995.

[DO86] Y. Desmedt and A.M. Odlyzko. A chosen text attack on the RSA cryptosystem and some discrete logarithm schemes. In *Advances in Cryptology — Crypto '85*, pages 516–522, Springer-Verlag, 1986.

[Dob95] H. Dobbertin. Alf Swindles Ann. *CryptoBytes*, 1(3): 5, 1995.

[DP83] D.W. Davies and G.I. Parkin. The average cycle size of the key stream in output feedback encipherment. In *Advances in Cryptology: Proceedings of Crypto '82*, pages 97–98, Plenum Press, 1983.

[DRB95] P. Domokos, M.J. Raimond, M. Brune, and S. Haroche. A simple cavity-QED two-bit universal quantum logic gate: principle and expected performances. *Physical Review A*. To appear.

[DVW92] W. Diffie, P.C. van Oorschot, and M.J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2: 107–125, 1992.

[ECS94] D. Eastlake, 3rd, S. Crocker, and J. Schiller. *RFC 1750: Randomness Recommendations for Security*. DEC, Cybercash, and MIT, December 1994.

[Elg85] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31: 469–472, 1985.

[Elg95] T. ElGamal. *Commerce on the Internet*. Version 1.00, Netscape Communications Corporation, Mountain View, CA, July 14, 1995

[Fei73] H. Feistel. Cryptography and Computer Privacy, *Scientific American*, May 1973.

[Fey82] R.P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6): 467–488, 1982.

[Fey86] R.P. Feynman. Quantum mechanical computers. *Optic News*, February 1985. Reprinted in *Foundations of Physics*, 16(6): 507–531, 1986.

[FFS88] U. Feige, A. Fiat and A. Shamir. Zero-knowledge proofs of identity. *Journal of Cryptography*, 1: 66–94, 1988.

[For94] W. Ford. *Computer Communications Security — Principles, Standard Protocols and Techniques*, Prentice-Hall, New Jersey, 1994.

[FR95] P. Fahn and M.J.B. Robshaw. *Results from the RSA Factoring Challenge*. Technical Report TR-501, version 1.3, RSA Laboratories, January 1995.

[FS87] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology — Crypto '86*, pages 186–194, Springer-Verlag, 1987.

[FY94] M. Franklin and M. Yung. Blind Weak Signature and its Applications: Putting Non-Cryptographic Secure Computation to Work. In *Advances in Cryptology — Eurocrypt '94*, pages 67–76, Springer-Verlag, 1994.

[Gan95] R. Ganesan. Yaksha: Augmenting Kerberos with public key cryptography. In *Proceedings of the 1995 Internet Society Symposium on Network and Distributed Systems Security*, pages 132–143, IEEE Press, 1995.

[GC89] D. Gollman and W.G. Chambers. Clock-controlled shift registers: a review. *IEEE Journal on Selected Areas in Communications*, 7(4): 525–533, May 1989.

[Gib93] J.K. Gibson. Severely denting the Babidulin version of the McElience public key cryptosystem. In *Prepoceedings of the 4th IMA Conference on Cryptography and Coding*, 1993.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28: 270–299, 1984.

[GM93] D.M. Gordon and K.S. McCurley. Massively parallel computation of discrete logarithms. In *Advances in Cryptology — Crypto '92*, pages 312–323, Springer-Verlag, 1993.

[GMR86] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen message attack. *SIAM Journal on Computing,* 17(2): 289–308, March 1988.

[Gor93] D.M. Gordon. Discrete logarithms in *GF(p)* using the number field sieve. *SIAM Journal of Computing*, 6(1): 124–138, February 1993.

[GPT91] E.M. Gabidulin, A.V. Paramonov, and O.V. Tretjakov. Ideals over a non-commutative ring and their application in cryptology. In *Advances in Cryptology – Eurocrypt '91*, pages 482–489, Springer-Verlag, 1991.

[GQ88] L.C. Guillou and J.J. Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In *Advances in Cryptology — Eurocrypt '88*, pages 123–128, Springer-Verlag, 1988.

[Has88] J. Hastad. Solving simultaneous modular equations of low degree. *SIAM Journal of Computing*, 17: 336–241, 1988.

[Hel80] M.E. Hellman. A cryptanalytic time-memory trade off. *IEEE Transactions on Information Theory*, IT-26: 401–406, 1980.

[Hic95] K.E.B. Hickman. *The SSL Protocol*. December 1995. `<http://www.netscape.com/newsref/std/ssl.html>`

[HKM95] C. Harpes, G.G. Kramer, and J.L. Massey. A generalization of linear cryptanalysis and the applicability of Matsui's piling-up lemma. In *Advances in Cryptology — Eurocrypt '95*, pages 24–38, Springer-Verlag, 1995.

[HS91] S. Haber and W.S. Stornetta. How to timestamp a digital document. *Journal of Cryptology*, 3(2): 99–111, 1991.

[IBM95] IBM, Netscape, GTE, CyberCash, and MasterCard. *Secure Electronic Payment Protocol (SEPP).* Draft, Version 1.2, November 3, 1995. `<http://www.mastercard.com/Sepp/sepptoc.html>`

[IEE95] IEEE Working Group P1363. Working Draft: IEEE 1363: Standard for RSA, Diffie-Hellman and Related Public-Key Cryptography. In preparation, 1995.

[ISO87] ISO DIS 8730. *Banking requirements for message authentication (wholesale).* 1987.

[ISO91] ISO/IEC 9979. *Data Cryptographic Techniques - Procedures for the Registration of Cryptographic Algorithms*. 1991.

[ISO92a] ISO/IEC 9798. *Entity authentication mechanisms using symmetric techniques*. 1992.

[ISO92b] ISO/IEC 10116. *Modes of operation for an n-bit block cipher algorithm.* 1992.

[ISO92c] ISO/IEC 10118. *Information technology - Security techniques - Hash functions*. 1992.

[JML93] D.B. Johnson, S.M. Matyas, A.V. Le, and J.D. Wilkins. Design of the commercial data masking facility data privacy algorithm. In *Proceedings of the 1st ACM Conference on Communications and Computer Security*, ACM Press, VA, 1993.

[Jue83] R.R. Jueneman. Analysis of certain aspects of output feedback mode. In *Advances in Cryptology: Proceedings of Crypto '82*, pages 99–127, Plenum Press, 1983.

[Kah67] D. Kahn. *The Codebreakers*. Macmillan Co., New York, 1967.

[Kal92] B.S. Kaliski Jr. *RFC 1319: The MD2 Message-Digest Algorithm*. RSA Laboratories, April 1992.

[Kal93a] B.S. Kaliski Jr. *RFC 1424: Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services*. RSA Laboratories, February 1993.

[Kal93b] B.S. Kaliski Jr. A survey of encryption standards. *IEEE Micro*, 13(6): 74–81, December 1993.

[Kal95] B.S. Kaliski Jr. A chosen message attack on Demytko's cryptosystem. Journal of Cryptology. To appear.

[Ken93] S. Kent. *RFC 1422: Privacy Enhancement for Internet Electronic Mail, Part II: Certificate-Based Key Management*. Internet Activities Board, February 1993.

[KMS95] P. Karn, P. Metzger, and W. Simpson. *RFC 1829: The ESP DES-CBC Transform*. Qualcomm, Piermont, and Daydreamer, August 1995.

[KN93] J. Kohl and B. Neuman. *The Kerberos Network Authentication Service.* Network Working Group RFC 1510, 1993.

[KNT94] J. Kohl, B. Neuman, and T. Tso. The evolution of the Kerberos authentication service. *Distributed Open Systems*, IEEE Press, 1994.

[Knu81] D.E. Knuth. *The Art of Computer Programming*, volume 2, *Seminumerical Algorithms*. Addison-Wesley, 2nd edition, 1981.

[Knu93] L.R. Knudsen. Practically secure Feistel ciphers. In *Proceedings of 1st Workshop* on *Fast Software Encryption*, pages 211–221, Springer-Verlag, 1993.

[Knu95] L.R. Knudsen. A key-schedule weakness in SAFER K-64. In *Advances in Cryptology — Crypto '95*, pages 274–286, Springer-Verlag, 1995.

[KO95] K. Kurosawa and K. Okada. Low exponent attack against elliptic curve RSA. In *Advances in Cryptology — Asiacrypt '94*, pages 376–383, Springer-Verlag, 1995.

[Kob87] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48: 203–209, 1987.

[Kob94] N. Koblitz. *A Course in Number Theory and Cryptography*. Springer-Verlag, 1994.

[Koc94] Ç.K. Koç. *High-Speed RSA Implementation*. Technical Report TR-201, version 2.0, RSA Laboratories, November 1994.

[KR94] B.S. Kaliski Jr. and M.J.B. Robshaw. Linear cryptanalysis using multiple approximations. In *Advances in Cryptology — Crypto '94*, pages 26–39, Springer-Verlag, 1994.

[KR95a] B.S. Kaliski Jr. and M.J.B. Robshaw. Linear cryptanalysis using multiple approximations and FEAL. In *Proceedings of 2nd Workshop on Fast Software Encryption*, pages 249–264, Springer-Verlag, 1995.

[KR95b] B.S. Kaliski Jr. and M.J.B. Robshaw. Message authentication with MD5. *CryptoBytes,* 1(1): 5–8, 1995.

[KR95c] B.S. Kaliski Jr. and M.J.B. Robshaw. The secure use of RSA. *CryptoBytes,* 1(3): 7–13, 1995.

[KR96] B.S. Kaliski Jr. and M.J.B. Robshaw. *Multiple encryption: weighing up security and performance*. *Dr. Dobb's Journal*, #243, pages 123–127, January 1996.

[Kra93] D. Kravitz. Digital signature algorithm. U.S. Patent #5,231,668, July 27, 1993.

[KRS88] B.S. Kaliski Jr., R.L. Rivest, and A.T. Sherman. Is the data encryption standard a group? *Journal of Cryptology*, 1: 3–36, 1988.

[KT91] V.I. Korzhik and A.I. Turkin. Cryptanalysis of McEliece's public-key cryptosystem. In *Advances in Cryptology — Eurocrypt '91*, pages 68–70, Springer-Verlag, 1991.

[KY95] B.S. Kaliski Jr. and Y.L. Yin. On differential and linear cryptanalysis of the RC5 encryption algorithm. In *Advances in Cryptology — Crypto '95*, pages 171–183, Springer-Verlag, 1995.

[Lan88] S. Landau. Zero knowledge and the Department of Defense. *Notices of the American Mathematical Society*, 35: 5–12, 1988.

[Len87] H.W. Lenstra Jr. Factoring integers with elliptic curves. *Annuals of Mathematics.*, 126: 649–673, 1987.

[LH94] S.K. Langford and M.E. Hellman. Differential-linear cryptanalysis. In *Advances in Cryptology — Crypto '94*, pages 17–25, Springer-Verlag, 1994.

[Lin93] J. Linn. *RFC 1508: Generic Security Services Application Programming Interface*. Geer Zolot Associates, September 1993.

[Lip94] R.J. Lipton. Speeding up computations via molecular biology. Princeton University, draft, December 1994.

[LL90] A.K. Lenstra and H.W. Lenstra Jr. Algorithms in number theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 673–715, MIT Press/Elsevier, Amsterdam, 1990.

[LLM93] A.K. Lenstra, H.W. Lenstra Jr., M.S. Manasse, and J.M. Pollard. The factorization of the ninth Fermat number. *Mathematics of Computation*, 61(203): 319–349, 1993.

[LM91a] X. Lai and J.L. Massey. A proposal for a new block encryption standard. In *Advances in Cryptology — Eurocrypt '90*, pages 389–404, Springer-Verlag, 1991.

[LM91b] A.K. Lenstra and M.S. Manasse. Factoring with two large primes. In *Advances in Cryptology — Eurocrypt '90*, pages 72–82, Springer-Verlag, 1991.

[LMM92] X. Lai, J.L. Massey and S. Murphy. Markov ciphers and differential cryptanalysis. In *Advances in Cryptology — Eurocrypt '91*, pages 17–38, Springer-Verlag, 1992.

[LO91] B.A. LaMacchia and A.M. Odlyzko. Computation of discrete logarithms in prime fields. *Designs, Codes and Cryptography*, 1: 47–62, 1991.

[LRW92] X. Lai, R.A. Rueppel, and J. Woollven. A fast cryptographic checksum algorithm based on stream ciphers. In *Advances in Cryptology — Auscrypt '92*, Springer-Verlag, 1992.

[Mas93] J.L. Massey. SAFER K-64: A byte-oriented block ciphering algorithm. In *Proceedings of 1st Workshop on Fast Software Encryption*, pages 1–17, Springer-Verlag, 1993.

[Mas95] J.L. Massey. SAFER K-64: One year later. In *Proceedings of 2nd Workshop on Fast Software Encryption*, pages 212–241, Springer-Verlag, 1995.

[Mat93] M. Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology — Eurocrypt '93*, pages 386–397, Springer-Verlag, 1993.

[Mat94] M. Matsui. The first experimental cryptanalysis of the data encryption standard. In *Advances in Cryptology — Crypto '94*, pages 1–11, Springer-Verlag, 1994.

[Mat96] T. Matthews. Suggestions for random number generation in software. Bulletin No. 1, RSA Laboratories, January 1996.

[Mau94] U. Maurer. Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms. In *Advances in Cryptology — Crypto '94*, pages 271–281, Springer-Verlag, 1994.

[Mce78] R.J. McEliece. A public-key cryptosystem based on algebraic coding theory. *JPL DSN Progress Report 42–44*, pages 114–116, 1978.

[Mcn95] F.L. McNulty. Clipper – Alive and well as a voluntary government standard for telecommunications. *The 1995 RSA Data Security Conference*, January 1995.

[Men93] A. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.

[Mer79] R.C. Merkle. Secrecy, authentication and public-key systems. Ph. D. Thesis, Stanford University, 1979.

[Mer90a] R.C. Merkle. One way hash functions and DES. In *Advances in Cryptology — Crypto '89*, pages 428–446, Springer-Verlag, 1990.

[Mer90b] R.C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology — Crypto '89*, pages 428–446, Springer-Verlag, 1990.

[Mer91] R.C. Merkle. Fast software encryption functions. In *Advances in Cryptology — Crypto '90*, pages 627–638, Springer-Verlag, 1991.

[MH78] R.C. Merkle and M.E. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, IT-24: 525–530, 1978.

[MH81] R.C. Merkle and M.E. Hellman. On the security of multiple encryption. *Communications of the ACM*, 24: 465–467, July 1981.

[Mic93] S. Micali. Fair public-key cryptosystems. In *Advances in Cryptology — Crypto '92*, pages 113–138, Springer-Verlag, 1993.

[Mic95] Microsoft Corporation. *STT Wire Formats and Protocols*. Version 0.902, Redmond, WA, October 5, 1995. <http://www.microsoft.com/windows/ie/STT.htm>

[Mil86] V.S. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology — Crypto '85*, pages 417–426, Springer-Verlag, 1986.

[MOV90] A. Menezes, T. Okamoto, and S. Vanstone. *Reducing elliptic curve logarithms to logarithms in a finite field.* Unpublished manuscript, September 1990.

[MQV95] A. Menezes, M. Qu, and S. Vanstone. Some new key agreement protocols providing implicit authentication. In *Preproceedings of Workshops on Selected Areas in Cryptography*, 1995.

[MS95] P. Metzger and W. Simpson. *RFC 1828: IP Authentication using Keyed MD5.* Piermont and Daydreamer, August 1995.

[MS95] W. Meier and O. Staffelbach. The self-shrinking generator. In *Advances in Cryptology — Eurocrypt '94*, pages 205–214, Springer-Verlag, 1995.

[Mur90] S. Murphy. The cryptanalysis of FEAL-4 with 20 chosen plaintexts. *Journal of Cryptology*, 2(3): 145–154, 1990.

[MY92] M. Matsui and A. Yamagishi. A new method for known plaintext attack of FEAL cipher. In *Advances in Cryptology — Eurocrypt '92*, pages 81–91, Springer-Verlag, 1992.

[NIS80] National Institute of Standards and Technology (NIST). *FIPS Publication 81: DES Modes of Operation*. December 2, 1980. Originally issued by National Bureau of Standards.

[NIS85] National Institute of Standards and Technology (NIST). *FIPS Publication 113: Computer Data Authentication*. 1985.

[NIS92] National Institute of Standards and Technology (NIST). The Digital Signature Standard, proposal and discussion. *Communications of the ACM*, 35(7): 36–54, July 1992.

[NIS93a] National Institute of Standards and Technology (NIST). *FIPS Publication 180: Secure Hash Standard (SHS)*. May 1993.

[NIS93b] National Institute of Standards and Technology (NIST). *FIPS Publication 46-2: Data Encryption Standard*. December 1993.

[NIS94a] National Institute of Standards and Technology (NIST). *FIPS Publication 185: Escrowed Encryption Standard*. February 1994.

[NIS94b] National Institute of Standards and Technology (NIST). *FIPS Publication 186: Digital Signature Standard (DSS)*. May 1994.

[NIS94c] National Institute of Standards and Technology (NIST). *Announcement of Weakness in the Secure Hash Standard*. May 1994.

[NK95] K. Nyberg and L.R. Knudsen. Provable security against a differential attack. *Journal of Cryptology*, 8(1): 27–37, 1995.

[NMR94] D. Naccache, D. M'raïhi, D. Raphaeli, and S. Vaudenay. Can D.S.A. be improved? Complexity trade-offs with the Digital Signature Standard. In *Advances in Cryptology — Eurocrypt '94*, pages 77–85, Springer-Verlag, 1994.

[NS78] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21: 993–999, 1978.

[NS94] M. Naor and A. Shamir. Visual cryptography. In *Advances in Cryptology — Eurocrypt '94*, pages 1–12, Springer-Verlag, 1994.

[NSA95] NSA Cross Organization CAPI Team. *Security Service API: Cryptographic API Recommendation*, 1995.

[Nyb95] K. Nyberg. Linear approximation of block ciphers. In *Advances in Cryptology — Eurocrypt '94* (rump session), pages 439–44, Springer-Verlag, 1995.

[OA94] K. Ohta and K. Aoki. Linear cryptanalysis of the fast data encipherment algorithm. *In Advances in Cryptology — Crypto '94*, pages 12–16, Springer-Verlag, 1994.

[Oco95] L. O'Connor. A unified markov approach to differential and linear cryptanalysis. In *Advances in Cryptology — Asiacrypt '94*, pages 387–397, Springer-Verlag, 1995.

[Odl84] A.M. Odlyzko. Discrete logarithms in finite fields and their cryptographic significance. In *Advances in Cryptology — Eurocrypt '84*, pages 224–314, Springer-Verlag, 1984.

[Odl95] A.M. Odlyzko. The future of integer factorization. *CryptoBytes,* 1(2): 5–12, 1995.

[Oka93] T. Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *Advances in Cryptology — Crypto '92*, pages 31–53, Springer-Verlag, 1993.

[OPS93] Office of the Press Secretary. *Statement*. The White House, April 16, 1993.

[Pol74] J. Pollard. Theorems of factorization and primality testing. *Proceedings of Cambridge Philosophical Society*, 76: 521–528, 1974.

[Pol75] J. Pollard. Monte Carlo method for factorization. *BIT*, 15: 331–334, 1975.

[Pre93] B. Preneel. *Analysis and Design of Cryptographic Hash Functions*. Ph.D. Thesis, Katholieke University Leuven, 1993.

[Pre94] B. Preneel. The State of DES. *1994 RSA Laboratories Seminar Series*, August 1994.

[QG90] J.J. Quisquater and L. Guillou. How to explain zero-knowledge protocols to your children. In *Advances in Cryptology — Crypto '89*, pages 628–631, Springer-Verlag, 1990.

[Rab79] M.O. Rabin. *Digitalized signatures and public-key functions as intractable as factorization.* Technical Report MIT/LCS/TR-212, MIT, 1979.

[RC93] P. Rogaway and D. Coppersmith. A software-optimized encryption algorithm. In *Proceedings of 1st Workshop on Fast Software Encryption*, pages 56–63, Springer-Verlag, 1993.

[RC95] N. Rogier and P. Chauvaud. The compression function of MD2 is not collision free. Presented at *Selected Areas in Cryptography '95*, Ottawa, Canada, May 18–19, 1995.

[RG91] D. Russell and G.T. Gangemi Sr. *Computer Security Basics*. O'Reilly & Associates, Inc., 1991.

[Riv90] R.L. Rivest. Cryptography. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 719–755, MIT Press/Elsevier, Amsterdam, 1990.

[Riv91a] R.L. Rivest. Finding four million random primes. In *Advances in Cryptology — Crypto '90*, pages 625–626, Springer-Verlag, 1991.

[Riv91b] R.L. Rivest. The MD4 message digest algorithm. In *Advances in Cryptology — Crypto '90*, pages 303–311, Springer-Verlag, 1991.

[Riv92a] R.L. Rivest. Response to NIST's proposal. *Communications of the ACM*, 35: 41–47, July 1992.

[Riv92b] R.L. Rivest. *RFC 1320: The MD4 Message-Digest Algorithm*. Network Working Group, April 1992.

[Riv92c] R.L. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm*. Internet Activities Board, April 1992.

[Riv95] R.L. Rivest. The RC5 encryption algorithm. *CryptoBytes,* 1(1): 9–11, 1995.

[Rob95a] M.J.B. Robshaw. *Block Ciphers*. Technical Report TR-601, version 2.0, RSA Laboratories, August 1995.

[Rob95b] M.J.B. Robshaw. *Stream Ciphers*. Technical Report TR-701, version 2.0, RSA Laboratories, July 1995.

[Rob95c] M.J.B. Robshaw. *MD2, MD4, MD5, SHA and Other Hash Functions.* Technical Report TR-101, version 4.0, RSA Laboratories, July 1995.

[Rob95d] M.J.B. Robshaw. *Security estimates for 512-bit RSA*. Technical Note, RSA Laboratories, June 1995.

[RS95] E. Rescorla and A. Schiffman. *The Secure HyperText Transfer Protocol*. Internet-Draft, EIT, July 1995.

[RSA78] R.L. Rivest, A. Shamir, and L.M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2): 120–126, February 1978.

[RSA95] RSA Laboratories. *PKCS #11: Cryptographic Token Interface Standard*. Version 1.0, April 1995.

[Rue92] R.A. Rueppel. Stream ciphers. In *Contemporary Cryptology — The Science of Information Integrity*. IEEE Press, 1992.

[SB93] M.E. Smid and D.K. Branstad. Response to comments on the NIST proposed Digital Signature Standard. In *Advances in Cryptology — Crypto '92*, pages 76–87, Springer-Verlag, 1993.

[Sch83] I. Schaumuller-Bichl. Cryptanalysis of the Data Encryption Standard by a method of formal coding. *Cryptography, Proc. Burg Feuerstein 1982*, 149: 235–255, Berlin,1983.

[Sch90] C.P. Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology — Crypto '89*, pages 239–251, Springer-Verlag, 1990.

[Sch91] C.P. Schnorr. Method for identifying subscribers and for generating and verifying electronic signatures in a data exchange system. U.S. Patent #4,995,082, February 19, 1991.

[Sch93] B. Schneier. Description of a new variable-length key, 64-bit block cipher (Blowfish). In *Proceedings of 1st Workshop on Fast Software Encryption*, pages 191–204, Springer-Verlag, 1993.

[Sch95a] B. Schneier. The Blowfish encryption algorithm: one year later. *Dr. Dobb's Journal*, No. 234, pages 137–138, September 1995.

[Sch95b] B. Schneier. *Applied Cryptography*: *Protocols, Algorithms, and Source Code in C*. Wiley, 2nd Edition, 1995.

[SH95] C.P. Schnorr and H.H. Hörner. Attacking the Chor-Rivest cryptosystem by improved lattice reduction. In *Advances in Cryptology — Eurocrypt '95*, pages 1–12, Springer-Verlag, 1995.

[Sha49] C.E. Shannon. Communication Theory of Secrecy Systems. *Bell Systems Technical Journal*, 28: 656–715, October 1949.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22: 612–613, 1979.

[Sha84] A. Shamir. A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem. *IEEE Transactions on Information Theory*, IT-30(5): 699–704, September 1984.

[Sha95] M. Shand. Personal communication. 1995.

[Sho94] P.W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual IEEE Symposium on the Foundations of Computer Science*, pages 124–134, 1994.

[Sil87] R.D. Silverman. The multiple polynomial quadratic sieve. *Mathematics of Computation*, 48: 329–339, 1987.

[Sim92] G.J. Simmons, editor. *Contemporary Cryptology — The Science of Information Integrity*. IEEE Press, 1992.

[SM88] A. Shimizu and S. Miyaguchi. Fast data encipherment algorithm FEAL. In *Advances in Cryptology — Eurocrypt '87*, pages 267–280, Springer-Verlag, 1988.

[SPC95] M. Stadler, J.M. Piveteau, and J. Carmenisch. Fair blind signatures. In *Advances in Cryptology — Eurocrypt '95*, pages 209–219, Springer-Verlag, 1995.

[SS95] P. Smith and C. Skinner. A public-key cryptosystem and a digital signature system based on the Lucas function analogue to discrete logarithms. In *Advances in Cryptology — Asiacrypt '94*, pages 357–364, Springer-Verlag, 1995.

[Sta95] W. Stallings. *Network and Internetwork Security – Principles and Practice*. Prentice-Hall, New Jersey, 1995.

[Sti95] D.R. Stinson. *Cryptography — Theory and Practice*. CRC Press, Boca Raton, 1995.

[SV93] M. Shand and J. Vuillemin. Fast implementations of RSA cryptography. In *Proceedings of the 11th IEEE Symposium on Computer Arithmetic*, pages 252–259, IEEE Computer Society Press, 1993.

[Ver26] G.S. Vernam. Cipher printing telegraph systems for secret wire and radio telegraphic communications. *J. Amer. Inst. Elec. Eng.*, vol. 45, pages 109–115, 1926.

[Vis95] Visa International. *Secure Transaction Technology Specifications*, Version 1.0, September 26, 1995. (`http://www.visa.com/visa-stt/`)

[VP92] E. van Heyst and T.P. Pederson. How to make efficient fail-stop signatures. In *Advances in Cryptology — Eurocrypt '92*, pages 366–377, Springer-Verlag, 1992.

[VW91] P. van Oorschot and M. Wiener. A known plaintext attack on two-key triple encryption. In *Advances in Cryptology — Eurocrypt '90*, pages 318–325, Springer-Verlag, 1991.

[VW94] P. van Oorschot and M. Wiener. Parallel collision search with application to hash functions and discrete logarithms. In *Proceedings of 2nd ACM Conference on Computer and Communication Security*, 1994.

[Wie94] M.J. Wiener. Efficient DES key search. Technical Report TR–244, School of Computer Science, Carleton University, Ottawa, Canada, May 1994.

[Xop95] X/Open Company Ltd. *Generic Cryptographic Service API (GCS-API).* Base – Draft 3, April 1995.

[Yuv79] G. Yuval. How to swindle Rabin. *Cryptologia*, July 1979.

[ZPS93] Y. Zheng, J. Pieprzyk and J. Seberry. HAVAL - a one-way hashing algorithm with variable length output. In *Advances in Cryptology — Auscrypt '92*, pages 83–104, Springer-Verlag, 1993.