RSA Laboratories

FAQ 4.0

FREQUENTLY ASKED QUESTIONS ABOUT TODAY'S CRYPTOGRAPHY

RSA Laboratories

A Division of RSA Data Security

Table of Contents

Foreword

This document is the fourth version of *RSA Laboratories' Frequently Asked Questions About Today's Cryptography*. Completely revised, restructured, and updated, it covers many more questions than the previous version, which was published in September 1995.

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, to Sambasivam Valliappan, who drafted the third version as an RSA Laboratories research assistant in Summer 1995.

Other contributors to the fourth version include Jim Bidzos, John Brainard, Victor Chang, Scott Contini, Dana Ellingen, James Gray, Ari Juels, Burt Kaliski, Patrick Lee, John Linn, Paul Livesay, Hoa Ly, Tim Matthews, Matthew Robshaw, Ray Sidney, Bob Silverman, Jessica Staddon, Jeff Stapleton, Kurt Stammberger, and Yiqun Lisa Yin from RSA Data Security, Michael Baum from VeriSign, Mathew Butler, Stuart Haber from Bellcore, Bart Preneel from Katholieke Universiteit, and Scott Stornetta from Surety Technologies.

Special thanks to Patrick Lee, Eliza Sachs, Michael Girdley, Jason Gillis, and Clint Chan for carefully reading through the final draft and providing many helpful suggestions. Special thanks also to Jason Thompson and Brandon Richards for converting the FAQ to the web. Finally, special thanks to Jeff Ylvisaker, who helped with the editing throughout.

The technical editors are RSA Laboratories research assistants Moses Liskov and Beverly Schmoock.

Comments on the FAQ are encouraged. Address correspondence to:

FAQ Editor
RSA Laboratories
2955 Campus Drive, Suite 400
San Mateo, CA 94403-2507 USA
Tel: 650-295-7600
Fax: 650-295-7700

Section 1: Introduction

Question 1.1. *What is the RSA Laboratories' Frequently Asked Questions About Today's Cryptography*?

The *RSA Laboratories' Frequently Asked Questions About Today's Cryptography* is a large collection of questions about modern cryptography, cryptanalysis, and issues related to them. The information is presented in question and answer form. Version 4.0 of the FAQ is markedly different from previous versions, primarily in the general structure of the document.

Section 1: Introduction.
This section is intended for those who are interested in learning about cryptography but don't know very much about it. The introduction gives a brief overview of the field of cryptography and related issues.

Section 2: Cryptography.
This section expands on the overview of the field of cryptography and related issues, providing more detail about the concepts involved in cryptography. It lays the conceptual groundwork for the next section.

Section 3: Techniques in Cryptography.
Cryptographic algorithms are the basic building blocks of cryptographic applications and protocols. This section presents most of the important encryption algorithms, hash functions, stream ciphers, and other basic cryptographic algorithms.

Section 4: Applications of Cryptography.
This section is an overview of the most important protocols and systems made possible by cryptography. In particular, it discusses the issues involved in establishing a cryptographic infrastructure, and it gives a brief overview of some of the electronic commerce techniques available today.

Section 5: Cryptography in the Real World.
This section goes over some of the most important cryptographic systems in place around the world today, including secure Internet communications, and some of the more popular cryptographic products. It also gives an overview of the major groups of cryptographic standards.

Section 6: Laws Concerning Cryptography.
This section deals with the legal and political issues associated with cryptography, including government involvement, patent issues, and import and export regulations. Note that while this section includes legal information, it should not be used as a substitute for consulting an attorney.

Section 7: Miscellaneous Topics.
This section contains additional applications of cryptography, including new technologies and techniques as well as some of the more important older techniques and applications.

Section 8: Further Reading.
This section lists suggestions for further information about cryptography and related issues.

Glossary.
Contains a brief definition of most of the terms in this document.

Note: We have not attempted to be, nor could we be, exhaustive in answering every possible question. We hope that this document will be both a useful introductory text and a useful reference for those interested in the field of cryptography.

Question 1.2. What is cryptography?

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. Cryptanalysis is the study of how to compromise (defeat) cryptographic mechanisms, and cryptology (from the Greek *kryptós lógos*, meaning "hidden word") is the discipline of cryptography and cryptanalysis combined. To most people, cryptography is concerned with keeping communications private. Indeed, the protection of sensitive communications has been the emphasis of cryptography throughout much of its history [Kah67]. However, this is only one part of today's cryptography.

Encryption is the transformation of data into a form that is as close to impossible as possible to read with out the appropriate knowledge (a key). Its purpose is to ensure privacy by keeping information hidden from anyone for whom it is not intended, even those who have access to the encrypted data. Decryption is the reverse of encryption; it is the transformation of encrypted data back into an intelligible form.

Encryption and decryption generally require the use of some secret information, referred to as a key. For some encryption mechanisms, the same key is used for both encryption and decryption; for other mechanisms, the keys used for encryption and decryption are different (see Question 2.1.1).

Today's cryptography is more than encryption and decryption. Authentication is as fundamentally a part of our lives as privacy. We use authentication throughout our everyday lives -when we sign our name to some document for instance and, as we move to a world where our decisions and agreements are communicated electronically, we need to have electronic techniques for providing authentication.

Cryptography provides mechanisms for such procedures. A digital signature (see Question 2.2.2) binds a document to the possessor of a particular key, while a digital timestamp (see Question 7.11) 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 a pay-per-view TV channel.

The field of cryptography encompasses other uses as well. With just a few basic cryptographic tools, it is possible to build elaborate schemes and protocols that allow us to pay using electronic money (see Question 4.2.1), to prove we know certain information without revealing the information itself (see Question 2.1.8), and to share a secret quantity in such a way that a subset of the shares can reconstruct the secret (see Question 2.1.9).

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. The problem may also be hard because it is intrinsically difficult to complete, such as finding a message that produces a given hash value.

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 [Sch96b]; Ford [For94] provides detailed coverage of issues such as cryptography standards and secure communication.

Question 1.3. What are some of the more popular techniques in cryptography?

There are two types of cryptosystems: secret key and public key (see Questions 2.1.2 and 2.1.1). In secret-key cryptography, also referred to as symmetric cryptography, the same key is used for both encryption and decryption. The most popular secret-key cryptosystem in use today is known as DES, the Data Encryption Standard. IBM developed DES in the middle 1970’s and it has been a Federal Standard ever since 1976.

In public-key cryptography, each user has a public key and a private key. The public key is made public while the private key remains secret. Encryption is performed with the public key while decryption is done with the private key. The RSA public-key cryptosystem is the most popular form of public-key cryptography. RSA stands for Rivest, Shamir, and Adleman, the inventors of the RSA cryptosystem.

The Digital Signature Algorithm (DSA) is also a popular public-key technique, though it can only be used only for signatures, not encryption. Elliptic curve cryptosystems (ECCs) are cryptosystems based on mathematical objects known as Elliptic Curves (see Question 2.3.10.). Elliptic curve cryptography has been gaining in popularity recently. Lastly, the Diffie-Hellman key agreement protocol is a popular public-key technique for establishing secret keys over an insecure channel.

Question 1.4. How is cryptography applied?

Cryptography is extremely useful; there is a multitude of applications, many of which are currently in use. A typical application of cryptography is a system built out of the basic techniques. Such systems can be of various levels of complexity. Some of the more simple applications are secure communication, identification, authentication, and secret sharing. More complicated applications include systems for electronic commerce, certification, secure electronic mail, key recovery, and secure computer access. In general, the less complex the application, the more quickly it becomes a reality. Identification and authentication schemes exist widely, while electronic commerce systems are just beginning to be established.

Secure Communication

Secure communication is the most straightforward use of cryptography. Two people may communicate securely by encrypting the messages sent between them. This can be done in such a way that a third party eavesdropping may never be able to decipher the messages. While secure communication has existed for centuries, the key management problem has prevented it from becoming commonplace. Thanks to the development of public-key cryptography, the tools exist to create a large-scale network of people who can communicate securely with one another even if they had never communicated before.

Identification and Authentication

Identification and authentication are two widely used applications of cryptography. Identification is the process of verifying someone's or something's identity. For example, when withdrawing money from a bank, a teller asks to see identification (*e.g.* a driver's license) to verify the identity of the owner of the account. This same process can be done electronically using cryptography. Every automatic teller machine (ATM) card is associated with a "secret" personal identification number (PIN), which binds the owner to the card and thus to the account. When the card is inserted into the ATM, the machine prompts the cardholder for the PIN. If the correct PIN is entered, the machine identifies that person as the rightful owner and grants access. Another important application of cryptography is authentication. Authentication is similar to identification, in that both allow an entity access to resources (such as an Internet account), but authentication is broader because it does not necessarily involve identifying a person or entity. Authentication merely determines whether that person or entity is authorized for whatever is in question. For more information on authentication and identification, see Question 2.2.5.

Secret Sharing

Another application of cryptography, called secret sharing, allows the trust of a secret to be distributed among a group of people. For example, in a (K, N)-threshold scheme, information about a secret is distributed in such a way that any K out of the N people (K<N) have enough information to determine the secret, but any set of K-1 people do not. In any secret sharing scheme, there are designated sets of people whose cumulative information suffices to determine the secret. In some implementations of secret sharing schemes, each participant receives the secret after it has been generated. In other implementations, the actual secret is never made visible to the participants, although the purpose for which they sought the secret (*e.g.* access to a building or permission to execute a process) is allowed. See Question 2.1.9 for more information on secret sharing.

Electronic Commerce

Over the past few years there has been a growing amount of business conducted over the Internet - this form of business is called electronic commerce or *e*-commerce. *E*-commerce is comprised of online banking, online brokerage accounts, and Internet shopping, to name a few of the many applications. One can book plane tickets, make hotel reservations, rent a car, transfer money from one account to another, buy compact disks (CDs), clothes, books and so on all while sitting in front of a computer. However, simply entering a credit card number on the Internet leaves one open to fraud. One cryptographic solution to this problem is to encrypt the credit card number (or other private information) when it is entered on-line, another is to secure the entire session (see Question 5.1.2.) When a computer encrypts this information and sends it out on the Internet, it is incomprehensible to a third party viewer. The web-server ("Internet shopping center") receives the encrypted information, decrypts it, and proceeds with the sale without fear that the credit card number (or other personal information) slipped into the wrong hands. As more and more business is conducted over the Internet, the need for protection against fraud, theft and corruption of vital information increases.

Certification

Another application of cryptography is certification; certification is a scheme by which trusted agents such as certifying authorities vouch for unknown agents, such as users. The trusted agents issue vouchers called certificates which each have some inherent meaning. Certification technology was developed to make identification and authentication possible on a large scale. See Question 4.1.3.10 for more information on certification.

Key Recovery
Key recovery is a technology that allows a key to be revealed under certain circumstances without the owner of the key revealing it. This is useful for two main reasons: first of all, if a user loses or accidentally deletes their key, key recovery could prevent a disaster. Secondly, if a law enforcement agency wishes to eavesdrop on a suspected criminal without their knowledge (akin to a wiretap), they must be able to recover the key. Key recovery techniques are in use in some instances; however, the use of key recovery as a law enforcement technique is somewhat controversial. See Question 7.12 for more on key recovery.

Remote Access
Secure remote access is another important application of cryptography. The basic system of passwords certainly gives a level of security for secure access, but it may not be enough in some cases. For instance, passwords can be eavesdropped, forgotten, stolen, or guessed. Many products supply cryptographic methods for remote access with a higher degree of security.

Other Applications
Cryptography is not confined to the world of computers. Cryptography is also used in cellular phones as a means of authentication; that is, it can be used to verify that a particular phone has the right to bill to a particular phone number. This prevents people from stealing ("cloning") cellular phone numbers and access codes.

Question 1.5. What are cryptography standards?

Cryptography standards are needed to create interoperability in the information security world. Essentially they are conditions and protocols set forth to allow uniformity within communication, transactions and virtually all computer activity. The continual evolution of information technology motivates the development of more standards, which in turn helps guide this evolution.

The main motivation behind standards is to allow technology from different manufacturers to "speak the same language", that is, to interact effectively. Perhaps this is best seen in the familiar standard VHS for VCRs. A few years ago there were two competing standards in the VCR industry, VHS and BETA. A VHS tape could not be played in a BETA machine and vice versa; they were incompatible formats. Imagine the chaos if every VCR manufacturer had their own format. People could only rent movies that were available on the format compatible with their VCR. Standards are necessary to insure that products from different companies are compatible.

In cryptography, standardization serves an additional purpose; it can serve as a proving ground for cryptographic techniques because complex protocols are prone to design flaws. By establishing a well-examined standard, the industry can produce a trustworthier product. Even a safe protocol is more trusted by customers after it becomes a standard, because of the ratification process involved.

The government, private industry, and other organizations contribute to the vast collection of standards on cryptography. A few of these are ISO, ANSI, IEEE, NIST, and IETF (see Section 5.3). There are many types of standards, some used within the banking industry, some internationally and others within the government. Standardization helps developers design new products. Instead of spending time developing a new standard they can follow a pre-existing standard throughout the development process. With this process in place consumers have the chance to choose among competing products or services.

Question 1.6. What is the role of the United States government in cryptography?

The U.S. government plays many roles in cryptography, ranging from use to export control to standardization efforts to the development of new cryptosystems. Recently the government has taken an even bigger interest in cryptography due to its ever-increasing use outside of the military. The U.S. government plays a much larger role in cryptography than any other government in the world.

One obvious reason the U.S. government is interested in cryptography stems from the crucial role of secure communication during wartime. Because the enemy may have access to the communication medium, messages must be encrypted. With certain cryptosystems, the receiver can determine whether or not the message was tampered with during transmission, and whether the message really came from who claims to have sent it.

In the past, the government has not only used cryptography itself, but has cracked other country's codes as well. A notable example of this occurred in 1940 when a group of Navy cryptanalysts, led by William *F*. Friedman, succeeded in breaking the Japanese diplomatic cipher known as Purple.

In 1952, the U.S. government established the NSA, The National Security Agency (see Question 6.2.2), whose job is to handle military and government data security as well as gather information about other countries' communications. Also established was NIST, The National Institute of Standards and Technology (see Question 6.2.1), which plays a major role in developing cryptography standards.

During the 1970's, IBM and the U.S. Department of Commerce - more precisely NIST (then known as NBS, The National Bureau of Standards) - developed along with NSA the Data Encryption Standard, DES. This algorithm has been a standard since 1977, with reviews leading to renewals every few years. The general consensus is that DES will not be strong enough for the future's encryption needs. During the next few years, NIST will be working on a new standard, AES, the Advanced Encryption Standard (see Question 3.3.1), to replace DES. It is expected that AES will remain a standard well into the 21st century.

Currently there are no restrictions on the use or strength of domestic encryption (encryption where the sender and recipient are in the U.S.). However, the government regulates the export of cryptography from the U.S. by setting restrictions (see Section 6.4) on how strong such encryption may be. Cryptography is dealt with as an item in the International Trade in Arms (ITAR) bill, and is considered for those purposes to be munitions.

Question 1.7. Why is cryptography important?

Cryptography allows people to carry over the confidence found in the physical world to the electronic world, thus allowing people to do business electronically without worries of deceit and deception. Every day hundreds of thousands of people interact electronically, whether it is through *e*-mail, *e*-commerce (business conducted over the Internet), ATM machines, or cellular phones. The perpetual increase of information transmitted electronically has lead to an increased reliance on cryptography.

Cryptography on the Internet

The Internet, comprised of millions of interconnected computers, allows nearly instantaneous communication and transfer of information, around the world. People use *e*-mail to correspond with one another. The World Wide Web is used for online business, data distribution, marketing, research, learning, and a myriad of other activities.

Cryptography makes secure websites (see Question 5.1.2) and electronic safe transmissions possible. For a website to be secure all of the data transmitted between the computers where the data is kept and where it is received must be encrypted. This allows people to do online banking, online trading, and make online purchases with their credit cards, without worrying that any of their account information is being compromised. Cryptography is very important to the continued growth of the Internet and electronic commerce.

E-commerce (see Section 4.2) is increasing at a very rapid rate. By the turn of the century, commercial transactions on the Internet are expected to total hundreds of billions of dollars a year. This level of activity could not be supported without cryptographic security. It has been said that one is safer using a credit card over the Internet than within a store or restaurant. It requires much more work to seize credit card numbers over computer networks than it does to simply walk by a table in a restaurant and lay hold of a credit card receipt. These levels of security, though not yet widely used, give the means to strengthen the foundation with which *e*-commerce can grow.

People use *e*-mail to conduct personal and business matters on a daily basis. *E*-mail has no physical form and may exist electronically in more than one place at a time. This poses a potential problem as it increases the opportunity for an eavesdropper to get a hold of the transmission. Encryption protects *e*-mail by rendering it very difficult to read by any unintended party. Digital signatures can also be used to authenticate the origin and the content of an *e*-mail message.

Authentication

In some cases cryptography allows you to have more confidence in your electronic transactions than you do in real life transactions. For example, signing documents in real life still leaves one vulnerable to the following scenario. After signing your will, agreeing to what is put forth in the document, someone can change that document and your signature is still attached. In the electronic world this type of falsification is much more difficult because digital signatures (see Question 2.2.2) are built using the contents of the document being signed.

Access Control

Cryptography is also used to regulate access to satellite and cable TV. Cable TV is set up so people can watch only the channels they pay for. Since there is a direct line from the cable company to each individual subscriber's home, the Cable Company will only send those channels that are paid for. Many companies offer pay-per-view channels to their subscribers. Pay-per-view cable allows cable subscribers to "rent" a movie directly through the cable box. What the cable box does is decode the incoming movie, but not until the movie has been "rented." If a person wants to watch a pay-per-view movie, he/she calls the cable company and requests it. In return, the Cable Company sends out a signal to the subscriber's cable box which unscrambles (decrypts) the requested movie.

Satellite TV works slightly differently since the satellite TV companies do not have a direct connection to each individual subscriber's home. This means that anyone with a satellite dish can pick up the signals.
To alleviate the problem of people getting free TV, they use cryptography. The trick is to allow only those who have paid for their service to unscramble the transmission, this is done with receivers (unscramblers). Each subscriber is given a receiver; the satellite transmits signals that can only be unscrambled by their receivers (ideally). Pay-per-view works in essentially the same way as it does for regular cable TV.

As seen, cryptography is widely used. Not only is it used over the Internet, but also it is used in phones, televisions, and a variety of other common household items. Without cryptography, hackers could get into our *e*-mail, listen in on our phone conversations, tap into our cable companies and acquire free cable service, or break into our bank/brokerage accounts.

Section 2: Cryptography

Section 2.1: Cryptographic Tools

Question 2.1.1. What is public-key cryptography?

In traditional cryptography, the sender and receiver of a message know and use 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 challenge 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, a phone system, or some other transmission medium to prevent the disclosure of the secret key. 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.

In order to solve the key management problem, Whitfield Diffie and Martin Hellman [DH76] introduced the concept of public-key cryptography in 1976. Public-key cryptosystems have two primary uses, encryption and digital signatures. In their system, each person gets a pair of keys, one called the public key and the other called the private key. The 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. In this system, it is no longer necessary to trust the security of some means of communications. The only requirement is that public keys be 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) and other various techniques.

In a public-key cryptosystem, the private key is always linked mathematically to the public key. Therefore, it is always possible to attack a public key system by deriving the private key from the public key. Typically, the defense against this is to make the problem of deriving the private key from the public key as difficult as possible. For instance, some public-key cryptosystem are designed such that deriving the private key from the public key requires the attacker to factor a large number, it this case it is computationally infeasible to perform the derivation. This is the idea behind the RSA public-key cryptosystem.

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 (because only Bob knows Bob's private key).

Digital Signatures
To sign a message, Alice does a computation involving both her private key and the message itself. The output is called a digital signature and is attached to the message. To verify the signature, Bob does a computation involving the message, the purported signature, and Alice's public key. If the result is correct according to a simple, prescribed mathematical relation, the signature is verified to be genuine; otherwise, the signature is fraudulent, or the message may have been altered.

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

Question 2.1.2. What is secret-key cryptography?

Secret-key cryptography is sometimes referred to as symmetric cryptography. It is the more traditional form of cryptography, in which a single key can be used to encrypt and decrypt a message. Secret-key cryptography not only deals with encryption, but it also deals with authentication. One such technique is called message authentication codes, MACs (see Question 2.1.7).

The main problem with secret-key cryptosystems is getting the sender and receiver to agree on the secret key without anyone else finding out. This requires a method by which the two parties can communicate without fear of eavesdropping. However, the advantage of secret-key cryptography is that it is generally faster than public-key cryptography.

The most common techniques in secret-key cryptography are block ciphers (see Question 2.1.4), stream ciphers (see Question 2.1.5), and message authentication codes.

Question 2.1.3. 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 be 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) since the same key is used for encryption and decryption. A serious concern is that there may be a chance that an enemy can discover the secret key during transmission.

Another major advantage of public key systems is they can provide digital signatures that cannot be repudiated. 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 the shared secret was somehow compromised (see Question 4.1.2.3) by one of the parties sharing the secret. For example, the Kerberos secret key authentication system (see Question 5.1.6) 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 many 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. Such a protocol is called a digital envelope, which is explained in more detail in Question 2.2.4.

Public-key cryptography may be vulnerable to impersonation, even if users' private keys are not available. A successful attack on a certification authority (see Question 4.1.3.14) will allow an adversary to impersonate whomever he or she chooses 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. These include environments where secure secret key distribution 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 establishment in a secret key system [DH76]; this is still one of its primary functions. Secret-key cryptography remains extremely important and is the subject of much ongoing study and research. Some secret-key cryptosystems are discussed in the sections on block ciphers and stream ciphers.

Question 2.1.4. What is a block cipher?

A block cipher is a type of symmetric-key encryption algorithm that transforms a fixed-length block of *plaintext* (unencrypted text) data into a block of *ciphertext* (encrypted text) 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. In the coming years the block size will increase to 128 bits as processors become more sophisticated.

For those with a mathematical background: Since different plaintext blocks are mapped to different ciphertext blocks (to allow unique decryption), a block cipher effectively provides a permutation (one to one reversible correspondence) 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]. The remainder of this answer deals with the structure of typical block ciphers and their modes of operation, and gets rather technical.

Iterated block ciphers encrypt a plaintext block by a process that has several rounds. In each round, the same transformation (also known as a round function) is applied to the data using a subkey. The set of subkeys is usually derived from the user-provided secret key by a special function. The set of subkeys is called the *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.

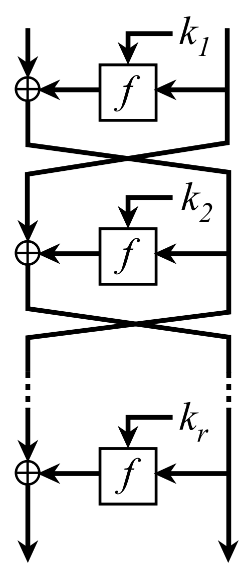


Figure 1. Feistel Cipher

Feistel ciphers [Fei73] are a special class of iterated block ciphers where the ciphertext is calculated from the plaintext by repeated application of the same transformation or round function. Feistel ciphers are sometimes called DES-like ciphers (see Question 3.2.1).

In a Feistel cipher (see Figure 1), 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 exclusive-ORed 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 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 3.6.7).

Modes of Operation

When we use a block cipher to encrypt a message of arbitrary length, we use techniques known as modes of operation for the block cipher. To be useful a mode 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 3.2.3) 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 2), each plaintext block is encrypted independently with the block cipher.

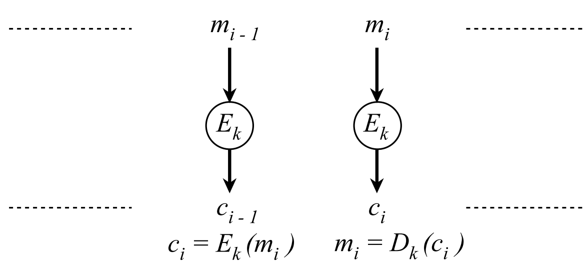
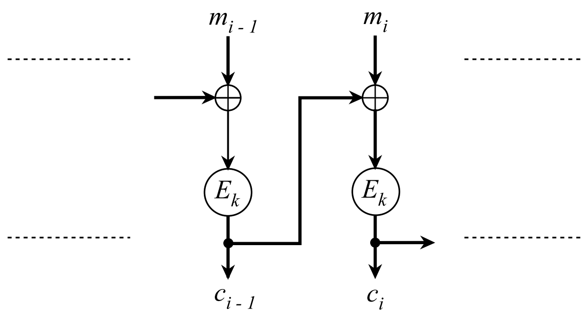


Figure 2. Electronic Code Book Mode

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 3), each plaintext block is exclusive-ORed with previous ciphertext block, then encrypted. An initialization vector or value c_0 is used as a "seed" for the process.

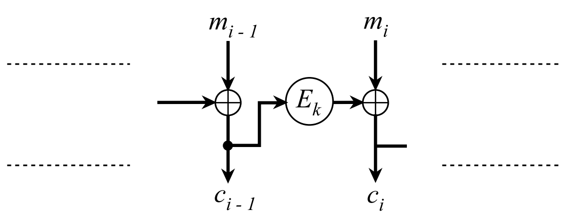


$$c_i = E_k(m_i \oplus c_{i-1}) \qquad m_i = D_k(c_i) \oplus c_{i-1}$$

Figure 3. 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 exclusive-ORing 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, although the decryption process can be.

In CFB mode (see Figure 4), the previous ciphertext block is encrypted and the output produced is combined with the plaintext block using exclusive-OR to produce the current ciphertext block. It is possible to define CFB mode so it uses feedback that is less than one full data block. An initialization vector or value c_0 is used as a “seed” for the process.



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

Figure 4. 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 exclusive-or 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 block cipher 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 5) is similar to CFB mode except the quantity exclusive-ORed 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 exclusive-OR of the plaintext block with the relevant data block.

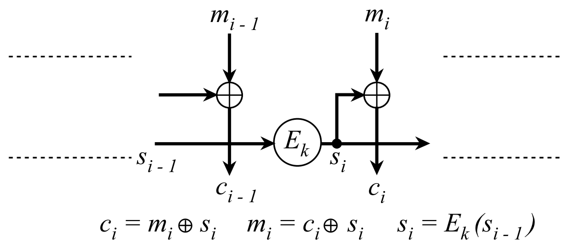


Figure 5. Output Feedback Mode

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 be easily 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.

Due to shortcomings in OFB mode, 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 $i + IV \pmod{2^{64}}$ 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. 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.

Question 2.1.5. What is a stream cipher?

A stream cipher is a type of symmetric encryption algorithm. Stream ciphers can be designed to be exceptionally fast, much faster than any block cipher (see Question 2.1.4). 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* (a sequence of bits used as a key). Encryption is accomplished by combining the keystream with the plaintext, usually with the bitwise exclusive-OR 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.

One-time Pads

Current interest in stream ciphers is most commonly attributed to the appealing theoretical properties of the *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 exclusive-OR with the plaintext to produce the ciphertext. Since the entire keystream is random, even an opponent with infinite computational resources can only guess the plaintext if he or she 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 in general impractical.

Stream ciphers were developed as an approximation to the action of the one-time pad. While contemporary stream ciphers are unable to provide the satisfying theoretical security of the one-time pad, they are at least practical.

As of now there is no stream cipher that has emerged as a de facto standard. The most widely used stream cipher is RC4 (see Question 3.6.3). Interestingly, certain modes of operation of a block cipher effectively transform it into a keystream generator and in this way, any block cipher can be used as a stream cipher; as in DES in CFB or OFB modes (see Questions 2.1.4 and 3.2.1). However, stream ciphers with a dedicated design are typically much faster.

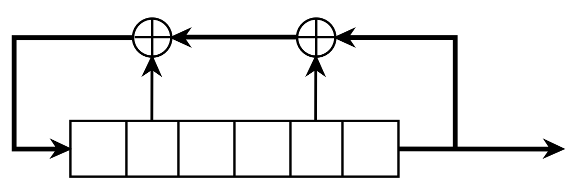


Figure 6. Linear Feedback Shift Register (LFSR)

A *Linear Feedback Shift Register* (LFSR) is a mechanism for generating a sequence of binary bits. The register (see Figure 6) 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 exclusive-or 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 particular bits that are used, in Figure 6, the 2nd and 5th bits are "tapped") the sequences that are generated can have a good statistical appearance. However, the sequences generated by a single LFSR 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.

A *shift register cascade* is a set of LFSRs 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].

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. 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* [MS95b], 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.

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 TR801 [Rob95b].

Question 2.1.6. What is a hash function?

A *hash function* H is a transformation that takes an 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 not equal to 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 $H(x) = H(y)$.

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

The hash value represents concisely the longer message or document from which it was computed; this value is called the message digest. 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 3.6.6) and SHA (see Question 3.6.5).

Perhaps the main role of a cryptographic hash function is in the provision of message integrity checks and digital signatures. Since hash functions are generally faster than encryption or digital signature algorithms, it is typical to compute the digital signature or integrity check to some document by applying cryptographic processing to 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 7.11) where, using hash functions, one can get a document timestamped without revealing its contents to the timestamping service.

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. Given a compression function, a hash function can be defined by 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 whose length depends on the compression function, and "padded" (for security reasons) so 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 7).

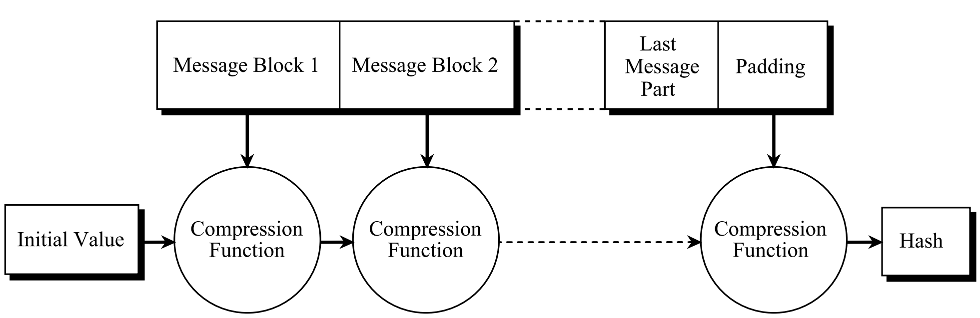


Figure 7. Damgård/Merkle iterative structure for hash functions

Question 2.1.7. What are Message Authentication Codes (MACs)?

A message authentication code (MAC) is an authentication tag (also called a checksum) derived by appying an authentication scheme, together with a secret key, to a message. Unlike digital signatures, MACs are computed and verified with the same key, so that they can only be verified by the intended recipient. There are four types of MACs: (1) unconditionally secure, (2) hash function-based, (3) stream cipher-based, or (4) block cipher-based.

Simmons and Stinson [Sti95] proposed an unconditionally secure MAC 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 (often called HMACs) use a key or keys in conjunction with a hash function (see Question 2.1.6) to produce a checksum that is appended to the message. An example is the keyed-MD5 (see Question 3.6.6) method of message authentication [KR95b].

Lai, Rueppel, and Woolven [LRW92] proposed a MAC based on stream ciphers (see Question 2.1.5). 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 2.1.5); the checksum is the final state of the two LFSRs.

MACs can also be derived from block ciphers (see Question 2.1.4). 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 2.1.4) and output the final block in the ciphertext as the checksum. Bellare et al. give an analysis of the security of this MAC [BKR94].

Question 2.1.8. 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. Apart from their theoretical interest, 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, such as the prover's private key.

It is useful for interactive proofs to have the following properties, especially in cryptographic applications:

- **Completeness:** The verifier always accepts the proof if the fact is true and both the prover and the verifier follow the protocol.
- **Soundness:** The verifier always rejects the proof if the fact is false, 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, even if the verifier does not follow the protocol (as long as the prover does). In a zero-knowledge proof, the verifier cannot even later prove the fact to anyone else. (Not all interactive proofs have this property.)

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.

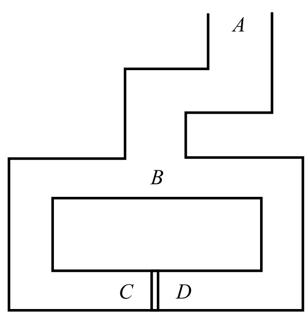


Figure 8. Ali Baba's Cave

Let us consider an intuitive example called Ali Baba's Cave [QG90] (see Figure 8). Alice wants to prove to Bob she knows the secret words that will open the portal at C-*D* 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 B and 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 she will come out from the right tunnel. Bob will repeat this round as many times as he desires until he is certain Alice knows the secret words. No matter how many times the proof repeats, Bob does not learn the secret words.

There are a number of zero-knowledge and interactive proof 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).

Identification schemes based on interactive proofs can usually be transformed into digital signature schemes (see Question 2.2.2 and [FS87]).

Question 2.1.9. 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 (for example, 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.

A general secret sharing scheme specifies the minimal sets of users who are able to recover the secret by sharing their secret information.

A common example of secret sharing is an m-out-of-n scheme (or (m,n)-threshold scheme) for integers $1 \le m \le n$. In such a 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 do not suffice to determine the secret. The pieces are usually called shares or shadows. Different choices for the values of m and n reflect the tradeoff between security and reliability. An m-out-of-n secret sharing scheme is *perfect* if any group of at most $m - 1$ participants (insiders) cannot determine any information about the secret.

Both Shamir's scheme and Blakley's scheme (see Question 3.6.12) are m-out-of-n secret sharing schemes (Shamir's scheme is perfect). They represent two different ways of constructing such schemes, based on which more advanced secret sharing schemes can be designed. The study of the combination of proactive techniques (see Question 7.16) with secret sharing schemes is an active area of research. For further information on secret sharing schemes, see [Sim92].

Question 2.2.1. What is privacy?

Privacy is perhaps the most obvious application of cryptography. Cryptography can be used to implement privacy simply by encrypting the information intended to remain private. In order for someone to read this private data, one must first decrypt it. Note that sometimes information is not supposed to be accessed by anyone, and in these cases, the information may be stored in such a way that reversing the process is virtually impossible. For instance, on a typical multi-user system, no one is supposed to know the list of passwords of everyone on the system. Often hash values of passwords are stored instead of the passwords themselves. This allows the users of the system to be confident their private information is actually kept private while still enabling an entered password to be verified (by computing its hash and comparing that result against a stored hash value).

Question 2.2.2. What is a digital signature and what is authentication?

Authentication is any process through which one proves and verifies certain information. Sometimes one may want to verify the origin of a document, the identity of the sender, the time and date a document was sent and/or signed, the identity of a computer or user, and so on. A digital signature is a cryptographic means through which many of these may be verified. The *digital signature* of a document is a piece of information based on both the document and the signer's private key. It is typically created through the use of a hash function (see Question 2.1.6) and a private signing function (encrypting with the signer's private key), but there are other methods.

Every day, people sign their names to letters, credit card receipts, and other documents, demonstrating they are in agreement with the contents. That is, they authenticate that they are in fact the sender or originator of the item. This allows others to verify that a particular message did indeed originate from the signer. However, this is not foolproof, since people can 'lift' signatures off one document and place them on another, thereby creating fraudulent documents. Written signatures are also vulnerable to forgery because it is possible to reproduce a signature on other documents as well as to alter documents after they have been signed.

Digital signatures and hand-written signatures both rely on the fact that it is very hard to find two people with the same signature. People use public-key cryptography to compute digital signatures by associating something unique with each person. When public-key cryptography is used to encrypt a message, the sender encrypts the message with the public key of the intended recipient. When public-key cryptography is used to calculate a digital signature, the sender encrypts the "digital fingerprint" of the document with his or her own private key. Anyone with access to the public key of the signer may verify the signature.

Suppose Alice wants to send a signed document or message to Bob. The first step is generally to apply a hash function to the message, creating what is called a message digest. The message digest is usually considerably shorter than the original message. In fact, the job of the hash function is to take a message of arbitrary length and shrink it down to a fixed length. To create a digital signature, one usually signs (encrypts) the message digest as opposed to the message itself. This saves a considerable amount of time, though it does create a slight insecurity (addressed below). Alice sends Bob the encrypted message digest and the message, which she may or may not encrypt. In order for Bob to authenticate the signature he must apply the same hash function as Alice to the message she sent him, decrypt the encrypted message digest using Alice's public key and compare the two. If the two are the same he has successfully authenticated the signature. If the two do not match there are a few possible explanations. Either someone is trying to impersonate Alice, the message itself has been altered since Alice signed it or an error occurred during transmission.

There is a potential problem with this type of digital signature. Alice not only signed the message she intended to but also signed all other messages that happen to hash to the same message digest. When two messages hash to the same message digest it is called a collision; the collision-free properties of hash functions (see Question 2.1.6) are a necessary security requirement for most digital signature schemes. A hash function is secure if it is very time consuming, if at all possible, to figure out the original message given its digest. However, there is an attack called the Birthday Attack that relies on the fact that it is easier to find two messages that hash to the same value than to find a message that hashes to a particular value. Its name arises from the fact that for a group of 23 or more people the probability that two or more people share the same birthday is better than 50%.

In addition someone could pretend to be Alice and sign documents with a key pair they claim is Alice's. To avoid scenarios such as this, there are digital documents called certificates that associate a person with a specific public key. For more information on certificates, see Question 4.1.3.10.

Digital time stamps may be used in connection with digital signatures to bind a document to a particular time of origin. It is not sufficient to just note the date in the message, since dates on computers can be easily manipulated. It is better that timestamping is done by someone everyone trusts, such as a certifying authority (see Question 4.1.3.12).

Question 2.2.3. What is a key agreement protocol?

A *key agreement* protocol, also called a key exchange protocol, is a series of steps used when two or more parties need to agree upon a key to use for a secret-key cryptosystem. These protocols allow people to share keys freely and securely over any insecure medium, without the need for a previously-established shared secret.

Suppose Alice and Bob want to use a secret-key cryptosystem (see Question 2.1.2) to communicate securely. They first must decide on a shared key. Instead of Bob calling Alice on the phone and discussing what the key will be, which would leave them vulnerable to an eavesdropper, they decide to use a key agreement protocol. By using a key agreement protocol, Alice and Bob may securely exchange a key in an insecure environment. One example of such a protocol is called the Diffie-Hellman key agreement (see Question 3.6.1). In many cases, public-key cryptography is used in a key agreement protocol. Another example is the use of digital envelopes (see Question 2.2.4) for key agreement.

Question 2.2.4. What is a digital envelope?

When using secret-key cryptosystems, users must first agree on a session key, that is, a secret key to be used for the duration of one message or communication session. In completing this task there is a risk the key will be intercepted during transmission. This is part of the key management problem. Public-key cryptography offers an attractive solution to this problem within a framework called a digital envelope.

The digital envelope consists of a message encrypted using secret-key cryptography and an encrypted secret key. While digital envelopes usually use public-key cryptography to encrypt the secret key, this is not necessary. If Alice and Bob have an established secret key, they could use this to encrypt the secret key in the digital envelope.

Suppose Alice wants to send a message to Bob using secret-key cryptography for message encryption and public-key cryptography to transfer the message encryption key. Alice chooses a secret key and encrypts the message with it, then encrypts the secret key using Bob's public key. She sends Bob both the encrypted secret key and the encrypted message. When Bob wants to read the message he decrypts the secret key, using his private key, and then decrypts the message, using the secret key. In a multi-addressed communications environment such as *e*-mail, this can be extended directly and usefully. If Alice's message is intended for both Bob and Carol, the message encryption key can be represented concisely in encrypted forms for Bob and for Carol, along with a single copy of the message's content encrypted under that message encryption key.

Alice and Bob may use this key to encrypt just one message or they may use it for an extended communication. One of the nice features about this technique is they may switch secret keys as frequently as they would like. Switching keys often is beneficial because it is more difficult for an adversary to find a key that is only used for a short period of time (see Question 4.1.2.3 for more information on the life cycle of a key).

Not only do digital envelopes help solve the key management problem, they increase performance (relative to using a public key system for direct encryption of message data) without sacrificing security. The increase in performance is obtained by using a secret-key cryptosystem to encrypt the large and variably sized amount of message data, reserving public-key cryptography for encryption of short-length keys. In general, secret-key cryptosystems are much faster than public-key cryptosystems.

The digital envelope technique is a method of key exchange, but not all key exchange protocols use digital envelopes (see Question 2.2.3).

Question 2.2.5. What is identification?

Identification is a process through which one ascertains the identity of another person or entity. In our daily lives, we identify our family members, friends, and coworkers by their physical properties, such as voice, face or other characteristics. These characteristics, called biometrics (see Question 7.20), can only be used on computer networks with special hardware. Entities on a network may also identify one another using cryptographic methods.

An identification scheme allows Alice to identify herself to Bob in such a way that someone listening in cannot pose as Alice later. One example of an identification scheme is a zero-knowledge proof (see Question 2.1.8). Zero knowledge proofs allow a person (or a server, website, etc.) to demonstrate they have a certain piece information without giving it away to the person (or entity) they are convincing. Suppose Alice knows how to solve the Rubik's cube and wants to convince Bob she can without giving away the solution. They could proceed as follows. Alice gives Bob a Rubik's cube which he thoroughly messes up and then gives back to Alice. Alice turns away from Bob, solves the puzzle and hands it back to Bob. This works because Bob saw that Alice solved the puzzle, but he did not see the solution.

This idea may be adapted to an identification scheme if each person involved is given a "puzzle" and its answer. The security of the system relies on the difficulty of solving the puzzles. In the case above, if Alice were the only person who could solve a Rubik's cube, then that could be her puzzle. In this scenario Bob is the verifier and is identifying Alice, the prover.

The idea is to associate with each person something unique; something only that person can reproduce. This in effect takes the place of a face or a voice, which are unique factors allowing people to identify one another in the physical world.

Authentication and identification are different. Identification requires that the verifier check the information presented against all the entities it knows about, while authentication requires that the information be checked for a single, previously identified, entity. In addition, while identification must, by definition, uniquely identify a given entity, authentication does not necessarily require uniqueness. For instance, someone logging into a shared account is not uniquely identified, but by knowing the shared password, they are authenticated as one of the users of the account. Furthermore, identification does not necessarily authenticate the user for a particular purpose.

Question 2.3.1. What is a hard problem?

Public-key cryptosystems (see Question 2.1.1) are based on a problem that is in some sense difficult to solve. Difficult in this case refers more to the computational requirements in finding a solution than the conception of the problem. These problems are called *hard problems*. Some of the most well known examples are factoring, theorem-proving, and the "traveling salesman problem" - finding the route through a given collection of cities which minimizes the total length of the path.

There are two major classes of problems which interest cryptographers - *P* and *NP*. Put simply, problems in *P* can be solved in polynomial time, and problems not known to be in *P*, problems that currently cannot be solved in polynomial time are in *NP*. We know that all problems in *P* are also in *NP*, but we do not know whether or not all problems in *NP* are in *P*. However, *NP* problems are abundant.

The question of whether or not $P = NP$ is one of the most important unsolved problems in all of mathematics and computer science. So far, there has been very little progress towards its solution. One thing we do have is the concept of an *NP*-complete problem. Certain *NP* problems are said to be *NP*-complete. A problem is called *NP*-complete if it can be *reduced* (transformed) into any other *NP* problem in polynomial time . If any *NP*-complete problem is solved in polynomial time, then all *NP* problems can be solved in polynomial time. "The traveling salesman problem is *NP*-complete. Thus, to prove that $P = NP$, it would suffice to find a polynomial-time algorithm for one of the *NP*-complete problems. However, it is commonly thought that $P \neq NP$. If it were to be proved that $P = NP$, we could theoretically solve an enormous variety of complex problems quickly without a significant advance in computing technology. For more on the theory of computation, we recommend [GJ79] and [LP98].

Question 2.3.2. What is a one-way function?

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

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 compute the function easily in both directions, but anyone lacking the trap door can only perform the function easily 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 (i.*e.*, to generate a legitimate signature) .

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 it is theoretically possible to discover algorithms that can compute the inverse direction easily without a trap door for some of the one-way functions; this development would render any cryptosystem based on these one-way functions 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; this would be a landmark event with significant positive ramifications for cryptography.

Question 2.3.3. 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 of two (or more) prime numbers is much more difficult.

Factoring is the underlying, presumably hard problem upon which several public-key cryptosystems are based, including the RSA algorithm. Factoring an RSA modulus (see Question 3.1.1) 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 the RSA algorithm depends on the factoring problem being difficult and the presence of no other types of attack. There has been some recent evidence that breaking RSA is not equivalent to factoring (see Dan Boneh's recent paper on low exponent RSA in the proceedings from EUROCRYPT '98). 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 2.3.5), though factoring researchers consider this possibility remote.

In general the larger the number the more time it takes to factor it. Of course if you have a number like 2^100 it is easier to factor than say, a number with half as many digits but the product of two primes of about the same length. 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 the RSA modulus is to an attack.

Question 2.3.4. 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 efficiencies (see Question 2.3.1). (*O*-notation refers to the upper bound on the asymptotic running time of an algorithm [CLR90]). 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 flavors, 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. The *Pollard* p + 1 method is also a special purpose factoring algorithm, with running time $O(p')$, where p' is the largest prime factor of p+1. All 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 $O(e^{(\sqrt{2 \ln p \ln \ln p})})$. The ECM is often used in practice to find factors of randomly generated numbers; it is not fast enough to factor a large RSA modulus.

The best general-purpose factoring algorithm today is the *Number Field Sieve* (NFS) [BLP94] [BLZ94], which runs in time approximately $O(e^{1.9223(\ln n^{1/3})(\ln \ln n^{2/3})})$. Previously, the most widely used general-purpose algorithm was the *Multiple Polynomial Quadratic Sieve* (MPQS) [Sil87], which has running time $O(e^{\sqrt{(\ln n \ln \ln n)}})$. Recent improvements to the Number Field Sieve make the NFS more efficient than MPQS in factoring numbers larger than about 115 digits [DL95]; MPQS is better for small integers. RSA-129 (see Question 2.3.6) was factored using a variation of MPQS. It is now estimated that if the NFS had been used, it would have taken one quarter of the time.

Clearly, NFS will overtake MPQS as the most widely used factoring algorithm, as the size of the numbers being factored increases from about 130 digits (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 are easier to factor than general numbers [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 2.3.6).

Question 2.3.5. What improvements are likely in factoring capability?

Factoring (see Question 2.3.3) has become easier over the last 15 years for three reasons: computer hardware has become more powerful, computers have become more plentiful and inexpensive, and better factoring algorithms have emerged.

Hardware improvement will continue inexorably, but it is important to realize 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. Therefore, although the hardware improvement does help the attacker, it helps the legitimate user much more. However, there is a danger that in the future factoring will take place using faster machines than are currently available, and these machines may be used to attack RSA keys generated in the past. In this scenario, the attacker alone benefits from 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.

Recently, the number of computers has increased dramatically. While the computers have become steadily more powerful, the increase in their power has not compared to their increase in number. Since some factoring algorithms can be done with multiple computers working together, the more computers devoted to a problem, the faster the problem can be solved. Unlike the hardware improvement factor, prevalence of computers does not make RSA more secure.

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.

Increasing the key size can offset any decrease in security due to algorithm improvements. 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. However, if an "easy" solution to the factoring problem can be found, the associated increase in key sizes will render the RSA system impractical.

Factoring is widely believed to be a hard problem (see Question 2.3.1), but this has not yet been proven. 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 active in factoring research.

There is also the possibility someone will prove 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 7.17) if one is ever developed.
Question 2.3.6. What is the RSA Factoring Challenge and what is RSA-129?

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

RSA-129 is a 129-digit (426-bit) integer published in Martin Gardner's column in Scientific American in 1977, and was not part of the RSA Factoring Challenge. 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 2.3.4).

Question 2.3.7. What is the discrete logarithm problem?

The *discrete logarithm problem* applies to mathematical entities called groups. A *group* is a collection of elements, together with an operation defined on them that is usually referred to as composition or multiplication and which obey certain rules. If the group has a finite number of elements, each element in the group has what is called an *order*, the minimum number of times it must be multiplied by itself to get back to the identity, which is usually one. The discrete logarithm problem is as follows: given an element g in a group G of order t, and another element y of G, the problem is to find x, where 0 (x (t - 1, such that y is the result of composing g with itself x times. In some groups there exist elements that can generate all the elements of G by exponentiating (i.*e*., applying the group operation repeatedly) with all the integers from 0 to t - 1. When this occurs, the element is called a generator and the group is called cyclic.

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 3.6.8 and Question 3.4.1). 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. Although the discrete logarithm problem exists in any group, when used for cryptographic purposes the group is usually Z_p^*.

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 algorithms 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.

In general, the discrete log in an arbitrary group can be computed in running time $O(\sqrt{p})$ [Pol74], though in many groups it can be done faster.

Question 2.3.8. What are the best discrete logarithm methods in use today?

Currently, the best algorithms to solve the discrete logarithm problem (see Question 2.3.7) are broken into two classes: *index-calculus* methods and *collision search* methods. The two classes of algorithms differ in the ways they are applied. Index calculus methods generally require certain arithmetic properties to be present in order to be successful, whereas collision search algorithms can be applied much more generally. The absence of mere properties in elliptic curve groups prevents the more powerful index-calculus techniques from being used to attack the elliptic curve analogues of the more traditional discrete logarithm based cryptosystems (see Question 3.5.1).

Index calculus methods are very similar to the fastest current methods for integer factoring and they run in what is termed sub-exponential time. They are not as fast as polynomial time algorithms, yet they are considerably faster than exponential time methods. There are two basic index calculus methods closely related to the quadratic sieve and number field sieve factoring algorithms (see Question 2.3.4).

As of this time, the largest discrete log problem that has been solved was over GF(2^{503}).

Collision search algorithms have purely exponential run time. The best general method is known as the Pollard rho method, so-called because the algorithm produces a trail of numbers that when graphically represented with a line connecting successive elements of the trail looks like the Greek letter rho. There is a tail and a loop; the objective is basically to find where the tail meets the loop. This method runs in time $(^{\pi}/_{2}\, p$ where p is the size of the group. The largest such problem that has been publicly solved has $p \sim 2^{79}$. This is the best known method of attack for the general elliptic curve discrete log problem.

Question 2.3.9. What are the prospects for a theoretical breakthrough in the discrete log problem?

It is impossible to predict when a mathematical breakthrough might occur; this is why they are called breakthroughs. Factoring algorithms have been studied for hundreds of years, general discrete log algorithms have been extensively studied since the early 1970s, and elliptic curve discrete logs have been studied since the mid-1980s. Each time a new algorithm has been announced it has come more or less as a surprise to the research community.

It should be noted that for integer factoring and general discrete logs, a 'breakthrough' means finding a polynomial time algorithm. However, for elliptic curve discrete logs, a breakthrough would consist of just finding a sub-exponential time method. If the latter were found it would mean that elliptic curve discrete logs would no longer be competitive with integer factoring and general discrete logs as a public-key method. Elliptic curve cryptosystems (see Question 3.5.1) derive their current advantage because they currently allow smaller keys. If a sub-exponential time method were found, key sizes would have to increase greatly, and at equivalent key. Elliptic curve cryptosystems are much slower than other public-key methods

Question 2.3.10. 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.

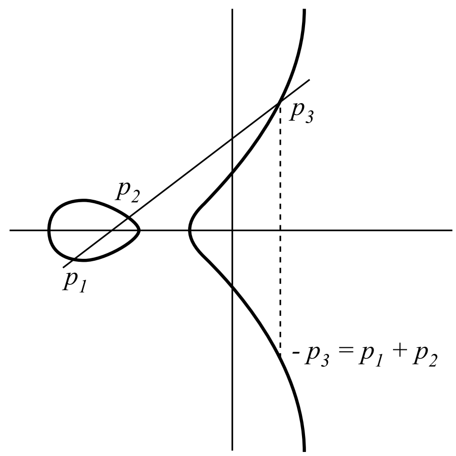


Figure 9. Elliptic curve addition.

An elliptic curve can be defined over any field (*e.g.*, real, rational, complex), though 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. (The elliptic curve formula is slightly different for some fields.)

Addition of two points on a elliptic curve is defined according to a set of simple rules (*e.g.*, in Figure 9, point p_1 plus point p_2 is equal to point $-p_3$). 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 an informative text by Koblitz [Kob94].

Question 2.3.11. What are lattice-based cryptosystems?

Lattice-based cryptosystems are based upon NP-complete problems (see Question 2.3.1) involving lattices. A *lattice* can be viewed as a linear combination of vectors in an *N*-dimensional vector space. An example of a lattice is the infinite square grid in two-dimensional space consisting of all points with integral coordinates. This lattice is generated by linear combinations of the pair of vectors <0,1> and <1,0>.

Lattice-based methods fall into two basic classes, although the solution methods for both are identical. In fact, there are efficient transformations between the two classes. The first class is based on the so-called *subset sum* problem: Given a set of numbers $S = \{a_1, a_2, \ldots a_t\}$ and another number K, find a subset of S whose values sum to K. The knapsack problem of Merkle and Hellman is an example of this [MH78].

Other lattice-based methods require finding short vectors embedded in a lattice or finding points in the vector space close to vertices of the lattice or close to vectors embedded in the lattice. The method of Ajtai and Dwork is an example of this type of method.

So far lattice-based methods have not proven effective as a foundation for public-key methods. In order for a lattice-based cryptosystem to be secure, the dimension of the underlying problem has to be large. This results in a large key size, rendering encryption and decryption quite slow. Ongoing research aims to improve the efficiency of these cryptosystems.

Question 2.3.12. What are some other hard problems?

There are many other kinds of hard problems (see Question 2.3.1). The list of NP-complete problems (see Question 2.3.1) is extensive and growing. So far, none of these has been effectively applied towards producing a public-key cryptosystem. A few examples of hard problems are the Traveling Salesman Problem, Integer and Mixed Integer Programming, Graph Coloring, the Hamiltonian Path problem and the Satisfiability Problem for Boolean Expressions. A good introduction to this topic may be found in [AHU74].

The Traveling Salesman problem is to find a minimal length tour among a set of cities, while visiting each one only once.

The Integer Programming problem is to solve a Linear Programming problem where some or all of the variables are restricted to being integers.

The Graph Coloring problem is to determine whether a graph can be colored with a fixed set of colors such that no two adjacent vertices have the same color, and to produce such a coloring.

The Hamiltonian path problem is to decide if one can traverse a graph by using each edge exactly once.

The Satisfiability Problem is to determine whether a Boolean expression in several variables has a solution.

Another hard problem is the Knapsack problem, a narrow case of the subset sum problem (see Question 2.3.11). Attempts have been made to make public-key cryptosystems based on the knapsack problem, but none have yielded strong results. The Knapsack problem is to determine which subset of a set of objects weighing different amounts has maximal total weight, but still has total weight less than the capacity of the "Knapsack."

Question 2.4.1. What is cryptanalysis?

Cryptanalysis is the flip-side of cryptography: it is the science of cracking codes, decoding secrets, violating authentication schemes, and in general, breaking cryptographic protocols.

In order to design a robust encryption algorithm or cryptographic protocol, one should use cryptanalysis to find and correct any weaknesses. This is precisely the reason why the best (most trusted) encryption algorithms are ones that have been made available to public scrutiny. For example, DES (see Question 3.2.1) has been exposed to public scrutiny for years, and is therefore well-trusted, while Skipjack (see Question 3.6.7) is secret and less well-trusted. It is a basic tenet of cryptology that the security of an algorithm should not rely on its secrecy. Inevitably, the algorithm will be discovered and its weaknesses (if any) will be exploited.

The various techniques in cryptanalysis attempting to compromise cryptosystems are referred to as *attacks*. Some attacks are general, whereas others apply only to certain types of cryptosystems. Some of the better-known attacks are mentioned in Question 2.4.2.

Question 2.4.2. What are some of the basic types of cryptanalytic attack?

Cryptanalytic attacks are generally classified into six categories that distinguish the kind of information the cryptanalyst has available to mount an attack. The categories of attack are listed here roughly in increasing order of the quality of information available to the cryptanalyst, or, equivalently, in decreasing order of the level of difficulty to the cryptanalyst. The objective of the cryptanalyst in all cases is to be able to decrypt new pieces of ciphertext without additional information. The ideal for a cryptanalyst is to extract the secret key.

A *ciphertext-only* attack is one in which the cryptanalyst obtains a sample of ciphertext, without the plaintext associated with it. This data is relatively easy to obtain in many scenarios, but a successful ciphertext-only attack is generally difficult, and requires a very large ciphertext sample.

A *known-plaintext* attack is one in which the cryptanalyst obtains a sample of ciphertext and the corresponding plaintext as well.

A *chosen-plaintext* attack is one in which the cryptanalyst is able to choose a quantity of plaintext and then obtain the corresponding encrypted ciphertext.

An *adaptive-chosen-plaintext* attack is a special case of chosen-plaintext attack in which the cryptanalyst is able to choose plaintext samples dynamically, and alter his or her choices based on the results of previous encryptions.

A *chosen-ciphertext* attack is one in which cryptanalyst may choose a piece of ciphertext and attempt to obtain the corresponding decrypted plaintext. This type of attack is generally most applicable to public-key cryptosystems.

An *adaptive-chosen-ciphertext* is the adaptive version of the above attack. A cryptanalyst can mount an attack of this type in a scenario in which he has free use of a piece of decryption hardware, but is unable to extract the decryption key from it.

Note that cryptanalytic attacks can be mounted not only against encryption algorithms, but also, analogously, against digital signature algorithms (see Question 2.2.2), MACing algorithms (see Question 2.1.7), and pseudo-random number generators (see Question 2.5.2).

Question 2.4.3. 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 2.1.4) 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 3.2.1) was designed, it was generally considered secure against exhaustive key search without a vast financial investment in hardware [DH77]. To date, there is no public evidence that such hardware has been constructed. Over the years, however, this line of attack will become increasingly attractive to a potential adversary [Wie94]. Another useful article on exhaustive key search can be found in the Winter 1997 issue of CryptoBytes available online at the following URL: (http://www.rsa.com/rsalabs/pubs/cryptobytes/html/article_index.html).

Exhaustive key search may also be performed in software running on standard desktop workstations and personal computers. While exhaustive search of DES's 56-bit key space would take hundreds of years on the fastest general purpose computer available today, the growth of the Internet has made it possible to utilize thousands of such machines in a distributed search by partitioning the key space and distributing small portions to each of a large number of computers. Recently, a group called distributed.net solved RSA's DES Challenge II, using an estimated 50,000 processors to search 85% of the possible keys, in 39 days.

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 an 80-bit key as used by Skipjack (see Question 3.6.7) can be expected to offer the same level of protection against exhaustive key search in 18 years time as DES does today [BDK93]. Absent a major breakthrough in quantum computing (see Question 7.17), it is unlikely that 128-bit keys, such as those used in IDEA (see Question 3.6.7) or RC5-32/12/16 (see Question 3.6.4), will be broken by exhaustive search in the foreseeable future.

Question 2.4.4. What is the RSA Secret Key Challenge?

RSA Laboratories started the RSA Secret Key Challenge in January 1997. The goal of the challenges is to quantify the security offered by secret key ciphers (see Question 2.1.2) with keys of various sizes. The information obtained from these contests is anticipated to be of value to researchers and developers alike as they estimate the strength of an algorithm or application against exhaustive key-search.

Initially, thirteen challenges were issued, of which four have been solved as of January 1, 1998. There were twelve RC5 challenges and one DES challenge, with key sizes ranging from 40 bits to 128 bits. The 56-bit DES challenge and the 40, 48, and 56 bit RC5 challenges have all been solved.

In January 1998, RSA Laboratories launched the DES challenge II, which consists of a series of DES challenges to be released twice per year. It is expected that each time the amount of time needed to solve the challenge will decrease substantially.

The first DES challenge II was solved by a coordinated team of computer programmers and enthusiasts known as distributed.net in 39 days, which is less than half the 90 days of computing time it took the original challenge to be solved by a university team.

For more information, send email to challenge-info@rsa.com or visit the web site at http://www.rsa.com/rsalabs/97challenge/. For more information on the DES II Challenge, visit the web at http://www.rsa.com/rsalabs/des2.

Question 2.4.5. What are the most important attacks on symmetric block ciphers?

There are several attacks which are specific to block ciphers (see Question 2.1.4). Four such attacks are *differential cryptanalysis*, *linear cryptanalysis*, the exploitation of *weak keys*, and *algebraic attacks*.

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 3.6.7), but they were later improved and perfected by Biham and Shamir [BS91a] [BS93b] who used them to attack DES (see Question 3.2.1). Differential cryptanalysis is basically a chosen plaintext attack (see Question 2.4.2); it 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 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 primitives such as hash functions.

Matsui and Yamagishi [MY92] first devised *linear cryptanalysis* in an attack on FEAL (see Question 3.6.7). It was extended by Matsui [Mat93] to attack DES (see Question 3.2.1). Linear cryptanalysis is a known plaintext attack (see Question 2.4.2) which 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 that combines elements of differential cryptanalysis 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].

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 3.2.1), 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 3.6.7), 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.

Algebraic attacks are a class of techniques that rely for their success on block ciphers 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 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. DES, however, it is known not to be a group.

There are a variety of other concerns with regards to algebraic attacks. See [Rob95a] for more details.
Question 2.4.6. What are some techniques against hash functions?

The essential cryptographic properties of a hash function are that it is both one-way and collision-free (see Question 2.1.6). 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 on the order of 2^n possible input values.

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.

If we are trying to find a collision, then by the birthday paradox we would expect that after trying $1.2(2^{n/2})$ 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:

- The adversary selects two messages: 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 key.

Pseudo-collisions are collisions for the compression function (see Question 2.1.6) 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 3.6.6) is an example of a hash function for which pseudo-collisions have been discovered and yet is still considered secure.

Question 2.4.7. What are the most important attacks on stream ciphers?

The most typical use of a stream cipher for encryption is to generate a keystream in a way that depends on the secret key and then to combine this (typically using bitwise exclusive-or) with the message being encrypted.

It is imperative the keystream "looks" random; that is, after seeing increasing amounts of the keystream, an adversary should have no additional advantage in being able to predict any of the subsequent bits of the sequence. While there are some attempts to guarantee this property in a provable, most stream ciphers rely on ad hoc analysis. A necessary condition for a secure stream cipher is that it pass a battery of statistical tests which assess (among other things) the frequencies with which individual bits or consecutive patterns of bits of different sizes occur. Such tests might also check for *correlation* between bits of the sequence occurring at some time instant and those at other points in the sequence. Clearly the amount of statistical testing will depend on the thoroughness of the designer. It is a very rare and very poor stream cipher that does not pass most suites of statistical tests.

A keystream might potentially have structural weaknesses that allow an adversary to deduce some of the keystream. Most obviously, if the *period* of a keystream, that is, the number of bits in the keystream before it begins to repeat again, is too short, the adversary can apply discovered parts of the keystream to help in the decryption of other parts of the ciphertext. A stream cipher design should be accompanied by a guarantee of the minimum period for the keystreams that might be generated or alternatively, good theoretical evidence for the value of the lower bound to such a period. Without this, the user of the cryptosystem cannot be assured that a given keystream will not repeat far sooner than might be required for cryptographic safety.

A more involved set of structural weaknesses might offer the opportunity of finding alternative ways to generate part or even the whole of the keystream. Chief among these approaches might be using a *linear feedback shift register* to replicate part of the sequence. The motivation to use a linear feedback shift register is due to an algorithm of *Berlekamp* and *Massey* that takes as input a finite sequence of bits and generates as output the details of a linear feedback shift register that could be used to generate that sequence. This gives rise to the measure of security known as the *linear complexity* of a sequence; for a given sequence, the linear complexity is the size of the linear feedback shift register that needs to be used to replicate the sequence. Clearly a necessary condition for the security of a stream cipher is that the sequences it produces have a high linear complexity. RSA Laboratories Technical Report TR-801 [Koc95] describes in more detail some of these issues and also some of the other alternative measures of complexity that might be of interest to the cryptographer and cryptanalyst.

Other attacks attempt to recover part of the secret key that was used. Apart from the most obvious attack of searching for the key by brute force, a powerful class of attacks can be described by the term *divide and conquer*. During off-line analysis the cryptanalyst identifies some part of the key that has a direct and immediate effect on some aspect or component of the generated keystream. By performing a brute-force search over this smaller part of the secret key and observing how well the sequences generated match the real keystream, the cryptanalyst can potentially deduce the correct value for this smaller fraction of the secret key [Koc95]. This correlation between the keystream produced after making some guess to part of the key and the intercepted keystream gives rise to what are termed correlation attacks and later the more efficient fast correlation attacks [Koc95].

Finally there are some implementation considerations. A synchronous stream cipher allows an adversary to change bits in the plaintext without any error-propagation to the rest of the message. If authentication of the message being encrypted is required, the use of a cryptographic MAC might be advisable. As a separate implementation issue synchronization between sender and receiver might sometimes be lost with a stream cipher and some method is required is ensure the keystreams can be put back into step. One typical way of doing this is for the sender of the message to intersperse synchronization markers into the transmission so only that part of the transmission which lies between synchronization markers might be lost. This process however does carry some security implications.

Question 2.4.8. What are the most important attacks on MACs?

There are a variety of threats to the security of a MAC (see Question 2.1.7). First and most obviously, the use of a MAC should not reveal information about the secret key being used. Second, it should not be possible for an adversary to forge the correct MAC to some message without knowing the secret key—even after seeing many legitimate message/MAC pairs. Third, it should not be possible to replace the message in a message/MAC pair with another message for which the MAC remains legitimate. There are a variety of threat models that depend on different assumptions about the data that might be collected. For example, can an adversary control the messages whose MACs are obtained, and if so, can the choice be adapted as more data is collected?

Depending on the design of the MAC there are a variety of different attacks that might apply. Perhaps the most important class of attacks is due to Preneel and van Oorschot [PV95]. These attacks involve a sophisticated application of the birthday paradox (see Question 2.4.6) to the analysis of message/MAC pairs and the attacks have been particularly useful in highlighting structural faults in the design of many MACs. Some considerable work was spent in the early to mid-1990's on designing MACs based around the use of a hash function. The attacks of Preneel and van Oorschot were instrumental in removing many of these flawed designs from consideration.

Question 2.4.9. At what point does an attack become practical?

There is no easy answer to this question as the answer 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. Furthermore, the value of the concealed information must be taken into account—it is reasonable to spend a million dollars of effort to uncover something worth more than a million dollars, however, any sane attacker would not, for example, invest one million dollars to uncover a secret worth one thousand dollars.

Also, it should be noted that cryptography and security are not equivalent. If a block cipher takes seven months of computational effort to crack, but the key can be recovered by bribery or extortion, a truly dedicated adversary will probably attempt the latter.

Question 2.5.1. What is primality testing?

Primality testing is the process of proving a number is *prime* (an integer greater than 1 is prime if it is divisible only by itself and 1). It is used in the key generation process for cryptosystems that depend on secret prime numbers, such as RSA. *Probabilistic primality testing* is a process which proves a number has a high probability of being prime.

To generate a random prime number, random numbers are generated (see Question 2.5.2) and tested for primality until one of them is found to be prime (or very likely to be prime, if probabilistic testing is used).

It is generally recommended to use probabilistic primality testing, which is much quicker than actually proving 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 Damgård [BD93b].

Question 2.5.2. What is random number generation?

Random number generation is used in a wide variety of cryptographic operations, such as key generation and challenge/response protocols. A random number generator is a function that outputs a sequence of 0s and 1s such that at any point, the next bit cannot be predicted based on the previous bits. However, true random number generation is difficult to do on a computer, since computers are deterministic devices. Thus, if the same random generator is run twice, identical results are received. True random number generators are in use, but they can be difficult to build. They typically take input from something in the physical world, such as the rate of neutron emission from a radioactive substance or a user's idle mouse movements.

Because of these difficulties, random number generation on a computer is usually only *pseudo*-random number generation. A pseudo-random number generator produces a sequence of bits that has a random looking distribution. With each different *seed (a typically random stream of bits used to generate a usually longer pseudo-random stream)*, the pseudo-random number generator generates a different pseudo-random sequence. With a relatively small random seed a pseudo-random number generator can produce a long apparently random string.

Pseudo-random number generators are often based on cryptographic functions like block ciphers or stream ciphers. For instance, iterated DES encryption starting with a 56-bit seed produces a pseudo-random sequence.

Question 3.1.1. What is RSA?

RSA is a public-key cryptosystem that offers both encryption and digital signatures (authentication). Ron Rivest, Adi Shamir, and Leonard Adleman developed RSA in 1977 [RSA78]; RSA stands for the first letter in each of its inventors' last names.

RSA works as follows: take two large primes, p and q, and compute 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 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 may be kept with the private key, or destroyed.

It is currently difficult to obtain the private key d from the public key (n, e). However if one could factor n into p and q, then one could obtain the private key d. Thus the security of RSA is based on the assumption that factoring is difficult. The discovery of an easy method of factoring would "break" RSA (see Question 3.1.3 and Question 2.3.3).

Here is how RSA can be used for encryption and digital signatures (in practice, the actual use is slightly different; see Question 3.1.7 and Question 3.1.8):

RSA 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 this message.

RSA Digital Signature
Suppose Alice wants to send a message m to Bob in such a way that Bob is assured the message is both authentic, has not been tampered with, and 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 another's public key or their own private key. Anyone can send an encrypted message or verify a signed message, but only someone in possession of the correct private key can decrypt or sign a message.

Question 3.1.2. How fast is RSA?

An "RSA operation," whether encrypting, decrypting, signing, or verifying is essentially a modular exponentiation. This computation is 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 the typical modular exponentiation algorithms used to implement RSA, public key operations take $O(k^2)$ steps, private key operations take $O(k^3)$ steps, and key generation takes $O(k4)$ steps, where k is the number of bits in the modulus. "Fast multiplication" techniques, such as FFT-based methods, require asymptotically fewer steps. In practice, however, they are not as common due to their greater software complexity and the fact that they may actually be slower for typical key sizes.

The speed and efficiency of the many commercially available software and hardware implementations of RSA are increasing rapidly. On a 90 MHz Pentium, RSA Data Security's cryptographic toolkit BSAFE 3.0 (see Question 5.2.3) 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 in late 1999.

By comparison, DES (see Question 3.2. 1) and other block ciphers are much faster than RSA. In software, DES is generally at least 100 times 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, due to high demand, but DES will get faster as well.

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

Question 3.1.3. 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 Questions 2.3.4 and 2.3.5 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 2.3.5).

Another way to break RSA is to find a technique to compute eth roots mod n. Since $c = m^e \bmod n$, the eth 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 have 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 and guesses that the message might be, for example, "Attack at dawn," and encrypts this guess with the public key of the recipient and by comparison with the actual ciphertext, the attacker knows whether or not the guess was correct. Appending some random bits to the message can thwart this attack. 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. See Section 4.1.3 for discussion of secure key management in an RSA system.

Question 3.1.4. What are strong primes and are they necessary for 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* have 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 some factoring methods (for instance, the Pollard p-1 and p+1 methods, see Question 2.3.4) 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 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 new factoring algorithms may be developed in the future which once again target primes with certain properties. If this happens, choosing strong primes may once again help to increase security.

Question 3.1.5. How large a key should be used in RSA?

The size of an RSA key typically refers to the size of the modulus n. The two primes, p and q, which compose the modulus, should be of roughly equal length; this makes the modulus harder to factor than if one of the primes is much smaller than the other. 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), or more generally, if their difference is close to any predetermined amount, then there is a potential security risk, but the probability that two randomly chosen primes are so close is negligible.

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 the value of the protected data and how long it needs to be protected, and, second, of how powerful one's potential threats might be.

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 an article by Odlyzko [Odl95]. Odlyzko considers the security of RSA key sizes based on factoring techniques available in 1995 and on potential future developments, and he also considers the ability to tap large computational resources via computer networks. In 1997, 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 [Rob95d]. It is believed that 512-bit keys no longer provide sufficient security for anything more than very short-term security needs. RSA Laboratories currently recommends key sizes of 768 bits for personal use, 1024 bits for corporate use, and 2048 bits for extremely valuable keys like the root key pair used by a certifying authority (see Question 4.1.3.12).

It is typical to ensure that the key of an individual user expires after a certain time, say, two years (see Question 4.1.3.5). This gives an opportunity to change keys regularly and to maintain a given level of security. Upon expiration, the user should generate a new key being sure to ascertain whether any changes in cryptanalytic skills make a move to longer key lengths appropriate. . (Of course, changing a key doesn't defend against attacks that attempt to recover messages encrypted with an old key, so key size should always be chosen according to the expected lifetime of the data. The opportunity to change keys allows one to adapt to new key size recommendations.) RSA Laboratories publishes recommended key lengths 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 the attacker may get lucky and factor some key quickly.

As for the slowdown caused by increasing the key size (see Question 3.1.2), 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. The reason public key operations are affected less than private key operations is that the public exponent can remain fixed while the modulus is increased, whereas the length of 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.

It should be noted that the key sizes for RSA (and other public-key techniques) are much larger than those for block ciphers like DES (see Question 3.2.1), but the security of an RSA key cannot be compared to the security of a key in another system purely in terms of length.

Question 3.1.6. Could users of RSA run out of distinct primes?

As Euclid proved over two thousand years ago, there are infinitely many prime numbers. Because RSA is generally implemented with a fixed key length, however, the number of primes available to a user of the algorithm is effectively finite. Although finite, this number is nonetheless very large. The Prime Number Theorem states that the number of primes less than or equal to n is asymptotic to $n/\ln n$. Hence, the number of prime numbers of length 512 bits or less is roughly 10^{150}. This is greater than the number of atoms in the known universe.

Question 3.1.7. How is RSA used for privacy in practice?

In practice, RSA is often used together with a secret-key cryptosystem, such as DES (see Question 3.2.1), to encrypt a message by means of an RSA digital envelope (see Question 2.2.4).

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 the message itself. This combines the high speed of DES with the key-management convenience of RSA.

Question 3.1.8. How is RSA used for authentication and digital signatures in practice?

The RSA Public-key cryptosystem can be used to authenticate (see Question 2.2.2) or identify another person or entity. The reason it works well is because each entity has an associated private key which (theoretically) no one else has access to. This allows for positive and unique identification.

Suppose Alice wishes to send a signed message to Bob. She applies a hash function (see Question 2.1.6) 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, creating the digital signature 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 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. Anybody who reads the message can verify the signature. This does not satisfy 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. Alternately, if it is necessary for intermediary third parties to validate the integrity of the message without being able to decrypt its content, a message digest may be computed on the encrypted message, rather than on its plaintext form.

In practice, the RSA public exponent is usually much smaller than the RSA private exponent. This means that verification of a signature is faster than 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 3.6.5 and Question 3.6.6) 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 4.1.3.10) 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.

Question 3.1.9. Is RSA currently in use?

RSA 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. RSA 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 S/MIME (see Question 5.1.1), SSL (see Question 5.1.2), and S/WAN (see Question 5.1.3). It is also used internally in many institutions, including branches of the U.S. government, major corporations, national laboratories, and universities.

At the time of this publication, RSA technology is licensed by about 350 companies. The estimated installed base of RSA encryption engines is around 300 million, making it by far the most widely used public-key cryptosystem in the world. This figure is expected to grow rapidly as the Internet and the World Wide Web expand. For a list of RSA licensees, see http://www.rsa.com/html/licensees.html.

Question 3.1.10. 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 5.3.2). RSA is part of the Society for Worldwide Interbank Financial Telecommunications (SWIFT) standard, the French financial industry's ETEBAC 5 standard, the ANSI X9.31 rDSA standard and the X9.44 draft standard for the U.S. banking industry (see Question 5.3.1). The Australian key management standard, AS2805.6.5.3, also specifies RSA.

RSA is found in Internet standards and proposed protocols including S/MIME (see Question 5.1.1), IPSec (see Question 5.1.4), and TLS, the Internet standards-track successor to SSL, as well as the PKCS standard (see Question 5.3.3) for the software industry. The OSI Implementers' Workshop (OIW) has issued implementers' agreements referring to PKCS (see Question 5.3.3), which includes RSA.

A number of other standards are currently being developed and will be announced over the next few 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].

Question 3.1.11. 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 (see Question 5.3.1) to support the federal requirement of using the Digital Signature Standard (see Question 3.4.1). They then modified X9.30 to X9.31 with the emphasis on RSA digital signatures to support the de facto 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. A digital signature is the exact tool necessary to convert the most essential paper-based documents to digital electronic media. Digital signatures make it possible for passports, college transcripts, wills, leases, checks and voter registration forms to exist in the electronic form; any paper version would just be a "copy" of the electronic original. The accepted standard for digital signatures has enabled all of this to happen.

Question 3.2.1. What is DES?

DES, an acronym for the Data Encryption Standard, is the name of the Federal Information Processing Standard (FIPS) 46-1, which describes the data encryption algorithm (DEA). The DEA is also defined in the ANSI standard X9.32. Originally developed by IBM and known as Lucifer, the NSA and the National Bureau of Standards (NBS, now the National Institute of Standards and Technology, NIST) played a substantial role in the final stages of development. The DEA, often called DES, has been extensively studied since its publication and is the best known and widely used symmetric algorithm in the world.

The DEA has a 64-bit block size (see Question 2.1.4) and uses a 56-bit key during execution (8 parity bits are stripped off from the full 64-bit key). The DEA is a symmetric cryptosystem, specifically a 16-round Feistel cipher (see Question 2.1.4) and was originally designed for implementation in hardware. When used for communication, both sender and receiver must know the same secret key, which can be used to encrypt and decrypt the message, or to generate and verify a message authentication code (MAC). The DEA 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 2.1.3).

NIST (see Question 6.2.1) has recertified DES (FIPS 46-1) every five years; DES was last recertified in 1993, by default. NIST has indicated, however, it will not recertify DES again. The development of AES, the Advanced Encryption Standard (see Question 3.3.1) is underway. AES will replace DES.

Question 3.2.2. Has DES been broken?

No easy attack on DES has been discovered, despite the efforts of researchers over many years. The obvious method of attack is a brute-force exhaustive search of the key space; this process takes 255 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 tradeoff that allows improvement over exhaustive search if memory space is plentiful. These ideas fostered doubts about the security of DES. There were also accusations the NSA (see Question 6.2.2) had intentionally weakened DES. Despite these suspicions, no feasible way to break DES faster than exhaustive search (see Question 2.4.3) has been discovered. The lost of a specialized computer to perform exhaustive search (requiring 3.5 hours on average) has been estimated by Wiener at one million dollars [Wie94]. This estimate was recently updated by Wiener [Wie98] to give an average time of 35 minutes for the same cost machine.

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 2.4.5). This attack requires the encryption of 247 chosen plaintexts (see Question 2.4.2); 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 they consider DES secure.

More recently Matsui [Mat94] has developed another attack, known as linear cryptanalysis (see Question 2.4.5). By means of this method, a DES key can be recovered by the analysis of 243 known plaintexts. The first experimental cryptanalysis of DES, based on Matsui's discovery, was successfully achieved in an attack requiring 50 days on 12 HP 9735 workstations. Clearly, this attack is still impractical.

Most recently, a DES cracking machine was used to recover a DES key in 56 hours [http://www.rsa.com/rsalabs/des2/].

The consensus of the cryptographic community is that DES, if not currently insecure, will soon be insecure, simply because 56 bit keys are becoming vulnerable to exhaustive search (see Question 2.4.4). Starting in November 1998, DES will no longer be allowed for US government use. Triple-DES (see Question 3.2.6) will be used to replace it until AES (see Question 3.3.1) is ready for general use.

Question 3.2.3. 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 from the sender to the receiver. Use of RSA (see Question 3.1.1) 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 might employ a master DES key that encrypts the list of DES keys used to encrypt the files; one can then change the master key frequently without much effort. Since the master key provides a more attractive point of attack than the individual DES keys used on a per file basis, it might be prudent to use triple-DES (see Question 3.2.6) as the encryption mechanism for protecting the file encryption keys.

DES can be used for encryption in several officially defined modes (see Question 2.1.4), 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 bitwise exclusive-ORed with the previous ciphertext block before being encrypted with the DES key. Thus, the encryption of each block depends on previous blocks and the same 64-bit plaintext block can encrypt to different ciphertext blocks 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]. The OFB mode essentially allows DES to be used as a stream cipher.

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 3.2.6).

Question 3.2.4. Should one test for weak keys in DES?

DES has four weak keys k for which $E_k(E_k(m)) = m$ (see Question 2.4.5). There are also twelve semi-weak keys which come in pairs k_1 and k_2 and are such that $E_{k_1}(E_{k_2}(m)) = m$.

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, weak keys 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.

Question 3.2.5. Is DES a group?

The question here is whether, for any message m and two arbitrary keys k_1 and k_2, there is always a third key k such that $E_k(m) = E_{k_1}(E_{k_2}(m))$. If this were the case, the set of all keys would form a mathematical object called a group, where the composition law on k_1 and k_2 yields k. This would be very harmful to the security of DES, as it would enable a *meet-in-the-middle attack* whereby a DES key could be found in about 2^{28} operations, rather than the usual 2^{56} operations (see [KRS88]). It would also render multiple DES encryption useless, since encrypting twice with two different keys would be the same as encrypting once with a third key.

However, DES is not a group. This issue, while strongly supported by initial evidence, was finally settled in 1993 [CW93]. The result seems to imply that techniques such as triple encryption do in fact increase the security of DES.

Question 3.2.6. 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 input data (in this case the single-DES key) is, in effect encrypted three times. There are of course a variety of ways of doing this; we will explore these ways below.

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 as 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}	known plaintext
triple	2	2^{56}	2^{56}	2^{56} known 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

Like all block ciphers, triple-DES can be used in a variety of modes. The ANSI X9.52 standard (see Question 5.3.1) details the different ways in which triple-DES might be used and is expected to be completed during 1998.

Question 3.2.7. What is DES-*X*?

DESX is a strengthened variant of DES supported by RSA Data Security's toolkits (see Question 5.2.3). The difference between DES and DESX is that, in DESX, the input plaintext is bitwise XORed with 64 bits of additional key material before encryption with DES and the output is also bitwise XORed with another 64 bits of key material. The security of DESX against differential and linear attack (see Question 2.4.5) appears to be equivalent to that of DES with independent subkeys (see Question 3.2.8) so there is not a great increase in security with regards to these attacks. However the main motivation for DESX was in providing a computationally simple way to dramatically improve on the resistance of DES to exhaustive key search attacks. This improved security was demonstrated in a formal manner by Killian and Rogaway [RK96] and Rogaway [Rog96]. The DESX construction is due to Rivest.

Question 3.2.8. What are some other DES variants?

G-DES is a variant on DES 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].

Another variant of DES uses 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 2.4.5) than ordinary DES. Biham estimated 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 [Bih95].

Question 3.3.1. What is the AES?

The AES is the Advanced Encryption Standard. This block cipher is intended to become a FIPS (see Question 6.2.1) standard and will replace DES. While reports over the last few years of the demise of DES have been greatly exaggerated, most now agree this venerable cipher is approaching the end of its useful life. DES will not be reaffirmed as a federal standard after 1998. On January 2, 1997 the AES initiative was announced and on September 12, 1997 the public was invited to propose suitable block ciphers as candidates for the AES. NIST is looking for a cipher that will remain secure well into the next century. For more information see http://csrc.nist.gov/encryption/aes/aes_home.htm.

Question 3.3.2. What are some candidates for the AES?

There is considerable interest in the AES initiative and 15 candidates were accepted for consideration in the first round. Among these were close variants of some of the more popular and trusted algorithms currently available, such as CAST, RC5 and SAFER-SK. Other good candidates from well-respected cryptographers were also submitted. The reason for close variants being proposed rather than the original ciphers is that one of the criteria for the AES submission is the ability to support 128-bit blocks of plaintext. Most ciphers were developed with an eye to providing a drop-in replacement for DES and, as a result, were often limited to having a 64-bit block size.

Question 3.3.3. What is the schedule for the AES?

It would be surprising if the process for choosing something as important as a block cipher standard for the next 20-30 years (which is the intended lifetime of the AES) were not long and involved. June 15th, 1998 was the last day to submit an algorithm. Following that, there will be a period of review before five candidates will be chosen for further, more involved scrutiny. From these five, it is intended that the AES will be chosen. It is unlikely that the process will be completed by the year 2000. For more information on the progress of the AES effort, please visit the NIST website at the following URL: http://csrc.ncsl.nist.gov/encryption/aes/aes_home.htm.

Question 3.4.1. What are DSA and DSS?

The National Institute of Standards and Technology (NIST) (see Question 6.2.1) published the Digital Signature Algorithm (DSA) in the Digital Signature Standard (DSS), which is a part of the U.S. government's Capstone project (see Question 6.2.3). DSS was selected by NIST, in cooperation with the NSA (see Question 6.2.2), 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 2.3.7) and is related to signature schemes that were proposed by Schnorr [Sch90] and ElGamal (see Question 3.6.8). While RSA can be used for both encryption and digitial signatures (see Question 2.2.2) the DSA can only be used to provide digital signatures. For a detailed description of DSA, see [NIS94b] or [NIS92].

In DSA, signature generation is faster than signature verification, whereas with RSA, signature verification is very much faster than signature generation (if the public and private exponents, respectively, are chosen for this property, which is the usual case). It might be claimed that it is advantageous for signing to be the faster operation, but since in many applications a piece of digital information is signed once, but verified often, it may well be more advantageous to have faster verification. The trade-offs and issues involved have been explored by Wiener [Wie98]. There has been work by many authors including Naccache et al [NMR94] on developing techniques to improve the efficiency of DSA, both for signing and verification.

Although several aspects of DSA have been criticized since its announcement, it is being incorporated into a number of systems and specifications. Initial criticism focused on a few main issues: it lacked the flexibility of the RSA cryptosystem; verification of signatures with DSA was too slow; the existence of a second authentication mechanism was likely to cause hardship to computer hardware and software vendors, who had already standardized on RSA; and that the process by which NIST chose DSA was too secretive and arbitrary, with too much influence wielded by the NSA. Other criticisms more related to the security of the scheme 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].

Question 3.4.2. Is DSA secure?

The Digital Signature Standard (see Question 3.4.1) 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. DSA is, at present, considered to be secure with 1024-bit keys.

DSA makes use of computation of 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]. No efficient attacks have yet been reported on this form of the discrete logarithm 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 and easily avoided if proper key-generation procedures are followed [SB93].

Question 3.5.1. What are elliptic curve cryptosystems?

Elliptic curve cryptosystems were first proposed independently by Victor Miller [Mil86] and Neal Koblitz [Kob87] in the mid-1980s. At a high level, they are analogs of existing public-key cryptosystems in which modular arithmetic is replaced by operations defined over elliptic curves (see Question 2.3.10). The elliptic curve cryptosystems that have appeared in the literature can be classified into two categories according to whether they are analogs to RSA or discrete logarithm based systems.

Just as in all public-key cryptosystems, the security of elliptic curve cryptosystems relies on the underlying hard mathematical problems (see Question 2.3.1). It turns out that elliptic curve analogs of RSA are mainly of academic interest and offer no practical advantage over ordinary RSA, since their security is based on the same underlying problem as RSA, namely integer factorization. The situation is quite different with elliptic curve variants of discrete logarithm based systems (see Question 2.3.7). The security of such systems depends on the following hard problem: Given two points G and Y on an elliptic curve such that $Y = kG$ (i.*e*., Y is G added to itself k times), find the integer k. This problem is commonly referred to as the "elliptic curve discrete logarithm problem."

Presently, the methods for computing general elliptic curve discrete logs are much less efficient than those for factoring or computing conventional discrete logs. As a result, shorter key sizes can be used to achieve the same security of conventional public-key cryptosystems, which might lead to better memory requirements and improved performance. One can easily construct elliptic curve encryption, signature, and key agreement schemes by making analogs of ElGamal, DSA, and Diffie-Hellman. These variants appear to offer certain implementation advantages over the original schemes, and they have recently drawn more and more attention from both the academic community and the industry.

For more information on elliptic curve cryptosystems, see the survey article by Matt Robshaw and Yiqun Lisa Yin [RY97] or the *CryptoBytes* article by Alfred Menezes [Men95].

Question 3.5.2. Are elliptic curve cryptosystems secure?

In general, the best attacks on the elliptic curve discrete logarithm problems have been general brute-force methods. The current lack of more specific attacks means that shorter key sizes for elliptic cryptosystems appear to give similar security as much larger keys that might be used in cryptosystems based on the discrete logarithm problem and integer factorization. For certain choices of elliptic curves there do exist more efficient attacks. Menezes, Okamoto, and Vanstone [MOV90] have been able to reduce the elliptic curve discrete logarithm problem to the traditional discrete logarithm problem for certain curves, thereby necessitating the same size keys as is used in more traditional public key systems. However these cases are readily classified and easily avoided.

In 1997, elliptic curve cryptography began to receive a lot more attention; by the middle of 1998, there were no major developments as to the security of these cryptosystems. The longer this situation continues, the more confidence will grow that they really do offer as much security as currently appears. However, a sizeable group of very respected researchers have some doubts as to whether this situation will remain unchanged for many years. In particular, there is some evidence that the use of special elliptic curves, sometimes known as Koblitz curves, which provide very fast implementations, might allow new specialized attacks. As a starting point, the basic brute-force attacks can be improved when attacking these curves [Wie98]. While RSA Laboratories believes that continued research into elliptic curve cryptosystems might eventually create the same level of wide-spread trust as is enjoyed by other public-key techniques (provided there are no upsets!), the use of special purpose curves will most likely always be viewed with extreme skepticism.

Question 3.5.3. Are elliptic curve cryptosystems widely used?

Elliptic curve cryptosystems have emerged as a promising new area in public-key cryptography in recent years due to their potential for offering similar security to established public-key cryptosystems with reduced key sizes. Improvements in various aspects of implementation, including the generation of elliptic curves, have made elliptic curve cryptography more practical than when it was first introduced in the mid 80's.

Elliptic curve cryptosystems are especially useful in applications for which memory, bandwidth, or computational power is limited. It is expected that the use of elliptic curve cryptosystems in these special areas will continue to grow in the future.

Standards efforts for elliptic curve cryptography are well underway. X9.*F*.1, an ANSI-accredited standards committee for the financial services industry is developing two standards: ANSI X9.62 for digital signatures and ANSI X9.63 for key agreement and key transport. IEEE P1363 is working on a general reference for public-key techniques from several families, including elliptic curves.

Question 3.5.4. How do elliptic curve cryptosystems compare with other cryptosystems?

The main attraction of elliptic curve cryptosystems over other public-key cryptosystems is the fact that they are based on a different, hard problem. This may lead to smaller key sizes and better performance in certain public key operations for the same level of security.

Very roughly speaking, when this FAQ was published elliptic curve cryptosystems with a 160-bit key offer the same security of RSA and discrete logarithm based systems with a 1024-bit key. As a result, the length of the public key and private key is much shorter in elliptic curve cryptosystems. In terms of speed, however, it is quite difficult to give a quantitative comparison, partly because of the various optimization techniques one can apply to different systems. It is perhaps fair to say the following: Elliptic curve cryptosystems are faster than the corresponding discrete logarithm based systems. Elliptic curve cryptosystems are faster than RSA in signing and decryption, but slower than RSA in signature verification and encryption. For more detailed comparisons, see the survey article by Matt Robshaw and Yiqun Lisa Yin [RY97].

With academic advances in attacking different hard mathematical problems both the security estimates for various key sizes in different systems and the performance comparisons between systems are likely to change.

Question 3.6.1. 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 groundbreaking 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, with the following property: for every number n between 1 and p-1 inclusive, there is a power k of g such that $g^k = n$ mod p.

Suppose 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. Both a and b are drawn from the set of integers [1, ..., p-2]. 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 $g^{ab} = (g^b)^a$ mod p, and Bob computes $g^{ba} = (g^a)^b$ mod p. Since $g^{ab} = g^{ba} = k$, 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 is equivalent to computing discrete logarithms under certain assumptions.

The Diffie-Hellman key exchange is vulnerable to a man-in-the-middle 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 other party. This vulnerability is present because Diffie-Hellman key exchange does not authenticate the participants. Possible solutions include the use of digital signatures and other protocol variants.

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 man-in-the-middle attack on the Diffie-Hellman key agreement protocol. The immunity is achieved by allowing the two parties to authenticate themselves to each other by the use of digital signatures (see Question 2.2.2) and public key certificates (see Question 4.1.3.10).

Roughly speaking, the basic idea is as follows. Prior to execution of the protocol, the two parties Alice and Bob each obtain a public/private key pair and a certificate for the public key. During the protocol, Alice computes a signature on certain messages, covering the public value g^a mod p. Bob 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 man-in-the-middle attack.

In recent years, the original Diffie-Hellman protocol been understood to be an example of a much more general cryptographic technique, the common element being the derivation of a shared secret value (i.*e*., key) from one party's public key and another party's private key. The parties' key pairs may be generated anew at each run of the protocol, as in the original Diffie-Hellman protocol. The public keys may be certified, so that the parties can be authenticated and there may be a combination of these attributes. The draft ANSI X9.42 (see Question 5.3.1) illustrates some of these combinations, and a recent paper by Blake-Wilson, Johnson, and Menezes provides some relevant security proofs.

Question 3.6.2. What is RC2?

RC2 is a variable key-size block cipher designed by Ron 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 3.2.1). 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.

An agreement between the Software Publishers Association (SPA) and the United States government gives RC2 and RC4 (see Question 3.6.3) 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.

Question 3.6.3. What is RC4?

RC4 is a stream cipher designed by Rivest for RSA Data Security, Inc. It is a variable key-size stream cipher with byte-oriented operations. The algorithm is based on the use of a random permutation. 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. Independent analysts have scrutinized the algorithm and it is considered secure. The RC4 stream cipher has a special status by which export from the U.S. can often be facilitated.

RC4 is used for file encryption in products such as RSA SecurPC (see Question 5.2.4). It is also used for secure communications, as in the encryption of traffic to and from secure websites using the SSL protocol (see Question 5.1.2).

Question 3.6.4. What is RC5?

RC5 [Riv95] is a fast block cipher designed by Ron Rivest for RSA Data Security in 1994. It is a parameterized algorithm with a variable block size, a variable key size, and a variable number of rounds. Typical choices for the block size will be 32 bits (for experimentation and evaluation purposes only), 64 bits (for use a drop-in replacement for DES) or 128 bits. 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 at all 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: integer addition, bitwise exclusive-or, and variable rotation. The exceptional simplicity of RC5 makes it easy to implement and analyze. Indeed, like RSA, the encryption steps of RC5 can be written on the "back of an envelope".

The heavy use of data-dependent rotations and the mixture of different operations provide the security of RC5. In particular, the use of data-dependent rotations helps defeat differential and linear cryptanalysis (see Question 2.4.5).

In the three years since RC5 was proposed, there have been numerous studies of RC5's security [KY95] [KM96] [BK98] [Sel98]. Each study has provided a greater understanding of how RC5's structure and components contribute to its security. For a summary of known cryptanalytic results, see the survey article by Yiqun Lisa Yin [Yin97]

The US patent office granted the RC5 patent to RSA Data Security, Inc. in May 1997.

Question 3.6.5. What are SHA and SHA-1?

The Secure Hash Algorithm (SHA), the algorithm specified in the Secure Hash Standard (SHS, FIPS PUB 180), was developed by NIST (see Question 6.2.1) [NIS93a]. SHA-1 [NIS94c] is 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 3.6.6). SHA-1 is also described in the ANSI X9.30 (part 2) standard.

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 3.6.6), but the larger message digest makes it more secure against brute-force collision and inversion attacks (see Question 2.4.6). SHA is part of the Capstone project (see Question 6.2.3). For further information on SHA, see [Pre93] and [Rob95c].

Question 3.6.6. 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. 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 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 the 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 by 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 2.1.6), 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, 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. In recent work, Dobbertin (Fast Software Encryption, 1998) has shown that a reduced version of MD4 in which the third round of the compression function is not executed but everything else remains the same, is not one-way. 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 has a slightly different design from that of MD4. Message-digest size, as well as padding requirements, remain the same. Den Boer and Bosselaers [DB94] have found pseudo-collisions for MD5 (see Question 2.4.6). More recent work by Dobbertin has extended the techniques used so effectively in the analysis of MD4 to find collisions for the compression function of MD5 [DB96b]. While stopping short of providing collisions for the hash function in its entirety this is clearly a significant step. For a comparison of these different techniques and their impact the reader is referred to [Rob96].

Van Oorschot and Wiener [VW94] have considered a brute-force search for collisions (see Question 2.1.6) in hash functions, and they estimate 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].

Question 3.6.7. What are some other block ciphers?

IDEA (International Data Encryption Algorithm) [LMM92] is the second version of a block cipher designed and presented by Lai and Massey [LM91]. It is a 64-bit iterative block cipher with a 128-bit key. The encryption process requires eight complex rounds. While the cipher does not have a Feistel structure (see Question 2.1.4), 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. However some of the arithmetic operations used in IDEA are not that fast in software. As a result the speed of IDEA in software is similar to that of DES.

One of the principles used during the design of IDEA was to facilitate analysis of its strength against differential cryptanalysis (see Question 2.4.5) and IDEA is considered to be immune to differential cryptanalysis. Furthermore there are no linear cryptanalytic attacks on IDEA and there are no known algebraic weaknesses in IDEA. The most significant cryptanalytic result is due to Daemen [DGV94], who discovered a large class of 251 weak keys (see Question 2.4.5) for which the use of such a key during encryption could be detected easily and the key recovered. However, since there are 2128 possible keys, this result has no impact on the practical security of the cipher for encryption provided the encryption keys are chosen at random. IDEA is generally considered to be a very secure cipher and both the cipher development and its theoretical basis have been openly and widely discussed.

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, 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 2.4.5) when the number of rounds is greater than six. However, 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 (though one joke has it that SK really stands for "Stop Knudsen" a wise precaution in the design of any block cipher). 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.

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 225 known plaintexts [OA94], and other improvements [KR95a] followed. In the wake of these numerous attacks, FEAL and its derivatives should be considered insecure.

Skipjack is the encryption algorithm contained in the Clipper chip (see Question 6.2.4), designed by the NSA (see Question 6.2.2). It uses an 80-bit key to encrypt 64-bit blocks of data. Skipjack is expected to be more secure than DES (see Question 3.2.1) in the absence of any analytic attack since it uses 80-bit keys. By contrast, DES uses 56-bit keys.

Initially the details of Skipjack were classified and the decision not to make the details of the algorithm publicly available was widely criticized. Some people were suspicious that Skipjack might not be secure, either due to an oversight by its designers, or by the deliberate introduction of a secret trapdoor. Since Skipjack was not public, it could not be widely scrutinized and there was little public confidence in the cipher.

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 Skipjack was secure.

In June of 1998 Skipjack was declassified by the NSA. Early cryptanalysis has failed to find any substantial weakness in the cipher.

Blowfish is a 64-bit block cipher developed by Schneier [Sch93]. It is a Feistel cipher (see Question 2.1.4) and each round consists of a key-dependent permutation and a key-and-data-dependent substitution. All operations are based on exclusive-ors 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]; among the results were the discoveries of existence of certain weak keys (see Question 2.4.5), 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.

Many of the block ciphers proposed in recent years, including those listed above, were developed (at least in part) as successors to DES (see Question 3.2.1). Many other proposed successors have appeared as candidates for the Advanced Encryption Standard, AES (see Question 3.3.1).

Question 3.6.8. What are some other public-key cryptosystems?

The ElGamal system [Elg85] is a public-key cryptosystem based on the discrete logarithm problem. It consists of both encryption and signature variants. The encryption algorithm is similar in nature to the Diffie-Hellman key agreement protocol (see Question 3.6.1).

The system parameters for ElGamal cryptosystem consist of a prime p and an integer g, whose powers modulo p generate a large number of elements (it is not necessary for g to be a generator, however, it is ideal), 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

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

where $\oplus$ denotes the bit-wise exclusive-or. Bob sends (y_1, y_2) to Alice. Upon receiving the ciphertext, Alice computes

$$m = (y_1 a \bmod p) \oplus y_2 .$$

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 3.1.1). DSA, The Digital Signature Algorithm (see Question 3.4.1), is based in part on the ElGamal signature algorithm.

Analysis based on the best available algorithms for both factoring and discrete logarithms show 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 generally unimportant if the cryptosystem is used only for exchange of secret keys.

The Merkle-Hellman knapsack cryptosystem [MH78] is a public-key cryptosystem 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. Shamir broke the single iteration knapsack cryptosystem introduced in 1978 [Sha84]. Merkle then published the multiple-iteration knapsack problem broken by Brickell [Bri85]. Merkle offered from his own pocket 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. 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. Eventually, 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 could be extended to attack Damgård’s knapsack hash function [Dam90].

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 3.6.9), Diffie-Hellman (see Question 3.6.1), and RSA (see Question 3.1.1) over Lucas sequences. LUCELG is the Lucas sequence analog of ElGamal, while LUCDIF and LUCRSA are the Diffie-Hellman and RSA analogs, respectively. 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.

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 (around half a megabit), substantial expansion of data, and possibly a certain similarity to the knapsack cryptosystem. Gabidulin, Paramonov, and Tretjakov [GPT91] proposed a modification of the McEliece cryptosystem by replacing Goppa codes with a different algebraic code and claimed the new version was more secure than the original system. However, Gibson [Gib93] later showed there was not really any advantage to the new version.

Question 3.6.9. What are some other signature schemes?

Merkle proposed a digital signature scheme based on both one-time signatures (see Question 7.7) and a hash function (see Question 2.1.6); this 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 to a node in a tree, with each node consisting of the verification parameters 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.

The Rabin signature scheme [Rab79] is a variant of the RSA signature scheme (see Question 3.1.1). 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 provably difficult otherwise. Anyone who can consistently forge the signature for a modulus n 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.)

Question 3.6.10. What are some other stream ciphers?

There are a number of alternative stream ciphers that have been proposed in cryptographic literature as well as a large number that appear in implementations and products world-wide. Many are based on the use of LFSRs (Linear Feedback Shift Registers - see Question 2.1.5), since such ciphers tend to be more amenable to analysis and it is easier to assess the security they offer.

Rueppel suggests 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. The second approach is that of system-theoretic design. In essence, the cryptographer designs the cipher along established guidelines that ensure 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 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 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.

A recent example of a stream cipher designed by a system-theoretic approach is the Software-optimized Encryption Algorithm (SEAL), which 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 3.6.5). 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.

A design that has system-theoretic as well as complexity-theoretic aspects is given by Aiello, Rajagopalan, and Venkatesan [ARV95]. The design, commonly referred to as "VRA," derives a fast stream cipher from an arbitrary secure block cipher. VRA is described as a pseudorandom generator (see Question 2.5.2), not a stream cipher, but the two concepts are closely connected, since a pseudorandom generator can produce a (pseudo) one-time pad for encryption.

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 an RSA Laboratories technical report [Rob95b].

Question 3.6.11. 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:

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

By building a hash function around a block cipher, a designer aims to leverage the security of a well-trusted block cipher such as DES (see Question 3.2.1) to obtain a well-trusted hash function. The so-called Davies-Meyer hash function [Pre93] is an example of a hash function built around the use of DES.

The purpose of employing modular arithmetic in the second class of hash functions is to save on implementation costs. A hash function is generally used in conjunction with a digital signature algorithm which itself makes use of 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 for use today.

The hash functions in the third class, with their so-called "dedicated" design, tend to be fast, achieving a considerable advantage over algorithms that are based around the use of a block cipher. MD4 is an early example of a popular hash function with such a design. Although MD4 is no longer considered secure for most cryptographic applications, most new dedicated hash functions make use of the same design principles as MD4 in a strengthened version. Their strength varies depending on the techniques, or combinations of techniques, employed in their design. Dedicated hash functions in current use include MD5 and SHA-1 (see Questions 3.6.5 and 3.6.6), as well as RIPEMD-160 [DBP96] and HAVAL [ZPS93].

Question 3.6.12. What are some secret sharing schemes?

Shamir's secret sharing scheme [Sha79] is a threshold scheme based on polynomial interpolation. To allow any m out of n people to construct a given secret, an (m - 1)-degree polynomial over the finite field GF(q)

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

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 not equal to 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 2.1.9).

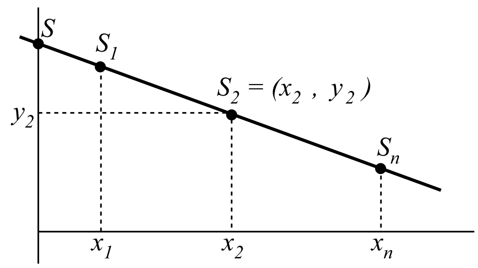


Figure 10. Shamir's secret sharing scheme

A special case where m = 2 (i.*e*., two shares are required for retrieval of the secret) is given in Figure 10. 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.

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 the secret is a point on his hyperplane. Nevertheless, this scheme can be modified to achieve perfect security [Sim92].

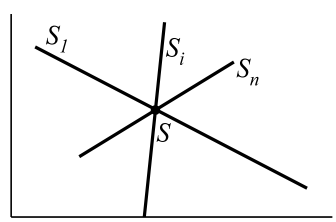


Figure 11. Blakely's scheme

A special case of Blakley's scheme is shown in Figure 11. 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.

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.

Section 4.1: Key Management

Question 4.1.1. What is key management?

Key management deals with the secure generation, distribution, and storage of keys. Secure methods of key management are extremely important. Once a key is randomly generated (see Question 4.1.2.2), it must remain secret to avoid unfortunate mishaps (such as impersonation). In practice, most attacks on public key systems will probably be aimed at the key management level, rather than at the cryptographic algorithm itself.

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 public key. Users must be able to legitimately obtain 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 4.1.3.10). Certificates must be unforgeable. The issuance of certificates must proceed in a secure way, impervious to attack. In particular, the issuer must authenticate the identity and the public key of an individual before issuing a certificate to that individual.

If someone's private key is lost or compromised, others must be made aware of this, so 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 no intruder can obtain them, yet the keys must be readily accessible for legitimate use. Keys need to be valid only until a specified expiration date but the expiration date must be chosen properly and publicized in an authenticated channel.

Question 4.1.2.1. What key size should be used?

The key size that should be used in a particular application of cryptography depends on two things. First of all, the value of the key is an important consideration. Secondly, the actual key size depends on what cryptographic algorithm is being used.

Below is a table of key sizes recommended as of this printing, for block ciphers, RSA, elliptic curve, and DSA. There are four different "grades," which refer to the strength of the protection. Export grade or nominal grade gives little real protection, but this is the approximate level of cryptography approved for export from the US. Personal grade is recommended for keys that are not very important, such as those which protect one person's personal *e*-mail, those which serve as "session keys" for low-importance transactions, et cetera. These should provide plenty of protection relative to how much they are worth to break. Commercial grade is recommended for information that is actually valuable, but not extremely sensitive. Military grade is recommended for information that is truly sensitive and must be kept secret at any cost. Military grade is also recommended for Certifying Authorities (see Question 4.1.3.12).

Keep in mind that items in the same row are not necessarily equal in security, but approximately equal.

	Block Cipher	RSA	Elliptic Curve	DSA
Export Grade	40 bits	512 bit modulus	80 bit curve	512 / 80 bits
Personal Grade	56 / 64 bits	768 bit modulus	136 bit curve	768 / 136 bits
Commercial Grade	128 bits	1024 bit modulus	160 bit curve	1024 / 160 bits
Military Grade	160 bits	2048 bit modulus	200 bit curve	2048 / 200 bits

Table 2

Notes: The Elliptic Curve key size refers to the minimum order of the base point on the elliptic curve (which should be slightly smaller than the field size). The DSA key sizes refer to the size of the modulus and the minimum size of a large subgroup.

Question 4.1.2.2. How does one find random numbers for keys?

Whether using a secret-key cryptosystem or a public-key cryptosystem, one needs a good source of random numbers for key generation. The main features of a good source are that it produces numbers that are unknown and unpredictable by potential adversaries. Random numbers obtained from a physical process are in principle the best, since many physical processes appear truly random. 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 inter-key stroke timings measured in microseconds. 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 unpredictable. By whichever method they are generated, the random numbers may still contain some correlation, thus preventing sufficient statistical randomness. Therefore, it is best to run them through a good hash function (see Question 2.1.6) before actually using them [ECS94].

Another approach is to use a pseudorandom number generator fed by a random seed. The primary difference between random and pseudorandom numbers is that pseudorandom numbers are necessarily periodic whereas truly random numbers are not. Since pseudorandom number generators are deterministic algorithms, it is important to find one that is cryptographically secure and also to use a good 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 2.3.7). Other generators perhaps based on DES (see Question 3.2.1) or a hash function (see Question 2.1.6) can also be considered to satisfy this definition, under reasonable assumptions.

A summary of 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. After generating the primes, and hence the modulus (see Question 3.1.1), one can simply choose an arbitrary value (subject to the standard constraints) for the public exponent, which then determines the private exponent.

Question 4.1.2.3. 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 (see Question 2.4.1). 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. Thus keys should have a limited lifetime. If you suspect that an attacker may have obtained your key, the key should be considered compromised, and its use discontinued.

Research in cryptanalysis can lead to possible attacks against either the key or the algorithm. For example, recommended 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. The recommended key length depends on the expected lifetime of the key. Temporary keys, which are valid for a day or less, may be as short as 512 bits. Keys used to sign long-term contracts for example, should be longer, say, 1024 bits or more.

Another reason for limiting the lifetime of a key is to minimize the damage from a compromised key. It is unlikely a user will discover an attacker has compromised his or her key 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:

1. key generation and possibly registration (for a public key)
2. key distribution
3. key activation/deactivation
4. key replacement or key update
5. key revocation
6. key termination, involving destruction or possibly archival

Question 4.1.3.1. What is a PKI?

A public key infrastructure (PKI) consists of protocols, services, and standards supporting applications of public-key cryptography. The term PKI, which is relatively recent, is defined variously in current literature. PKI sometimes refers simply to a trust hierarchy based on public key certificates [1], and in other contexts embraces encryption and digital signature services provided to end-user applications as well [OG97]. A middle view is that a PKI includes services and protocols for managing public keys, often through the use of Certification Authority (CA) and Registration Authority (RA) components, but not necessarily for performing cryptographic operations with the keys.

Among the services likely to be found in a PKI are the following:

- key registration: issuing a new certificate for a public key
- certificate revocation: canceling a previously issued certificate
- key selection: obtaining a party's public key
- trust evaluation: determining whether a certificate is valid and what operations it authorizes

Key recovery has also been suggested as a possible aspect of a PKI.

There is no single pervasive public key infrastructure today, though efforts to define a PKI generally presume there will eventually be one, or, increasingly, that multiple independent PKIs will evolve with varying degrees of coexistence and interoperability. In this sense, the PKI today can be viewed akin to local and wide-area networks in the 1980's, before there was widespread connectivity via the Internet. As a result of this view toward a global PKI, certificate formats and trust mechanisms are defined in an open and scaleable manner, but with usage profiles corresponding to trust and policy requirements of particular customer and application environments. For instance, it is usually accepted that there will be multiple "root" or "top-level" certificate authorities in a global PKI, not just one "root," although in a local PKI there may be only one root. Accordingly, protocols are defined with provision for specifying which roots are trusted by a given application or user.

Efforts to define a PKI today are underway in several governments as well as standards organizations. The U.S. Department of the Treasury and NIST both have PKI programs [2,3], as do Canada [4] and the United Kingdom [5]. NIST has published an interoperability profile for PKI components [BDN97]; it specifies algorithms and certificate formats that certification authorities should support. Standards bodies working on a PKI includes the IETF's PKIX and SPKI working groups [6,7] and The Open Group [8].

Most PKI definitions are based on *X*.509 certificates, with the notable exception of the IETF's SPKI.

[1] PKI - PC Webopaedia Definitions and Links. http://www.sandybay.com/pc-web/PKI.htm.
[2] Government Information Technology Services, Federal Public key Infrastructure. http://gits-sec.treas.gov/fpki.htm.
[3] NIST Public key Infrastructure Program. http://csrc.ncsl.nist.gov/pki/.
[4] The Government of Canada Public key Infrastructure. http://www.cse.dnd.ca/cse/english/gov.html.
[5] The Open Group Public key Infrastructure, Latest Proposals for an HMG PKI. http://www.opengroup.org/public/tech/security/pki/cki/.
[6] Public key Infrastructure (*X*.509) (pkix) working group. http://www.ietf.org/html.charters/pkix-charter.html.
[7] Simple Public key Infrastructure (spki) working group. http://www.ietf.org/html.charters/spki-charter.html.
[8] The Open Group Public key Infrastructure. http://www.opengroup.org/security/pki/.

Question 4.1.3.2. 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. In fact, it is advisable to use separate key pairs for signing messages and receiving encrypted messages. As another 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, webservers (websites) and printers, as well as organizational entities such as a corporate department, a hotel registration desk, or a university registrar's office. Key pairs allow people and other entities to authenticate (see Question 2.2.2) and encrypt messages.

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 non-escrowed key pair for authentication. The lengths of the encryption and authentication key pairs may be varied according to the desired security.

Question 4.1.3.3. 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, often do not allow local key generation, but instead use a central server to generate keys.

Once a key has been generated, the user must register his or her public key with some central administration, called a Certifying Authority (CA). The CA returns to the user a certificate attesting to the validity of the user's public key along with other information (see Question 4.1.3.10 through Question 4.1.3.12). 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.

Question 4.1.3.4. Should a key pair be shared among users?

Keyword: impersonation, public key, private key, key sharing

Users who share a private key can impersonate one another (i.*e*., sign messages as one another and decrypt messages intended for one another), so in general, private keys should not be shared among users. However, some parts of a key may be shared, depending on the algorithm (see Question 3.6.12).

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 3.6.1, Question 3.4.1, and Question 3.6.8), a group of people can share a set of system parameters, which can lead to simpler implementations. This is also true for systems based on elliptic curve discrete logarithms. 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 set of system parameters 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 set common parameters than if every key had a distinct set of parameters.

Question 4.1.3.5. 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 4.1.2.3). The time to expiration must therefore be much shorter than the expected time for cryptanalysis. That is, 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. The appropriate key size is determined by the validity period, together with the value of the information protected by the key and the estimated strength of an expected attacker. In a certificate (see Question 4.1.3.10), 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 in which it is important that a signed document be considered valid for a much longer period of time. Question 7.11 discusses digital time stamping as a way to achieve this.

After expiration, the old key should be destroyed to preserve the security of old messages. At this point, 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 (see Question 4.1.2.1 for recent key length recommendations). 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 makes it prudent to replace expired keys with newer, longer keys every few years. Key replacement enables one to take advantage of any 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 2.3.5).

Question 4.1.3.6. What happens if my key is lost?

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. The CA (see Question 4.1.3.12) must be notified immediately so that the key can be revoked and placed on a certificate revocation list (CRL, see Question 4.1.3.16) to prevent any illegitimate use if the key is found or recovered by an adversary. Loss of a private key 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 should also obtain a new key right away to minimize the number of messages people send you that are encrypted under your old key, since these can no longer be read.

Question 4.1.3.7. 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 should assume the attacker can read any encrypted messages sent to you under the corresponding public key, and forge your signature on documents as long as others continue to accept that public key as yours. The seriousness of these consequences underscores the importance of protecting your private key with extremely strong mechanisms (see Question 4.1.3.8).

You must immediately notify any certifying authorities for the public keys and have your public key placed on a certificate revocation list (see Question 4.1.3.16); this will inform people that the private key has been compromised and the public key has been revoked. Then generate a new key pair and obtain a new certificate for the public key. You may wish to use the new private key to re-sign documents you had signed with the compromised private key, though documents that had been time stamped as well as signed might still be valid (see Question 7.11). You should also change the way you store your private key to prevent a compromise of the new key.

Question 4.1.3.8. 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 4.1.3.7). The measures taken to protect a private key must be at least equal to the required security of the messages encrypted with that key. In general, a private key should never be stored anywhere in plaintext form. The simplest storage mechanism is to encrypt a private key under a password and store the result on a disk. However, passwords are sometimes very easily guessed; when this scheme is followed, a password should be chosen very carefully since the security is tied directly to the password.

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 the user can remove and take with her when she has finished using a particular computer. Private keys may also be stored on portable hardware, such as a smart card. Users with extremely high security needs, such as certifying authorities, should use tamper-resistant devices to protect their private keys (see Question 4.1.3.13).

Question 4.1.3.9. How do I find someone else's public key?

Suppose Alice wants to find Bob's public key. There are several possible ways of doing this. She could call him up and ask him to send his public key via *e*-mail. She could request it via *e*-mail, exchange it in person, as well as many other ways. Since the public key is public knowledge, there is no need to encrypt it while transferring it, though one should verify the authenticity of a public key. A mischievous third party could intercept the transmission, replace Bob's key with his or her own and thereby be able intercept and decrypt messages that are sent from Alice to Bob and encrypted using the "fake" public key. For this reason one should personally verify the key (*e.g.*, this can be done by computing a hash of the key and verifying it with Bob over the phone) or rely on certifying authorities (see Question 4.1.3.12 for more information on certifying authorities). Certifying authorities may provide directory services; if Bob works for company *Z*, Alice could look in the directory kept by Z's certifying authority. Directories must be secure against tampering, so users can be confident a public key listed in the directory actually belongs to the person listed. Otherwise, they might send private, encrypted information to the wrong person or accept documents signed by 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 5.3.2), the directories will contain certificates as well as public keys and the presence of certificates will lower the directories' security needs.

Question 4.1.3.10. 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 specific public key does in fact belong to a specific individual. Certificates help prevent someone from using a phony key to impersonate someone else. In some cases it may be necessary to create a chain of certificates, each one certifying the previous one until the parties involved are confident in the identity in question.

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 5.3.2); 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].

Question 4.1.3.11. How are certificates used?

Certificates are typically used to generate confidence in the legitimacy of a public key. Certificates are essentially digital signatures that protect public keys from forgery, false representation, or alteration. The verification of a signature therefore can include checking the validity of the certificate for the public key involved. Such verification 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 certificate 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. It is interesting to note that there are alternative trust models being pursued by a variety of researchers that avoid this hierarchical approach.

The more familiar the sender is to the receiver of the message, or more precisely, the more trust the receiver places in the claim that the public key really is that of the sender, the less need there is to enclose and verify certificates. If Alice sends messages to Bob every day, Alice can enclose a certificate chain on the first day that Bob verifies. Bob thereafter stores Alice's public key and no more certificates or certificate certifications 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 or more certificates. A good rule of thumb is to enclose just enough of a certificate chain so 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.

Question 4.1.3.12. 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, or a university to its students, or 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 12 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 the request really did come from Alice and that the public key was not modified in transit, 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 their own members and employees. There are also CAs that issue certificates to unaffiliated individuals.

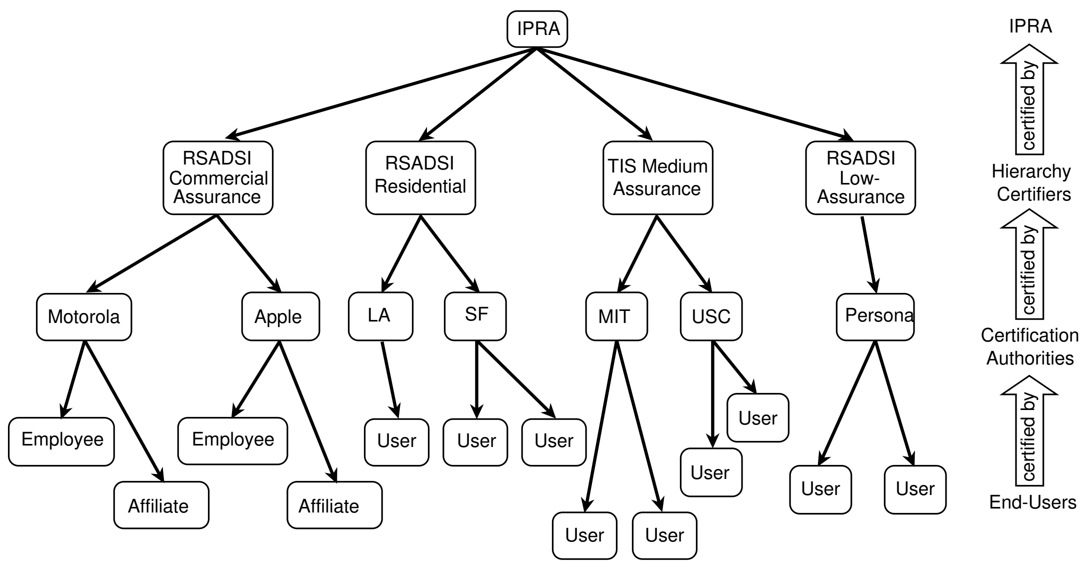


Figure 12. Example of a certification hierarchy.

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 verifiers can attach the appropriate level of confidence to 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.

For more information about certificate-related products, visit the VeriSign, Inc. website; see *http://www.verisign.com/* for more information.

Question 4.1.3.13. How do certifying authorities store their private keys?

It is extremely important that the private keys of certifying authorities (see Question 4.1.3.12) are stored securely. The compromise of this information would allow the generation of certificates for fraudulent public keys. 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 accidentally, then 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 a lost private key can be restored into a new device.
(see Question 4.1.3.6 for a discussion of lost CA private keys).

Question 4.1.3.14. How are certifying authorities susceptible to attack?

One can think of many attacks aimed at certifying authorities (see Question 4.1.3.12) 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 4.1.3.13 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. Time stamps are the solution in this case (see Question 2.2.2). ****

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 so 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.

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.)

Question 4.1.3.15. 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 4.1.3.13 and Question 4.1.3.14.

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 the new CA key. These measures could be relaxed somewhat if the certificates were registered with a digital time stamping service (see Question 7.11). 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.

Question 4.1.3.16. 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 examines 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. A CRL is maintained by a CA, and it provides information about revoked certificates that were issued by that CA. CRLs only list current certificates, since expired certificates should not be accepted in any case: when a revoked certificate’s expiration date occurs, that certificate can be removed from the CRL.

CRLs are usually distributed in one of two ways. In the “pull” model, verifiers download the CRL from the CA, as needed. In the “push” model, the CA sends the CRL to the verifiers at regular intervals. Some systems use a hybrid approach where the CRL is pushed to several intermediate repositories from which the verifiers may retrieve it as needed.

Although CRLs are maintained in a distributed manner, there may be central repositories for CRLs, such as, 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 5.3.2) are more flexible.

Question 4.2.1. 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. 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. Anonymous schemes do not reveal the identity of the customer and are based on blind signature schemes (see Question 7.3). 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 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).

Question 4.2.2. What is iKP?

The Internet Keyed Payments Protocol (iKP) is 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.

iKP is no longer widely in use, however it is the current foundation for SET (see Question 4.2.3).

Additional information on iKP is available from:
http://www.zurich.ibm.com/Technology/Security/extern/ecommerce/iKP.html.

Question 4.2.3. 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 protocols for purchasing goods and services electronically, requesting authorization of payment, and requesting "credentials" (i.*e*. certificates) binding public keys to identities, among other services. Once SET is fully adopted, the necessary confidence in secure electronic transactions will be in place, allowing merchants and customers to partake in electronic commerce.

SET supports DES (see Question 3.2.1) for bulk data encryption and RSA (see Question 3.1.1) for signatures and public key encryption of data encryption keys and bankcard numbers. The RSA public - key encryption employs Optimal Asymmetric Encryption Padding [BR94].

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*.

Question 4.2.4. What is Mondex?

Mondex is a payment system in which currency is stored in smartcards. These smartcards are similar in shape and size to credit cards, and generally permit the storage of sums of money up to several hundred dollars. Money may be transferred from card to card arbitrarily many times and in any chosen amounts. There is no concern about coin sizes, as with traditional currency. The Mondex system also provides a limited amount of anonymity. The system carries with it one of the disadvantages of physical currency: if a Mondex card is lost, the money it contains is also lost. Transfers of funds from card to card are effected with any one of a range of intermediate hardware devices.

The Mondex system relies for its security on a combination of cryptography and tamper-resistant hardware. The protocol for transferring funds from one card to another, for instance, makes use of digital signatures (although Mondex has not yet divulged information about the algorithms employed). Additionally, the system assumes that users cannot tamper with cards, i.*e*., access and alter the balances stored in their cards. The Mondex system is managed by a corporation known as Mondex International Ltd., with a number of associated national franchises. Pilots of the system have been initiated in numerous cities around the world.

For more information on Mondex, visit their website at http://www.mondex.com.

Question 4.2.5. What are micropayments?

Micropayments are payments of small sums of money, generally in denominations smaller than those in which physical currency is available. It is envisioned that sums of as little as 1/1000th of a cent may someday be used to pay for content access or for small quantities of network resources. Conventional electronic payment systems require too much computation to handle such sums with acceptable efficiency. Micropayment systems enable payments of this size to be achieved in a computationally lightweight manner, generally by sacrificing some degree of security.

One example of a micropayment system, proposed by Rivest and Shamir, is known as MicroMint. In MicroMint, a coin consists of a hash collision computed under certain carefully tuned constraints. By investing in extensive computational resources, a mint may compute a number of these coins, and then sell them in batches. MicroMint exploits the efficiency of hash function calculations: any party can quickly verify the legitimacy of coin. While deriving new coins is hard in MicroMint, it is possible for users to re-spend the same coin or to set up an expensive forging operation. The MicroMint system addresses this shortcoming by presuming that it will not be worthwhile for a user to cheat in this manner.

Section 5.1: Security on the Internet

Question 5.1.1. 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 these 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 5.3.3) for digital signatures and encryption. The MIME body section carries a PKCS #7 message, which itself is the result of cryptographic processing on other MIME body sections.

S/MIME has been 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/rsa/S-MIME/.

Question 5.1.2. What is SSL?

The SSL (Secure Sockets 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 (HyperText Transfer Protocol), 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 an optional client authentication. 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 authenticated with the master key. Subsequent data is encrypted and authenticated 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 3.1.1) is used. After the exchange of keys, a number of ciphers are used. These include RC2 (see Question 3.6.2), RC4 (see Question 3.6.3), IDEA (see Question 3.6.7), DES (see Question 3.2.1), and triple-DES (see Question 3.2.6). The MD5 message-digest algorithm (see Question 3.6.6) is also used. The public key certificates follow the *X*.509 syntax (see Question 5.3.3).

Question 5.1.3. What is S/WAN?

The S/WAN (Secure Wide Area Network, pronounced "swan") is an initiative to promote the widespread deployment of Internet-based Virtual Private Networks (VPNs). This is accomplished by adopting a standard specification for implementing IPSec, the security architecture for the Internet Protocol [Atk95a] [Atk95b][Atk95c] [KMS95][MS95a], thereby ensuring interoperability among firewall and TCP/IP products. The use of IPSec allows 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 an 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 and lower-level security than higher-level protocols, such as SSL (see Question 5.1.4). It is expected that higher-level security specifications, including SSL, will be routinely layered on top of S/WAN implementations, and these security specifications will work together.

To guarantee IPSec interoperability, S/WAN defines a common set of algorithms, modes, and options. S/WAN uses RC5 (see Question 3.6.4) at key sizes ranging from 40 bits (for exportability) to 128 bits. S/WAN can also be implemented using DES (see Question 3.2.1).

Question 5.1.4. What is IPSec?

The Internet Engineering Task Force (IETF)'s IP Security Protocol (IPSec) working group is defining a set of specifications for cryptographically-based authentication, integrity, and confidentiality services at the IP datagram layer. These specifications are expected to emerge as Internet Proposed Standard RFCs early in 1998. The IPSec group's results comprise a basis for interoperably secured host-to-host pipes, encapsulated tunnels, and Virtual Public Networks (VPNs), thus providing protection for client protocols residing above the IP layer.

The protocol formats for IPSec's Authentication Header (AH) and IP Encapsulating Security Payload (ESP) are independent of the cryptographic algorithm, although certain algorithm sets are specified as mandatory for support in the interest of interoperability. Similarly, multiple algorithms are supported for key management purposes (establishing session keys for traffic protection), within IPSec's ISAKMP/Oakley framework.

For more info on IPSec: http://www.ietf.org/html.charters/ipsec-charter.html

Question 5.1.5. What is SSH?

SSH, or Secure Shell, is a protocol which permits secure remote access over a network from one computer to another. SSH negotiates and establishes an encrypted connection between an SSH client and an SSH server, authenticating the client and server in any of a variety of ways (some of the possibilities for authentication are RSA; Security Dynamics SecurID tokens; and passwords). That connection can then be used for a variety of purposes, such as creating a secure remote login on the server (effectively replacing commands such as telnet, rlogin, and rsh) or setting up a VPN (Virtual Private Network).

When used for creating secure logins, SSH can be configured to forward X11 connections automatically over the encrypted "tunnel" so as to give the remote user secure access to the SSH server within a full-featured windowing environment. SSH connections and their X11 forwarding can be cascaded to give an authenticated user convenient secure windowed access to a complete network of hosts. Other TCP/IP connections can also be tunneled through SSH to the server so that the remote user can have secure access to mail, the web, file sharing, FTP, and other services.

The SSH protocol is currently being standardized in the IETF's SECSH working group.

More information about SSH, including how to obtain commercial implementations, is available from SSH Communications Security, Ltd. (http://www.ssh.fi), Data Fellows, Ltd. (http://www.datafellows.com), and VanDyke Technologies, Inc. (http://www.vandyke.com).

Question 5.1.6. 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 2.1.2) for encryption and authentication. Version 4 could only use DES (see Question 3.2.1). Unlike a public key authentication system, Kerberos does not produce digital signatures (see Question 2.2.2). Instead 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 (the Kerberos server). If the server is compromised, the integrity of the whole system is lost. Public-key cryptography was designed precisely to avoid the necessity to trust third parties with secrets (see Question 2.2.1). Kerberos is generally considered adequate within an administrative domain; however 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].

Question 5.2.1. What are CAPIs?

A CAPI, or cryptographic application programming interface, is an interface to a library of functions 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 6.2.2 and Question 6.2.3) 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 6.2.6), 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.

Question 5.2.2. What is the GSS-API?

The Generic Security Service API (GSS-API) is a CAPI for distributed security services. It has the capacity to handle session communication securely, including authentication, data integrity, and data confidentiality. The GSS-API is designed to insulate its users from the specifics of underlying mechanisms. GSS-API implementations have been constructed atop a range of secret key and public-key technologies. The current (Version 2) GSS-API definition is available in Internet Proposed Standard RFC 2078 and GSS-API is also incorporated as an element of the Open Group Common Environment Specification. Related ongoing work items include definitions of a complementary API (GSS-IDUP) oriented to store-and-forward messaging, of a negotiation facility for selection of a common mechanism shared between peers, and of individual underlying GSS-API mechanisms.

Question 5.2.3. What are RSA BSAFE CRYPTO-C and RSA BSAFE CRYPTO-J?

RSA BSAFE Crypto-C (formerly BSAFE) and RSA BSAFE Crypto-J (formerly JSAFE) are low-level cryptographic toolkits that offer developers the tools to add privacy and authentication features to their applications.

RSA BSAFE CRYPTO-C and RSA BSAFE CRYPTO-J are designed to provide the security tools for a wide range of applications, such as digitally signed electronic forms, virus detection, or virtual private networks. RSA BSAFE CRYPTO-C can support virtually any global security standard; and RSA BSAFE CRYPTO-J is compatible with various industry standards, including S/MIME, IPSec, SSL, S/WAN and SET (see Question 5.1.1, Question 5.1.4, Question 5.1.3, Question 4.2.3, and Question 4.2.3). RSA BSAFE CRYPTO-C and RSA BSAFE CRYPTO-J fully support PKCS (see Question 5.3.3).

RSA has introduced a whole new family of elliptic curve public-key cryptographic methods to RSA BSAFE CRYPTO-C; which now includes elliptic curve cryptographic routines for encryption (analogous to the El Gamal encryption system), key agreement (analogous to Diffie-Hellman key agreement) and digital signatures (analogous to DSA, Schnorr, etc.). This implementation of elliptic curve cryptography includes all variants of ECC: Odd Prime, Even Normal, and Even Polynomial, as well as the ability to generate new curve parameters for all three fields.

RSA BSAFE CRYPTO-J is RSA's first cryptographic toolkit designed specifically for Java developers. RSA BSAFE CRYPTO-J has a full suite of Cryptographic Algorithms including RSA Public-key cryptosystem and Diffie-Hellman Key Negotiation, DES, Triple-DES, RC2, RC4, RC5, MD5and SHA-1; RSA BSAFE CRYPTO-J provides developers with a state-of-the-art implementation of the most important privacy, authentication, and data integrity routines all in Java. RSA BSAFE CRYPTO-J uses the same Java Security API developers are used to. The toolkit also includes source code for sample applications and easy-to-use self-test modules. This means proven security and shorter time-to-market for new Java project.

For more information on RSA BSAFE CRYPTO-C, RSA BSAFE CRYPTO-J and other RSA products, see http://www.rsa.com/rsa/products/

Question 5.2.4. What is SecurPC?

RSA SecurPC is a software utility that encrypts disks and files on both desktop and laptop personal computers. SecurPC extends the Windows™ File Manager or Explorer to include options for encrypting and decrypting individually selected files or files within selected folders. Each file is encrypted, using RSA's RC4 symmetric cipher, with a randomly generated, 128-bit key (40 bits for some non-U.S. users.) The random key is encrypted under the user's secret key, which is encrypted under a key derived from the user's passphrase. This allows the user's passphrase to be changed without decrypting and reencrypting all encrypted files.

SecurPC provides for optional emergency access to encrypted files, based on a *k*-of-*n* threshold scheme. The user's secret key may be stored, encrypted with the RSA algorithm, under an emergency access public key. The corresponding private key is given, in shares, to any number of trustees. A designated number of these trustees must present their shares in order to decrypt the encrypted files. For more on RSA SecurPC, see http://www.securitydynamics.com/solutions/products/securpc.html .

Question 5.2.5. What is SecurID?

SecurID is a two-factor authentication system, developed and sold by Security Dynamics. It is generally used to secure either local or remote access to computer networks. Each SecurID user has a memorized PIN or password, and a hand-held token with a LCD display. The token displays a new pseudorandom value, called the tokencode, at a fixed time interval, usually one minute. The user combines the memorized factor with the tokencode, either by simple concatenation or entry on an optional keypad on the token, to create the passcode, which is then entered to gain access to the protected resource.

The SecurID token is a battery powered, hand-held device containing a dedicated microcontroller. The microcontroller stores, in RAM, the current time, and a 64-bit seed value that is unique to a particular token. At the specified interval, the seed value and the time are combined through a proprietary algorithm stored in the microcontroller's ROM, to create the tokencode value.

An authentication server verifies the passcodes. The server maintains a database which contains the seed value for each token and the PIN or password for each user. From this information, and the current time, the server generates a set of valid passcodes for the user and checks each one against the entered value. For more on SecurID, see *http://www.securitydynamics.com*.

Question 5.2.6. 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 program that can run on multiple platforms. It provides message encryption, digital signatures, data compression, and *e*-mail compatibility.

The algorithms used for encryption are RSA (see Question 3.1.1) and Diffie-Hellman for key transport and IDEA (see Question 3.6.7), CAST, and triple-DES for bulk encryption of messages. Digital signatures are achieved by the use of RSA or DSA for signing and MD5 (see Question 3.6.6), RIPEMD-160, or SHA-1 for computing message digests. The shareware program ZIP is used to compress messages for transmission and storage. *E*-mail compatibility is achieved by the use of Radix-64 conversion.

PGP is bound by Federal export laws due to its use of the RSA public-key cryptosystem.

Question 5.3.1. What are ANSI X9 standards?

American National Standards Institute (ANSI) is broken down into committees, one being ANSI X9. The committee ANSI X9 develops standards for the financial industry, more specifically for personal identification number (PIN) management, check processing, electronic transfer of funds, etc. Within the committee of X9, there are subcommittees; further broken down are the actual documents, such as X9.9 and X9.17.

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 2.1.7) based on DES (see Question 3.2.1) in either CBC or CFB modes (see Question 2.1.4). 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 (see Question 5.3.4).

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. 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 (KEKs), 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 3.2.6) as a method by which keys can be distributed. ANSI X9.17 is gradually being supplemented by public-key techniques such as Diffie-Hellman encryption (see Question 3.6.1).

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.

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 encryption (see Question 3.6.5); ANSI X9.31 requires the MDC-2 hash algorithm [ISO92c]. A related document, X9.57, covers certificate management encryption.

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 encryption (see Question 3.6.1), authenticated Diffie-Hellman encryption, 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.

Question 5.3.2. What are the ITU-*T* (CCITT) Standards?

The International Telecommunications Union, ITU-*T* (formerly know as CCITT), is a multinational union that provides standards for telecommunication equipment and systems. ITU-*T* possesses a particular fashion for naming an ITU-*T* *X*.500 directory [CCI88b], *X*.509 certificates and Distinguished Names. Distinguished names are the standard form of naming. 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; this is something simple names, such as *e*-mail addresses, do not offer.

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) 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. The MHS message structure is similar to the MIME (see Question 5.1.1) 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.

ITU-*T* Recommendation *X*.435 [CCI91] and its equivalent *F*.435 are *X*.400-based and 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 the recipient has received the document and accepts the responsibility for it, while negative notification means the recipient refused to accept the document due to a specified reason. Forwarding notification means 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.

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 3.1.1).

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 subject'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 4.1.3.16).
The *X*.509 standard is supported by a number of protocols, including PKCS (see Question 5.3.3) and SSL (see Question 5.1.2).

Question 5.3.3. What is PKCS?

The Public-key cryptography Standards (PKCS) are 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. The PKCS have been cited by the OIW (OSI Implementers' Workshop) as a method for implementation of OSI standards. The PKCS are designed for binary and ASCII data; PKCS are also compatible with the ITU-*T X*.509 standard (see Question 5.3.2). The published standards are PKCS #1, #3, #5, #7, #8, #9, #10 #11 and #12; PCKS #13 and #14 are currently being developed.

PKCS includes both algorithm-specific and algorithm-independent implementation standards. Many algorithms are supported, including RSA (see Question 3.1.1) and Diffie-Hellman key exchange (see Question 3.6.1), however, only the latter two are specifically detailed. PKCS also defines an algorithm-independent syntax for digital signatures, digital envelopes, 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*).

The following are the Public-key cryptography Standards (PKCS):

- 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 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 syntax for certification requests.
- PKCS #11 defines a technology-independent programming interface, called Cryptoki, for cryptographic devices such as smart cards and PCMCIA cards.
- PKCS #12 specifies a portable format for storing or transporting a user's private keys, certificates, miscellaneous secrets, etc.
- PKCS #13 defines mechanisms for encrypting and signing data using Elliptic Curve Cryptography.
- PKCS #14 gives a standard for pseudo-random number generation.

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.

Question 5.3.4. What are ISO standards?

The International Organization for Standardization, (ISO), is a non-governmental body promoting standardization developments globally. Altogether, ISO is broken down into about 2700 Technical Committees, subcommittees and working groups. ISO/IEC (International Electrotechnical Commission) is the joint technical committee developing the standards for information technology.

One of the more important information technology standards developed by ISO/IEC is ISO/IEC 9798 [ISO92a]. This is an emerging international standard for entity authentication techniques. It consists of five parts. Part 1 is introductory, and Parts 2 and 3 define protocols for entity authentication using secret-key techniques and public-key techniques. Part 4 defines protocols based on cryptographic checksums, and part 5 addresses zero-knowledge techniques.

ISO/IEC 9796 is another ISO standard that defines procedures for digital signature schemes giving message recovery (such as RSA and Rabin-Williams). ISO/IEC International Standard 9594-8 is also published (and is better known) as ITU-*T* Recommendation *X*.509, "Information Technology - Open Systems Interconnection - The Directory: Authentication Framework, and is the basic document defining the most widely used form of public key certificate.

Another example of an ISO/IEC standard is the ISO/IEC 9979 [ISO91] standard defining 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.

For more information on ISO, contact their official website: *http://www.iso.ch*.

Question 5.3.5. What is IEEE P1363?

The IEEE P1363 is an emerging standard that aims to provide a comprehensive coverage of established public-key techniques. It continues to move toward completion, with balloting expected later this year. The project, begun in 1993, has produced a draft standard covering public-key techniques from the discrete logarithm, elliptic curve, and integer factorization families. Contributions are currently solicited for an addendum, IEEE P1363a, which will cover additional public-key techniques.

The project is closely coordinated with emerging ANSI standards for public-key cryptography in banking, and forthcoming revisions of RSA Laboratories' Public-key cryptography Standards will also be aligned with IEEE P1363.

For more information, see http://grouper.ieee.org/groups/1363/.

Question 5.3.6. What are some other cryptography specifications?

Although the following is not a description of actual standards, they are specifications for aspects of overall secure system design.

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 via emanation of radiation from computer systems. For instance, an attacker might intercept the radiation from the cathode ray tube (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. For more details about TEMPEST, see [RG91]

The Department of Defense (DOD) publication *Trusted Computer System Evaluation Criteria* (TCSEC), also called the Orange Book, specifies the criteria the DOD uses when 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, initially named the *Trusted Network Interpretation (TNI) of the Trusted Computer System Evaluation Criteria*, was published to provide subsidiary information to enable the Orange Book principles to be applied in a network environment. 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.

Section 6: Laws Concerning Cryptography

Section 6.1. Legal Disclaimer.

The materials should not be treated or relied upon as advice on technical and non-technical issues and the materials have not been updated to reflect recent changes in technology, the law, or any other areas. Furthermore, RSA cannot warrant that the information herein is complete or accurate and does not assume, and hereby disclaims, any liability to any person for any loss or damage caused by errors or omissions in the FAQ result from negligence, accident or any other cause.

Question 6.2.1. What is NIST?

NIST is an acronym for the National Institute of Standards and Technology, a division of the U.S. Department of Commerce. NIST 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 the development of computer-based economic activity. NIST issues standards and guidelines intended to be adopted in 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.

NIST issues standards for cryptographic algorithms that U.S. government agencies are required to use.
A large percentage of the private sector often adopts them as well. In January 1977, NIST declared DES (see Question 3.2.1) the official U.S. encryption standard and published it as FIPS Publication 46; DES soon became a de facto standard throughout the United States. NIST is currently taking nominations for the Advanced Encryption Standard (AES), which is to replace DES, (see Questions 3.3.1 - 3.3.3 for more details on AES). There is no definite deadline for the completion of the AES; however, submissions were only accepted until June 15, 1998. (see Question 3.3.3)

Several 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 6.2.3). After a few years of rather secretive deliberations, NIST, in cooperation with the NSA (see Question 6.2.2.), issued proposals for various standards in cryptography. The combination of these proposals, including digital signatures (DSS, see Question 3.4.1) and data encryption (the Clipper chip, see Question 6.2.4), formed the Capstone project.

NIST has been criticized for allowing the NSA too much power in setting cryptographic standards, since the interests of the NSA sometimes 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 so it is perhaps unrealistic to expect NIST to forego such readily available assistance.

For more information on NIST, visit their website at: http://www.nist.gov/

Question 6.2.2. What is the NSA?

NSA is the National Security Agency, a highly secretive agency of the U.S. government created by Harry Truman in 1952. The NSA's 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 to slow the spread of publicly available cryptography in order to prevent national enemies from employing encryption methods that are presumably 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 3.2.1), 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. Also 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.

The NSA's charter limits its activities to foreign intelligence. However, the NSA is concerned with the development of commercial cryptography, since 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 that strong commercial cryptography may 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 export of any products used for authentication purposes only, no matter how large the key size, so long as the product cannot be easily 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. 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 6.2.6), 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.

Question 6.2.3. 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 6.2.2). 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, often referred to as Clipper (see Question 6.2.4), which is the encryption chip that includes the Skipjack algorithm. The digital signature algorithm is DSA (see Question 3.4.1) and the hash function used is SHA-1 (see Question 3.6.5). The key exchange protocol is not published, but is generally considered to be related to Diffie-Hellman (see Question 3.6.1).

The Skipjack algorithm and the concept of a Law Enforcement Access Field (LEAF's, see Question 7.13) have been accepted as FIPS 185; DSS has been published as FIPS 186, and finally SHS has been published as FIPS 180.

All parts of Capstone are aimed at the 80-bit security level. The symmetric-keys involved are 80 bits long and other aspects of the algorithm suite are designed to withstand an "80-bit" attack, that is, an effort equivalent to 2^{80} operations.

Question 6.2.4. What is Clipper?

The Clipper chip contains an encryption algorithm called Skipjack (see Question 3.6.7). 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 recovered from 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 implementer'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, see Question 7.13), 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 access field are classified. The receiver decrypts the law-enforcement access 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 access 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].

Matt Blaze, AT&*T*, showed that it is possible to modify the LEAF in a way such that law enforcement cannot determine where the message originally came from [Bla94].

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. It is essential for the escrow agencies to 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. 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 key escrow agencies will be vulnerable to attack; there are not enough key escrow agencies (the current escrow agents are NIST and the automated systems division of the department of treasury [DB95]); the keys on the Clipper chips are not generated in a sufficiently secure fashion; there will not be sufficient competition among implementers, 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. He called his system 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].

Question 6.2.5. What is the Current Status of Clipper?

Clipper has been accepted as FIPS 185 [NIS94a] by the federal government. Various forms of the Clipper chip were produced; however, it is no longer in production. The chip is still used in the AT&*T* TSD 3600 and in various Fortezza products (see Question 6.2.6), including PC Cards, encrypting modems, and PCI board Fortezza. All Capstone-based products have suppressed LEAF (see Question 7.13) escrow access function. There is now a CA (Certifying Authority) performing key recovery.

Question 6.2.6. What is Fortezza?

The Fortezza Crypto Card, formerly called Tessera, is a PC card (formerly PCMCIA, Personal Computer Memory Card International Association) developed by NSA that implements the Capstone algorithms. The card provides security through verification, authentication, non-repudiation, and encryption.

Fortezza is intended for use with the Defense Messaging Service (DMS) and is export controlled. A number of vendors have announced support for the Fortezza card; NSA has also built and demonstrated a PKCS #11-based library (see Question 5.3.3) that interfaces to the card.

Currently, the NSA is working with companies, such as VLSI, to develop commercial products that implement Fortezza algorithms. VLSI is devising a "Regent" chip that adds DES and RSA algorithms. The NSA also supports commercial development of smart card chips with Fortezza algorithm capability.

Question 6.3.1. Is RSA patented?

RSA is patented under U.S. Patent 4,405,829, issued September 29, 1983 and held by RSA Data Security, Inc.; the patent expires 17 years after issue, in the year 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.

Question 6.3.2. Is DSA patented?

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. RSA Data Security has also asserted coverage of certain implementations of DSA by the Schnorr patent.

Question 6.3.3. Is DES patented?

U.S. Patent 3,962,539, which describes the Data Encryption Standard (DES), was assigned to IBM Corporation in 1976. IBM subsequently placed the patent in the public domain, offering royalty-free licenses conditional on adherence to the specifications of the standard. The patent expired in 1993.

Question 6.3.4. Are elliptic curve cryptosystems patented?

Elliptic curve cryptosystems, as introduced in 1985 by Neal Koblitz and Victor Miller, have no general patents, though some newer elliptic curve algorithms and certain efficient implementation techniques may be covered by patents.

Among the relevant implementation patents:

- Apple Computer holds a patent on efficient implementation of odd-characteristic elliptic curves, including elliptic curves over GF(p) where p is close to a power of 2
- Certicom holds a patent on efficient finite field multiplication in normal basis representation, which applies to elliptic curves with such a representation
- Cylink also holds a patent on multiplication in normal basis

Certicom also has two additional patents pending. The first of these covers the MQV (Menezes, Qu, and Vanstone) key agreement technique. Although this technique may be implemented as a discrete log system, a number of standards bodies are considering adoption of elliptic-curve-based variants. The second patent filing treats techniques for compressing elliptic curve point representations to achieve efficient storage in memory.

In all of these cases, it is the implementation technique that is patented, not the prime or representation, and there are alternative, compatible implementation techniques that are not covered by the patents. One example of such an alternative is a polynomial basis implementation with conversion to normal basis representation where needed. (This should not be taken as a guarantee that there are no other patents, of course, as this is not a legal opinion.) The issue of patents and representations is a motivation for supporting both representations in the IEEE P1363 and ANSI X9.62 standards efforts.

The patent issue for elliptic curve cryptosystems is the opposite of that for RSA and Diffie-Hellman, where the cryptosystems themselves have patents, but efficient implementation techniques often do not.

Question 6.3.5. What are the important patents in cryptography?

The following table gives a selection of some of the important and well established patents in cryptography, including several expired patents of historical interest. Note that the expiration date for these patents is 17 years after their dates of issue.

Area	Inventors	U.S. Patent #	Year of Issue	Assignee
DES	Ehrsam et al.	3,962,539	1976	IBM
Notes: This patent, which covered the DES cipher, was placed in the public domain by IBM. It is now expired.				
Diffie-Hellman	Hellman, Diffie, and Merkle	4,200,770	1980	Stanford University
Notes: This is the first patent covering a public-key cryptosystem. It describes Diffie-Hellman key agreement, as well as a means of authentication using long-term Diffie-Hellman public keys. This patent is now expired.				
Public-key cryptosystems	Hellman and Merkle	4,218,582	1980	Stanford University
Notes: The Hellman-Merkle patent covers public key systems based on the knapsack problem (and now known to be insecure). Its broader claims cover general methods of public key encryption and digital signatures using public keys. This patent is expired.				
RSA	Rivest, Shamir, Adelman	4,405,829	1983	MIT
Notes: This patent describes the RSA public-key cryptosystem as used for both encryption and signing. It served as the basis for the founding of RSADSI.				
Fiat-Shamir identification	Shamir and Fiat	4,748,668	1988	Yeda Research and Development (Israel)
Notes: Describes the Fiat-Shamir identification scheme.				
Control vectors	Matyas. Meyer, and Brachtl	4,850,017	1989	IBM
Notes: Patent 4,850,017 is the most prominent among a number describing the use of control vectors for key management. This patent describes a method enabling a description of privileges to be bound to a cryptographic key, serving as a deterrent to the key's misuse.				
GQ identification	Guillou-Quisquater	5,140,634	1992	U.S. Phillips Corporation
Notes: Describes the GQ identification scheme.				
DSA	Kravitz	5,231,668	1993	United States of America
Notes: This patent covers the Digital Signature Algorithm (DSA), the algorithm specified in the Digital Signature Standard (DSS) of the U.S. National Institute of Standards (NIST).				
IDEA	Massey-Lai	5,214,703	1993	Ascom Tech AG (Switzerland)
Notes: Patent 5,214,703 covers the IDEA block cipher, an alternative to DES that employs 128-bit keys.				
Fair cryptosystems	Micali	5,276,737*	1994	none
**with continuation in part 5,315,658. Notes: Covers systems in which keys are held in escrow among multiple trustees, only a specified quorum of which can reconstruct these keys.*				

Table 3

Question 6.4.1. 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 RSA 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 RSA for encryption. RSA for export is generally limited to 512 bits for key management purposes. The use of RSA for data encryption is generally prohibited.

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 encryption for financial transactions.

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

Question 6.4.2. Can DES be exported from the United States?

For a number of years, the government rarely approved the export of DES for use outside of the financial sector or by foreign subsidiaries of U.S. companies. Export policy has recently been liberalized to permit unrestricted exportation of DES to companies that demonstrate plans to implement key recovery systems in the next few years.

Question 6.4.3. Why is cryptography export-controlled?

Cryptography is export-controlled for several reasons. Strong cryptography can be used for criminal purposes or even as a weapon of war. During wartime, the ability to intercept and decipher enemy communications is crucial. For that reason, strong cryptography is usually classified on the U.S. Munitions List as an export-controlled commodity, just like tanks and missiles. Cryptography is just one of many technologies which is covered by the ITAR (International Traffic in Arms Regulations).

In the United States, government agencies consider *strong encryption* to be systems that use RSA with key sizes over 512-bits or symmetric algorithms (like DES, IDEA, or RC5) with key sizes over 40-bits. Since government encryption policy is heavily influenced by the agencies responsible for gathering domestic and international intelligence (the FBI and NSA, respectively) the government is compelled to balance the conflicting requirements of making strong cryptography available for commercial purposes while still making it possible for those agencies to break those codes, if need be. The US government does, however, allow 56-bit block ciphers to be exported for financial cryptography.

To most cryptographers, this level of cryptography is not considered “strong” at all. In fact, it is worth noting that RSA Laboratories has considered this level of cryptography to be commercially inadequate for several years, and currently recommends that domestic customers utilize at least 80-bit secret key or 768-bit public-key cryptography.

Government agencies often prefer to use the terms “strategic” and “standard” to differentiate encryption systems. “Standard” refers to algorithms that have been drafted and selected as a federal standard - DES is the primary example. The government defines “strategic” as any algorithm which requires “excessive work factors” to successfully attack. Unfortunately, the government rarely publishes criteria for what it defines as “acceptable” or “excessive” work factors.

Question 6.4.4. Are digital signature applications exportable from the United States?

Digital signature applications are one of the nine special categories of cryptography that automatically fall under the more relaxed Commerce regulations. Digital signature implementations using RSA key sizes in excess of 512 bits are exportable. However, there are some restrictions when developing a digital signature application using a reversible algorithm (i.*e.*, the signing operation is sort of the reverse operation for encryption), such as RSA. In this case, the application should sign a hash of the message, not the message itself. Otherwise, the message must be transmitted with the signature appended. If the message is not transmitted with the signature, the NSA considers this quasi-encryption and the State controls would apply.

In practice, however, prior to shipping overseas, a commodity classification request should be submitted to the Department of Commerce. This classification will indicate the specific export licensing requirements for the product. Experience has shown Commerce personnel often request a commodity jurisdiction (CJ) request be obtained prior to issuance of the classification notice. In this event, contact the NSA Office of Export Control to coordinate the CJ submission. It is not necessary to register with the Department of State prior to the submission of commodity jurisdiction requests or to export products controlled by Commerce.
Question 6.5.1. Which major countries have import restrictions on cryptography?

France, Israel, Russia and South Africa all have import restrictions on cryptography. Some countries such as France and Singapore require vendors to obtain a license before importing cryptographic products. Many governments use such import licenses to pursue domestic policy goals. In some instances, governments require foreign vendors to provide technical information to obtain an import license. This information is then used to steer business toward local companies. Other governments have been accused of using this same information for outright industrial espionage.

Should you desire to do business in a country with such import restrictions, and you can accept the requirements for technical disclosure that the government may impose, you should consult with export agencies or legal firms with multi-national experience in order to comply with all applicable regulations.

Question 6.5.1. Which major countries have import restrictions on cryptography?

France, Israel, Russia and South Africa all have import restrictions on cryptography. Some countries such as France and Singapore require vendors to obtain a license before importing cryptographic products. Many governments use such import licenses to pursue domestic policy goals. In some instances, governments require foreign vendors to provide technical information to obtain an import license. This information is then used to steer business toward local companies. Other governments have been accused of using this same information for outright industrial espionage.

Should you desire to do business in a country with such import restrictions, and you can accept the requirements for technical disclosure that the government may impose, you should consult with export agencies or legal firms with multi-national experience in order to comply with all applicable regulations.

Question 7.1. What is probabilistic encryption?

Probabilistic encryption, developed 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). This is done 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, it was not 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].

Question 7.2. What are special signature schemes?

Since the time Diffie and Hellman introduced the concept of digital signatures (see Question 2.2.2), 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-repudable) 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 other security needs that might be desired in different applications. Questions 7.3 through 7.8 deal with some of these special signature schemes.

Question 7.3. 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 (see Question 3.1.1) 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 remains 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 7.11), anonymous access control, and digital cash (see Question 4.2.1). 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].

Question 7.4. What is a designated confirmer signature?

A designated confirmer signature [Cha94] strikes a balance between self-authenticating digital signatures (see Question 7.2) and zero-knowledge proofs (see Question 2.1.8). 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 3.1.1).

Question 7.5. 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 7.7), 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.)

Question 7.6. 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 that 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.

Question 7.7. 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 3.1.1), 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 3.6.9), which does not require new key pairs for each message.

Question 7.8. 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 7.2), 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 2.3.7). 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 they have signed.

Question 7.9. What are on-line/off-line signatures?

On-line/off-line signature schemes are a way of getting around the fact that many general-purpose digital signature schemes have high computational requirements. On-line/off-line schemes are created by joining together a general-purpose signature scheme (see Question 2.2.2) and a one-time signature scheme (see Question 7.7) in such a way that the bulk of the computational burden for a signature operation can be performed before the signer knows the message that will be signed.

More precisely, let a general-purpose digital signature scheme and a one-time signature scheme be fixed. These schemes can be used together to define an on-line/off-line signature scheme which works as follows:

1. *Keypair generation*. A public/private keypair K_P/K_S for the general-purpose signature scheme is generated. These are the public and private keys for the on-line/off-line scheme as well.
2. *Off-line phase of signing*. A public/private keypair T_P/T_S for the one-time signature scheme is generated. The public key T_P for the one-time scheme is signed with the private key K_S for the general-purpose scheme to produce a signature $S_K(T_P)$.
3. *On-line phase of signing*. To sign a message m, use the one-time scheme to sign m with the private key T_S, computing the value $S_T(m)$. The signature of m is then the triple $(T_P, S_K(T_P), S_T(m))$.

Note that steps 2 and 3 must be performed for each message signed; however, the point of using an on-line/off-line scheme is that step 2 can be performed before the message m has been chosen and made available to the signer. An on-line/off-line signature scheme can use a one-time signature scheme that is much faster than a general-purpose signature scheme, and this can make digital signatures much more practical in a variety of scenarios. An on-line/off-line signature scheme can be viewed as the digital signature analog of a digital envelope (see Question 2.2.4).

For more information about on-line/off-line signatures, see [EGM89].

Question 7.10. What is OAEP?

Optimal Asymmetric Encryption Padding (OAEP) is a method for encoding messages developed by Mihir Bellare and Phil Rogaway [BR94]. The technique of encoding a message with OAEP and then encrypting it with RSA is provably secure in the random oracle model. Informally, this means that if hash functions are truly random, then an adversary who can recover such a message must be able to break RSA.

An OAEP encoded message consists of a “masked data” string concatenated with a “masked random number”. In the simplest form of OAEP, the masked data is formed by taking the exclusive-or of the plaintext message, M, and the hash, G, of a random string r. The masked random number is the exclusive-or of r with the hash, H, of the masked data. The input to the RSA encryption function is then

$$[M \oplus G(r)] \mid\mid [r \oplus H(M \oplus G(r))].$$

Often, OAEP is used to encode small items such as keys. There are other variations on OAEP (differing only slightly from the above) that include a feature called “plaintext-awareness”. This means that to construct a valid OAEP encoded message, an adversary must know the original plaintext. To accomplish this, the plaintext message M is first padded (*e.g.*, with a string of zeroes) before the masked data is formed. OAEP is supported in the ANSI X9.44, IEEE P1363 and SET standards.

Question 7.11. What is digital timestamping?

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 about exactly this record-exactly this sequence of bits. 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 2.2.2). A system for answering the second question is called a digital timestamping scheme. Such systems are described in [BHS93] and [HS91], and an implementation is commercially available from Surety Technologies (*http://www.surety.com/*).

Any system allowing users to answer these questions must include two procedures. First, there must be a signing 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 string of bytes that serves as the signature. Second, there must be a verification procedure by which any user can check a record and its purported signature to make sure it correctly answers (1) who and what? or (2) when and what? about the record in question.

The signing procedure of a digital timestamping system 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.

One nice thing about digital timestamps is that the document being timestamped does not have to be released to anybody to create a timestamp. The originator of the document computes the hash values himself, and sends them in to the timestamping service. The document itself is only needed for verifying the timestamp. This is very useful for many reasons (like protecting something that you might want to patent).

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 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-reputability 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 hash functions (see Question 2.1.6). (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.

Question 7.12. What is key recovery?

One of the barriers to the widespread use of encryption in certain contexts is the fact that when a key is somehow “lost”, any data encrypted with that key becomes unusable. Key recovery is a general term encompassing the numerous ways of permitting “emergency access” to encrypted data.

One common way to perform key recovery, called key escrow, is to split a decryption key (typically a secret key or an RSA private key) into several parts and distribute these parts to escrow agents or “trustees”. In an emergency situation (exactly what defines an “emergency situation” is context-dependent), these trustees can use their “shares” of the keys either to reconstruct the missing key or simply to decrypt encrypted communications directly. This method is used by Security Dynamics’ RSA SecurPC product.

Another recovery method, called key encapsulation, is to encrypt data in a communication with a “session key” (which varies from communication to communication) and to encrypt that session key with a trustee’s public key. The encrypted session key is sent with the encrypted communication, and so the trustee is able to decrypt the communication when necessary. A variant of this method, in which the session key is split into several pieces, each encrypted with a different trustee’s public key, is used by TIS’ RecoverKey.

Dorothy Denning and Dennis Branstad have written a survey of key recovery methods [DB96].

Key recovery first gained notoriety as a potential work-around to the United States Government’s policies on exporting “strong” cryptography. To make a long story short, the Government agreed to permit the export of systems employing strong cryptography as long as a key recovery method that permits the Government to read encrypted communications (under appropriate circumstances) was incorporated. For the Government’s purposes, then, “emergency access” can be viewed as a way of ensuring that the Government has access to the plaintext of communications it is interested in, rather than as a way of ensuring that communications can be decrypted even if the required key is lost.

Key recovery can also be performed on keys other than decryption keys. For example, a user’s private signing key might be recovered. From a security point of view, however, the rationale for recovering a signing key is generally less compelling than that for recovering a decryption key.

Question 7.13. What are LEAFs?

A LEAF, or Law Enforcement Access Field, is a small piece of "extra" cryptographic information that is sent or stored with an encrypted communication to ensure that appropriate Government entities, or other authorized parties, can obtain the plaintext of some communication. For a typical escrowed communication system, a LEAF might be constructed by taking the decryption key for the communication, splitting it into several shares, encrypting each share with a different key escrow agent's public key, and concatenating the encrypted shares together.

The term "LEAF" originated with the Clipper Chip (see Question 6.2.4, "What is Clipper?," for more information).

Question 7.14. What is PSS/PSS-*R*?

PSS (Probabilistic Signature Scheme) is a provably secure way of creating signatures with RSA (see Question 3.1.8) due to Mihir Bellare and Phillip Rogaway [BR96]. Informally, a digital signature scheme is provably secure if its security can be tied closely to that of an underlying cryptographic primitive. The proof of security for PSS takes place in the *random oracle model*, in which hash functions are modeled as being truly random functions. Although this model is not realistically attainable, there is evidence that practical instantiations of provably secure schemes are still better than schemes without provable security [BR93]. The method for creating digital signatures with RSA that is described in PKCS #1 (see Question 5.3.3) has not been proven secure even if the underlying RSA primitive is secure; in contrast, PSS uses hashing in a sophisticated way to tie the security of the signature scheme to the security of RSA.

A PSS signature for a message M is formed by the following steps:

1. A random string r is concatenated with M to form $M \mid\mid r$.
2. A hash function h is applied to $M \mid\mid r$, forming $h(M \mid\mid r)$.
3. A hash function g is applied to $h(M \mid\mid r)$, forming $g(h(M \mid\mid r))$.
4. $g(h(M \mid\mid r))$ is divided into two parts, $g(h(M \mid\mid r))=[\mathrm{g1}(h(M \mid\mid r)) \mid\mid \mathrm{g2}(h(M \mid\mid r))]$.
5. The exclusive-OR of $\mathrm{g1}(h(M \mid\mid r))$ and r is taken, forming $g_1(h(M \mid\mid r)) \oplus r$
6. The following string is formed: $w = [0 \mid\mid h(M \mid\mid r) \mid\mid g_1(h(M \mid\mid r)) \oplus r \mid\mid g_2(h(M \mid\mid r))]$
7. The inverse RSA function is applied to w, forming f-1(w).
8. The output is the following message-signature pair, $(M, f^{-1}(w))$.

A verifier uses the sender's public key to recover w from $f^{-1}(w)$. The verifier then computes $h(M \mid\mid r)$ from w and applies g_1 to it. The resulting string can be exclusive-ORed with the relevant part of w to recover r. Now that the user has both M and r he can compute the string in step 6 directly, and compare it with w.

To minimize the length of communications, it is often desirable to have signature schemes in which the message can be "folded" into the signature. Schemes that accomplish this are called *message recovery* signature schemes. PSS-*R* is a message recovery variant of PSS. To create a PSS-*R* signature, form w as above and then take the exclusive-or of M and $\mathrm{g2}(h(M \mid\mid r))$ and place this new string where $g_2(h(M \mid\mid r))$ was in w. To verify, all one needs is the signature. The verifier can recover the message from the signature by applying g_2 to $h(M \mid\mid r)$ and then taking the exclusive-OR of that with the exclusive-OR of M and $g_2(h(M \mid\mid r))$. The user then proceeds with the verification as in PSS. PSS-R has the same provable security as PSS.

Question 7.15. What are covert channels?

Covert communication channels (also called subliminal channels) are often motivated as being solutions to the "prisoner's problem." Consider two prisoners in separate cells who want to exchange messages, but must do so through the warden, who demands full view of the messages (i.*e*. no encryption). A covert channel enables the prisoners to exchange secret information through messages that appear to be innocuous. A covert channel requires prior agreement on the part of the prisoners. For example if an odd length word corresponds to "1" and an even length word corresponds to "0", then the previous sentence contains the subliminal message "101011010011".

An important use of covert channels is in digital signatures. If such signatures are used, a prisoner can both authenticate the message and extract the subliminal message. Gustavus Simmons [Sim93a] devised a way to embed a subliminal channel in DSA (see Question 3.4.1) that uses all of the available bits (i.*e*. those not being used for the security of the signature), but requires the recipient to have the sender's secret key. Such a scheme is called *broadband* and has the drawback that the recipient is able to forge the sender's signature. Simmons [Sim93b] also devised schemes that use fewer of the available bits for a subliminal channel (called *narrowband* schemes) but don't require the recipient to have the sender's secret key.

Question 7.16. What are proactive security techniques?

Proactive security combines the ideas of distributed cryptography (also called secret sharing) (see Question 2.1.9) with the refreshment of secrets. The term *proactive* refers to the fact that it's not necessary for a breach of security to occur before secrets are refreshed, the refreshment is done periodically (and hence, proactively). Key refreshment is an important addition to distributed cryptography because without it, an adversary who is able to recover all the distributed secrets given enough time will eventually be successful in breaking the system. For example, consider the following proactive version of Shamir's secret sharing scheme (see Question 3.6.12):

1. User i has a point $(x_i, y_i)^t$ (x_i is nonzero) on the (m-1)-degree polynomial over GF(q), $F^t(x)=a_0+a_1x+...+a_{m-1}x^{m-1}$. t indicates the refreshment period (t=0,1,...).
2. The secret is $F^t(0)$.
3. The polynomial F^t is updated at time t by adding to it a random (m-1)-degree polynomial G, such that $G(0)=0$ ($F^{t+1}= F^t +G$). Therefore, the secret is unchanged ($F^{t+1}(0)= F^t(0)+G(0)$), but user i's new secret share is $(x_i, y_1)^{t+1} = (x_i, F^t(x_i)+G(x_i))$.

An adversary who knows less than m current secret shares during any particular time period, knows nothing about the secret.

More recent techniques in proactive security include proactive RSA [FGM97] and proactive signatures (see [GJK96] and [HJJ97]). For an overview of proactive techniques see [CGH97].

Question 7.17. 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, a bit has a discrete range and can represent either a zero state or a one state. A qubit can be in a linear superposition of the two states. . Hence, when a qubit is measured the result will be zero with a certain probability and one with the complementary probability. A quantum register consists of n qubits. Because of superposition, a phenomenon known as quantum parallelism allows exponentially many computations to take place simultaneously, thus vastly increasing the speed of computation.

Quantum interference, the analog of Young's double-slit experiment that 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 2.3.3) and compute discrete logarithms (see Question 2.3.7) 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 7.18) is quite different from, and currently more viable than, quantum computing.

Question 7.18. 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 13.

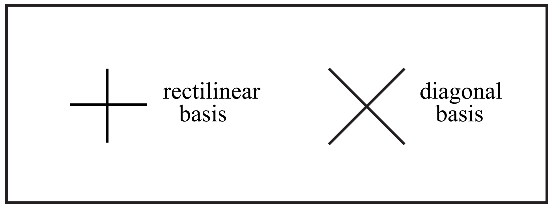


Figure 13. Bases

If a photon's polarization is read in the same basis twice, 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 14.

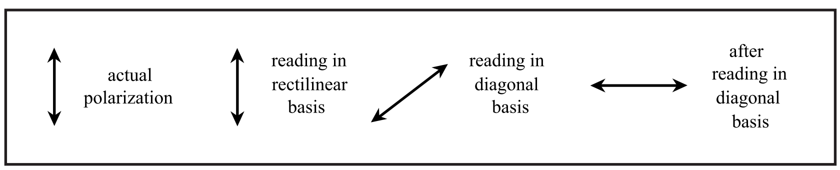


Figure 14. Polarization readings

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 not the same as quantum cryptography), see Question 7.17.

Question 7.19. What is DNA computing?

DNA computing, also known as molecular computing, is a new approach to massively parallel computation based on groundbreaking 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 estimated that to factor a 1000-bit number following Adleman's original approach, the required amount of solution would be 10^{200000} liters! 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 7.17), on the other hand, factoring can theoretically be done in (quantum) polynomial time, so quantum computing might be viewed with more concern than DNA computing.

Question 7.20. What are biometric techniques?

The term biometrics applies to a broad range of electronic techniques that employ the physical characteristics of human beings as a means of authentication. In a sense, human beings already routinely authenticate one another biometrically: confirming the identity of a friend on the telephone by the sound of his or her voice is a simple instance of this. A number of biometric techniques have been proposed for use with computer systems. These include (among a wide variety of others) fingerprint readers, iris scanners, face imaging devices, hand geometry readers, and voice readers. Usage of biometric authentication techniques is often recommended in conjunction with other user authentication methods, rather than as a single, exclusive method.

Fingerprint readers are likely to become a common form of biometric authentication device in the coming years. To identify herself to a server using a fingerprint reader, a user places her finger on a small reading device. This device measures various characteristics of the patterns associated with the fingerprint of the user, and typically transmits these measurements to a server. The server compares the measurements taken by the reader against a registered set of measurements for the user. The server authenticates the user only if the two sets of measurements correspond closely to one another. One significant characteristic of this and other biometric technologies is that matching must generally be determined on an approximate basis, with parameters tuned appropriately to make the occurrence of false positive matches or false negative rejections acceptably infrequent.

Question 7.21. What is tamper-resistant hardware?

One part of designing a secure computer system is ensuring that various cryptographic keys can be accessed only by their intended user(s) and only for their intended purposes. Keys stored inside a computer can be vulnerable to use, abuse, and/or modification by an unauthorized attacker.

For a variety of situations, an appropriate way to protect keys is to store them in a tamper-resistant hardware device. These devices can be used for applications ranging from secure *e*-mail to electronic cash and credit cards. They offer physical protection to the keys residing inside them, thereby providing some assurance that these keys have not been maliciously read or modified. Typically, gaining access to the contents of a tamper-resistant device requires knowledge of a PIN or password; exactly what type of access can be gained with this knowledge is device-dependent.

Some tamper-resistant devices do not permit certain keys to be exported outside the hardware. This can provide a very strong guarantee that these keys cannot be abused: the only way to use these keys is to physically possess the particular device. Of course, these devices must actually be able to perform cryptographic functions with their protected keys, since these keys would otherwise be useless.

Tamper-proof devices come in a variety of forms and capabilities. One common type of device is a “smartcard”, which is approximately the size and shape of a credit card. To use a smartcard, one inserts it into a smartcard reader that is attached to a computer. Smartcards are frequently used to hold a user’s private keys for financial applications; Mondex (see Question 4.2.4) is a system that makes use of tamper-resistant hardware in this fashion.

Question 7.22. How are hardware devices made tamper-resistant?

There are many techniques that are used to make hardware tamper-resistant (see Question 7.21). Some of these techniques are intended to thwart direct attempts at opening a device and reading information out of its memory; others offer protection against subtler attacks, such as timing attacks and induced hardware-fault attacks.

At a very high level, a few of the general techniques currently in use to make devices tamper-resistant are:

- Employing sensors of various types (for example, light, temperature, and resistivity sensors) in attempt to detect occurrences of malicious probing.
- Packing device circuitry as densely as possible (dense circuitry makes it difficult for attackers to use a logic probe effectively).
- Using error-correcting memory.
- Making use of non-volatile memory so that the device can tell if it has been reset (or how many times it has been reset).
- Using redundant processors to perform calculations, and ensuring that all the calculated answers agree before outputting a result.

Question 8.1. Where can I learn more about cryptography?

There are a number of textbooks available to the student of cryptography. Among the most useful are the following three.

Applied Cryptography, by B. Schneier, John Wiley & Sons, Inc., 1996.
Schneier's book is an accessible and practically oriented book with very broad coverage of recent and established cryptographic techniques.

Handbook of Applied Cryptography, by A.J. Menezes, *P*.C. van Oorschot, S.A. Vanstone. CRC Press, 1996.
The *HAC* offers a thorough treatment of cryptographic theory and protocols, with a great deal of detailed technical information. It is an excellent reference book, but somewhat technical, and not aimed to serve as an introduction to cryptography.

Cryptography: Theory and Practice, by *D. R.* Stinson. CRC Press, 1995.
This is a textbook, and includes exercises. Theory comes before practice in both title and content, but the book provides a good introduction to the fundamentals of cryptography.

For additional information, or more detailed information about specific topics, the reader is referred to the chapter summaries and bibliographies in any one of these texts.

Question 8.2. Where can I learn more about cryptographic protocols and architecture?

The best way to learn more about cryptographic protocols and architecture is to read the proceedings from the various conferences on cryptography. The International Association for Cryptographic Research (IACR) is an excellent source for this: www.iacr.org.

Also, RSA Laboratories regularly publishes a newsletter on recent cryptography happenings called CryptoBytes. It can be found at www.rsa.com/rsalabs/pubs/cryptobytes/.

Question 8.3. Where can I learn more about recent advances in cryptography?

There are many annual conferences devoted to cryptographic research. The proceedings from these conferences are excellent sources for information about recent advances. The IACR (www.iacr.org) runs many of the more prominent conferences, and their web site contains information on the proceedings. Some of the major cryptographic research conferences are:

ACM Conference
AsiaCrypt
Crypto
EuroCrypt
IEEE Symposium on Security and Privacy, and
ISOC Symposium

Question 8.4. Where can I learn more about electronic commerce?

As electronic commerce is a very rapidly changing field, the best resources are perhaps those available on the World Wide Web. The following is a selection of survey sites available as of the beginning of 1998.

Payment mechanisms designed for the Internet:
http://ganges.cs.tcd.ie/mepeirce/Project/oninternet.html

iWORLD's guide to electronic commerce:
http://*e*-comm.iworld.com/

Electronic Commerce, Payment Systems, and Security:
http://www.semper.org/sirene/outsideworld/ecommerce.html#syst

Electronic Payment Schemes:
http://www.w3.org/pub/WWW/Payments/roadmap.html

Question 8.5. Where can I learn more about cryptography standards?

Several organizations are involved in defining standards related to aspects of cryptography and its application.

ANSI:
The American National Standards Institute (ANSI) has a broadly based standards program, and some of the groups within its Financial Services area (Committee X9) establish standards related to cryptographic algorithms. Examples include X9.17 (key management: wholesale), X9.19 (message authentication: retail), and X9.30 (public-key cryptography). Information can be found at http://www.x9.org.

IEEE:
The Institute of Electrical and Electronic Engineers (IEEE) has a broadly based standards program, including P1363 (Public-key cryptography). Information can be found at http://www.ieee.org.

IETF:
The Internet Engineering Task Force (IETF) is the defining body for Internet protocol standards. Its security area working groups specify means for incorporating security into the Internet's layered protocols. Examples include IP layer security (IPSec), transport layer security (TLS), Domain Name System security (DNSsec) and Generic Security Service API (GSS-API). Information can be found at http://www.ietf.org.

ISO and ITU:
The International Standards Organization's International Electrotechnical Commission (ISO/IEC) and the International Telecommunications Union's Telecommunication Standardization Sector (ITU-*T*) have broadly-based standards programs (many of which are collaborative between the organizations), which include cryptographically-related activities. Example results are: ITU-*T* Recommendation *X*.509, which defines facilities for public key certification, and the ISO/IEC 9798 document series, which defines means for entity authentication. ITU information can be found at http://www.itu.ch, and ISO information at http://www.iso.ch.

NIST:
The U.S. National Institute of Standards and Technology (NIST)'s Information Technology Laboratory produces a series of information processing specifications (Federal Information Processing Standards (FIPS)), several of which are related to cryptographic algorithms and usage. Examples include FIPS PUB 46-2 (Data Encryption Standard (DES)) and FIPS PUB 186 (Digital Signature Standard (DSS)). Information is available at http://www.nist.gov.

Open Group:
The Open Group produces a range of standards, some of which are related to cryptographic interfaces (APIs) and infrastructure components. Examples include Common Data Security Architecture (CDSA) and Generic Crypto Service API (GCS-API). Information can be found at http://www.opengroup.org.

PKCS:
RSA Laboratories is responsible for the development of the Public-key cryptography Standards (PKCS) series of specifications, which define common cryptographic data elements and structures. Information can be found at http://www.rsa.com/rsalabs/pubs/.

Question 8.6. Where can I learn more about laws concerning cryptography?

The best way to learn more about any specific question you might have about laws concerning cryptography is to consult with an attorney. Beyond that, www.epic.com and www.c2.net are web pages of organizations devoted to following laws and legislation concerning cryptography. Also, any legal archive is a good source for information about laws concerning cryptography.

Glossary

adaptive-chosen-ciphertext - A version of the chosen-ciphertext attack where the cryptanalyst can choose ciphertexts dynamically. A cryptanalyst can mount an attack of this type in a scenario in which he or she has free use of a piece of decryption hardware, but is unable to extract the decryption key from it.

adaptive-chosen-plaintext - A special case of the chosen-plaintext attack in which the cryptanalyst is able to choose plaintexts dynamically, and alter his or her choices based on the results of previous encryptions.

adversary - Commonly used to refer to the opponent, the enemy, or any other mischievous person that desires to compromise one's security.

AES - The Advanced Encryption Standard that will replace DES (The Data Encryption Standard) around the turn of the century.

algebraic attack - A method of cryptanalytic attack used against block ciphers that exhibit a significant amount of mathematical structure.

algorithm - A series of steps used to complete a task.

Alice -The name traditionally used for the first user of cryptography in a system; Bob's friend.

ANSI - American National Standards Institute.

API - Application Programming Interface.

attack - Either a successful or unsuccessful attempt at breaking part or all of a cryptosystem. See algebraic attack, birthday attack, brute force attack, chosen ciphertext attack, chosen plaintext attack, differential cryptanalysis, known plaintext attack, linear cryptanalysis, middleperson attack.

authentication - The action of verifying information such as identity, ownership or authorization.

big-*O* notation - Used in complexity theory to quantify the long-term time dependence an algorithm with respect to the size of the input.

biometrics - The science of using biological properties to identify individuals; for example, finger prints, a retina scan, and voice recognition.

birthday attack - A brute-force attack used to find collisions. It gets its name from the surprising result that the probability of two or more people in a group of 23 sharing the same birthday is greater than 1/2.

bit - A binary digit, either 1 or 0.

blind signature scheme - Allows one party to have a second party sign a message without revealing any (or very little) information about the message to the second party.

block - A sequence of bits of fixed length; longer sequences of bits can be broken down into blocks.

block cipher - A symmetric cipher which encrypts a message by breaking it down into blocks and encrypting each block.

block cipher based MAC - MAC that is performed by using a block cipher as a keyed compression function.

Bob - The name traditionally used for the second user of cryptography in a system; Alice's friend.

boolean expression - A mathematical expression in which all variables involved are either 0 or 1; it evaluates to either 0 or 1.

brute force attack - This attack requires trying all (or a large fraction of all) possible values till the right value is found; also called an exhaustive search.

CA - See certifying authority

CAPI - Cryptographic Application Programming Interface.

Capstone - The U.S. government's project to develop a set of standards for publicly available cryptography, as authorized by the Computer Security Act of 1987. See Clipper, DSA, DSS, and Skipjack.

certificate - In cryptography, an electronic document binding some pieces of information together, such as a user's identity and public key. Certifying Authorities (CA's) provide certificates.

certificate revocation list - A list of certificates that have been revoked before their expiration date.

Certifying Authority (CA) - A person or organization that creates certificates.

checksum - Used in error detection, a checksum is a computation done on the message and transmitted with the message; similar to using parity bits.

chosen ciphertext attack - An attack where the cryptanalyst may choose the ciphertext to be decrypted.

chosen plaintext attack - A form of cryptanalysis where the cryptanalyst may choose the plaintext to be encrypted.

cipher - An encryption - decryption algorithm.

ciphertext - Encrypted data.

ciphertext-only attack - A form of cryptanalysis where the cryptanalyst has some ciphertext but nothing else.

Clipper - Clipper is an encryption chip developed and sponsored by the U.S. government as part of the Capstone project.

collision - Two values x and y form a *collision* of a (supposedly) one-way function F if $x \neq y$ but $F(x) = F(y)$.

collision free - A hash function is *collision free* if collisions are hard to find. The function is *weakly collision free* if it is computationally hard to find a collision for a given message x. That is, it is computationally infeasible to find a message $y \neq x$ such that $H(x) = H(y)$. A hash function is *strongly collision free* if it is computationally infeasible to find *any* messages x, y such that $x \neq y$ and $H(x) = H(y)$.

collision search - The search for a collision of a one-way function.

commutative - When a mathematical operator yields the same result regardless of the order the objects are operated on. For example if a, b are integers then $a+b = b+a$, that is, the addition operator acting on integers is commutative.

commutative group - A group where the operator is commutative, also called an Abelian group.

computational complexity - Refers to the amount of space (memory) and time required to solve a problem.

NP - Nondeterministic Polynomial. Refers to the running time of the best known algorithm for solving a particular problem. A set of problems where the best-known algorithm solving it would run in polynomial time on a nondeterministic computer; such problems are said to be "*NP*" or "in *NP*".

***NP*-complete** - A problem is *NP*-complete if any *NP* problem can be reduced (transformed) to it, and it is itself *NP*.

P - Polynomial. Refers to the running time of the best known algorithm solving a particular problem. A set of problems where the best known algorithm solving it runs in polynomial time; such a problem is said to be "*P*" or "in *P*".

space - Referring to spatial (memory) constraints involved in a certain computation.

time - Referring to the temporal constraints involved in a certain computation.

compression function - A function that takes a fixed length input and returns a shorter, fixed length output. See also hash functions.

compromise - The unintended disclosure or discovery of a cryptographic key or secret.

concatenate - To place two (or more) things together one directly after the other. For example, treehouse is the concatenation of the words *tree* and *house*.

covert channel - A hidden communication medium. See also subliminal channel.

CRL - Certificate Revocation List.

cryptanalysis - The art and science of breaking encryption or any form of cryptography. See attack.

cryptography - The art and science of using mathematics to secure information and create a high degree of trust in the electronic realm. See also public key, secret key, symmetric-key, and threshold cryptography.

cryptology - The branch of mathematics concerned with cryptography and cryptanalysis.

cryptosystem - An encryption - decryption algorithm (cipher), together with all possible plaintexts, ciphertexts and keys.

Data Encryption Standard - See DES.

decryption - The inverse (reverse) of encryption.

DES - Data Encryption Standard, a block cipher developed by IBM and the U.S. government in the 1970's as an official standard. See also block cipher.

dictionary attack - A brute force attack that tries passwords and or keys from a precompiled list of values. This is often done as a precomputation attack.

Diffie-Hellman key exchange - A key exchange protocol allowing the participants to agree on a key over an insecure channel.

differential cryptanalysis - A chosen plaintext attack relying on the analysis of the evolution of the differences between two plaintexts.

digest - Commonly used to refer to the output of a hash function, *e.g*. message digest refers to the hash of a message.

digital cash - See electronic money

digital envelope - A key exchange protocol that uses a public-key cryptosystem to encrypt a secret key for a secret-key cryptosystem.

digital fingerprint - See digital signature.

digital signature - The encryption of a message digest with a private key.

digital timestamp - A record mathematically linking a document to a time and date.

discrete logarithm - Given two elements d, g, in a group such that there is an integer r satisfying $g^r = d$, r is called the discrete logarithm.

discrete logarithm problem - The problem of given d and g in a group, to find r such that $g^r = d$. For some groups, the discrete log problem is a hard problem that can be used in public-key cryptography.

distributed key - A key that is split up into many parts and shared (distributed) among different participants. See also secret sharing.

DMS - Defense Messaging Service.

DOD - Department of Defense.

DSA - Digital Signature Algorithm. DSA is a public-key method based on the discrete log problem.

DSS - Digital Signature Standard. DSA is the Digital Signature Standard.

ECC - Elliptic Curve Cryptosystem; A public-key cryptosystem based on the properties of elliptic curves.

ECDL - See elliptic curve discrete logarithm.

EDI - Electronic (business) Data Interchange.

electronic commerce (*e*-commerce) - Business transactions conducted over the Internet.

electronic mail (*e*-mail) - Messages sent electronically from one person to another via the Internet.

electronic money - Electronic mathematical representation of money.

elliptic curve - The set of points (x, y) satisfying an equation of the form $y^2 = x^3 + ax + b$, for variables x, y and constants a, b.

elliptic curve cryptosystem - See ECC.

elliptic curve discrete logarithm (ECDL) problem - The Problem of given two points P and Q on an elliptic curve, to find m satisfying $mP = Q$, assuming such an m exists.

elliptic curve (factoring) method - A special-purpose factoring algorithm that attempts to find a prime factor p of an integer n by finding an elliptic curve whose number of points modulo p is divisible by only small primes.

encryption - The transformation of plaintext into an apparently less readable form (called ciphertext) through a mathematical process. The ciphertext may be read by anyone who has the key that decrypts (undoes the encryption) the ciphertext.

exclusive or - See XOR.

exhaustive search - Checking every possibility individually till the right value is found. See also attack.

expiration date - Certificates and keys may have a limited lifetime; expiration dates are used to monitor this.

exponential function - A function where the variable is in the exponent of some base, for example, b^N where N is the variable, and b is some constant.

exponential running time - If the running time, given as a function of the length of the input, is an exponential function, the algorithm is said to have exponential running time.

export encryption - Encryption, in any form, which leaves its country of origin. For example, encrypted information or a computer disk holding encryption algorithms that is sent out of the country.

factor - Given an integer N, any number that divides it is called a factor.

factoring - The breaking down of an integer into its prime factors. This is a hard problem.

factoring methods - See elliptic curve method, multiple polynomial quadratic sieve, number field sieve, Pollard p-1 and Pollard p+1 method, Pollard rho method, quadratic sieve.

FBI - Federal Bureau of Investigation, a U.S. government law enforcement agency.

Feistel cipher - A special class of iterated block ciphers where the ciphertext is calculated from the plaintext by repeated application of the same transformation called a *round function*.

field - A mathematical structure with multiplication and addition that behave as they do with the real numbers. A mathematical structure with the following algebraic properties.

> A nonempty set F is a field if:
>
> 1) F is *closed* under two binary operators denoted by + and * usually referred to as addition and multiplication respectively. Closure means that for any two elements f, h in F, $f+h$ and $f*h$ are in F.
>
> 2) F forms a *commutative group* (see the definition of a group) with respect to +.
>
> 3) F - {0} forms a *commutative group* (see the definition of a group) with respect to *.
>
> 4) The operator * distributes over the operator +, that is $a*(b+c) = a*b+a*c$.

FIPS - Federal Information Processing Standards. See NIST.

flat keyspace - See Linear Key Space.

function - A mathematical relationship between two values called the input and the output, such that for each input there is precisely one output.

Galois field - A finite field.

general-purpose factoring algorithm - An algorithm whose running time depends only on the size of the number being factored. See special purpose factoring algorithm.

Goppa code - A class of error correcting codes, used in the McEliece public-key cryptosystem.

graph - In mathematics, a set of points called nodes (or vertices) and a set of lines connecting them or some subset of them to one another called edges.

graph coloring problem - The problem of determining whether a graph can be colored with a fixed set of colors such that no two adjacent vertices have the same color and producing such a coloring.

group - A mathematical structure in which elements are combined.

A nonempty set G is a group if:

1) The set G is closed under the binary operator *, that is, for any two elements g, h in G, $g*h$ is in G.

2) The operator * is associative, that is, for any a, b, c in G, $a*(b*c) = (a*b)*c$.

3) There exists an identity element e in G, such that for any element g in G, $g*e = e*g = g$.

4) For every element g in G there is an inverse h in G such that, $g*h = h*g = e$, the identity.

GSS-API - generic security service application program interface.

hacker - A person who tries and/or succeeds at defeating computer security measures.

Hamiltonian path problem - A *Hamiltonian path* is a path through a graph that passes through each vertex exactly once. The associated problem is given a graph G is there a Hamiltonian path. This is a hard problem.

handshake - A protocol two computers use to initiate a communication session.

hard problem - A computationally-intensive problem; a problem that is computationally difficult to solve.

hash-based MAC - MAC that uses a hash function to reduce the size of the data it processes.

hash function - A function that takes a variable sized input and has a fixed size output.

HMAC - see MAC.

hyperplane - A mathematical object which may be thought of as an extension (into higher dimensions) of a 3 dimensional plane passing through the point (0,0,0).

IEEE - Institute of Electrical and Electronics Engineers, a body that creates some cryptography standards.

iKP - Internet Keyed Payments Protocol.

ISO - International Standards Organization, creates international standards, including cryptography standards.

identification - A process through which one ascertains the identity of another person or entity.

impersonation - Occurs when an entity pretends to be someone or something it is not.

import encryption - Encryption, in any form, coming into a country.

index calculus - A method used to solve the discrete log problem.

integer programming problem - The problem is to solve a linear programming problem where the variables are restricted to integers.

interactive proof - 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. This is usually done in a question response format, where the verifier asks the prover questions that only the prover can answer with a certain success rate.

Internet - The connection of computer networks from all over the world forming a worldwide network.

intractable - In complexity theory, referring to a problem with no efficient means of deriving a solution.

ITAR - International Traffic in Arms Regulations.

ITEF - Internet Engineering Task Force.

ITU-*T* - International Telecommunications Union - Telecommunications standardization sector.

Kerberos - An authentication service developed by the Project Athena team at MIT.

key - A string of bits used widely in cryptography, allowing people to encrypt and decrypt data; a key can be used to perform other mathematical operations as well. Given a cipher, a key determines the mapping of the plaintext to the ciphertext. See also distributed key, private key, public key, secret key, session key, shared key, sub key, symmetric key, weak key.

key agreement - A process used by two or more parties to agree upon a secret symmetric key.

key escrow - The process of having a third party hold onto encryption keys.

key exchange - A process used by two more parties to exchange keys in cryptosystems.

key expansion - A process that creates a larger key from the original key.

key generation - The act of creating a key.

key management - The various processes that deal with the creation, distribution, authentication, and storage of keys.

key pair - The full key information in a public-key cryptosystem, consisting of the public key and private key.

key recovery - A special feature of a key management scheme that allows messages to be decrypted even if the original key is lost.

key schedule - An algorithm that generates the subkeys in a block cipher.

keyspace - The collection of all possible keys for a given cryptosystem. See also flat keyspace, linear key space, nonlinear key space, and reduced key space.

knapsack problem - A problem that involves selecting a number of objects with given weights from a set, such that the sum of the weights is maximal but less than a pre-specified weight.

known plaintext attack - A form of cryptanalysis where the cryptanalyst knows both the plaintext and the associated ciphertext.

lattice - A lattice can be viewed as an N-dimensional grid.

LEAF - Law Enforcement Agency Field a component in the Clipper Chip.

life cycle - The length of time a key can be kept in use and still provide an appropriate level of security.

linear complexity - Referring to a sequence of 0's and 1's, the size of the smallest linear feedback shift register (LFSR) that would replicate the sequence. See also linear feedback shift register.

linear cryptanalysis - A known plaintext attack that uses linear approximations to describe the behavior of the block cipher. See known plaintext attack.

linear keyspace - A key space where each key is equally strong.

LFSR - linear feedback shift register. Used in many keystream generators because of its ability to produce sequences with certain desirable properties.

MAC - See message authentication code.

meet-in-the-middle attack - A known plaintext attack against double encryption with two separate keys where the attacker encrypts a plaintext with a key and "decrypts" the original ciphertext with another key and hopes to get the same value.

Message Authentication Code(MAC) - A MAC is a function that takes a variable length input and a key to produce a fixed-length output. See also hash-based MAC, stream-cipher based MAC, and block-cipher based MAC.

message digest - The result of applying a hash function to a message.

MHS - Message Handling System.

middle-person attack - A person who intercepts keys and impersonates the intended recipients.

MIME - Multipurpose Internet Mail Extensions.

MIPS - Millions of Instructions Per Second, a measurement of computing speed.

MIPS-Year - One year's worth of time on a MIPS machine.

mixed integer programming - The problem is to solve a linear programming problem where some of the variables are restricted to being integers.

modular arithmetic - A form of arithmetic where integers are considered equal if they leave the same remainder when divided by the modulus.

modulus - The integer used to divide out by in modular arithmetic.

multiple polynomial quadratic sieve(MPQS) - A variation of the quadratic sieve that sieves on multiple polynomials to find the desired relations. MPQS was used to factor RSA-129.

NIST - National Institute of Standards and Technology, a United States agency that produces security and cryptography related standards (as well as others); these standards are published as FIPS documents.

non-repudiation - A property of a cryptosystem. Non-repudiation cryptosystems are those in which the users cannot deny actions they performed.

nondeterministic - Not determined or decided by previous information.

nondeterministic computer - Currently only a theoretical computer capable of performing many computations simultaneously.

nondeterministic polynomial running time (*NP*) - If the running time, given as a function of the length of the input, is a polynomial function when running on a theoretical, nondeterministic computer, then the algorithm is said to be NP.

nonlinear keyspace - A key space comprised of strong and weak keys.

NSA - National Security Agency. A security-conscious U. S. government agency whose mission is to decipher and monitor foreign communications.

number field sieve - A method of factoring, currently the fastest general purpose factoring algorithm published. It was used to factor RSA-130.

number theory - A branch of mathematics that investigates the relationships and properties of numbers.

OAEP - Optimal Asymmetric Encryption Padding; a provably secure way of encrypting a message.

one-time pad - A secret key cipher in which the key is a truly random sequence of bits that is as long as the message itself, and encryption is performed by XORing the message with the key. This is theoretically unbreakable.

one-way function - A function that is easy to compute in one direction but quite difficult to reverse compute (compute in the opposite direction.)

one-way hash function - A one-way function that takes a variable sized input and creates a fixed size output.

patent - The sole right, granted by the government, to sell, use, and manufacture an invention or creation.

PKI - Public key Infrastructure. PKIs are designed to solve the key management problem. See also key management.

padding - Extra bits concatenated with a key, password, or plaintext.

password - A character string used as a key to control access to files or encrypt them.

PKCS - Public-key cryptography Standards. A series of cryptographic standards dealing with public key issues, published by RSA Laboratories.

plaintext - The data to be encrypted.

plane - A geometric object defined by any three non-colinear points, containing every line passing through any two of them.

Pollard p-1 and Pollard p+1 methods - Algorithms that attempt to find a prime factor p of a number N by exploiting properties of p-1 and p+1. See also factoring, prime factor, prime number.

Pollard Rho method - A method for solving the discrete logarithm and elliptic curve discrete logarithm.

polynomial - An algebraic expression written as a sum of constants multiplied by different powers of a variable, for example $a_n x^n + a_{n-1} x^{n-}1 + ... + a_1 x^1 + a_0$, where the a_j are the constants and x is the variable.

polynomial running time - If the running time, given as a function of the length of the input is a polynomial the algorithm is said to have polynomial running time. Polynomial running time algorithms are sub-exponential, but not all sub-exponential algorithms are polynomial running time.

precomputation attack - An attack where the adversary precomputes a look-up table of values used to crack encryption or passwords. See also dictionary attack.

primality testing - A test that determines, with varying degree of probability, whether or not a particular number is prime.

prime factor - A prime number that is a factor of another number is called a prime factor of that number.

prime number - Any integer greater than 1 that is divisible only by 1 and itself.

privacy - The state or quality of being secluded from the view and or presence of others.

private exponent - The private key in the RSA public-key cryptosystem.

private key - In public-key cryptography, this key is the secret key. It is primarily used for decryption but is also used for encryption with digital signatures.

proactive security - A property of a cryptographic protocol or structure which minimizes potential security compromises by refreshing a shared key or secret.

probabilistic signature scheme (PSS) - A provably secure way of creating signatures using the RSA algorithm.

protocol - A series of steps that two or more parties agree upon to complete a task.

provably secure - A property of a digital signature scheme stating that it is provably secure if its security can be tied closely to that of the cryptosystem involved. See also digital signature scheme.

pseudorandom number - A number extracted from a pseudorandom sequence.

pseudorandom sequence - A deterministic function which produces a sequence of bits with qualities similar to that of a truly random sequence.

PSS - See probabilistic signature scheme.

public exponent - The public key in the RSA public-key cryptosystem.

public key - In public-key cryptography this key is made public to all, it is primarily used for encryption but can be used for verifying signatures.

public-key cryptography - Cryptography based on methods involving a public key and a private key.

quadratic sieve - A method of factoring an integer, developed by Carl Pomerance.

quantum computer - A theoretical computer based on ideas from quantum theory; theoretically it is capable of operating nondeterministically.

RSA algorithm - A public-key cryptosystem based on the factoring problem. RSA stands for Rivest, Shamir and Adleman, the developers of the RSA public-key cryptosystem and the founders of RSA Data Security, Inc.

random number - As opposed to a pseudorandom number, a truly random number is a number produced independently of its generating criteria. For cryptographic purposes, numbers based on physical measurements, such as a Geiger counter, are considered random.

reduced keyspace - When using an n bit key, some implementations may only use $r < n$ bits of the key; the result is a smaller (reduced) key space.

relatively prime - Two integers are relatively prime if they have no common factors, i.*e*. (14, 25).

reverse engineer - To ascertain the functional basis of something by taking it apart and studying how it works.

rounds - The number of times a function, called a round function, is applied to a block in a Feistel cipher.

running time - A measurement of the time required for a particular algorithm to run as a function of the input size. See also exponential running time, nondeterministic polynomial running time, polynomial running time, and sub-exponential running time.

S-HTTP - Secure HyperText Transfer Protocol, a secure way of transferring information over the World Wide Web.

S/MIME - Secure Multipurpose Internet Mail Extensions.

SSL - Secure Socket Layer. A protocol used for secure Internet communications.

SWIFT - Society for Worldwide Interbank Financial Telecommunications.

salt - A string of random (or pseudorandom) bits concatenated with a key or password to foil precomputation attacks.

satisfiability problem - Given a Boolean expression determine if there is an assignment of 1's and 0's such that the expression evaluates to 1. This is hard problem.

secret key - In secret-key cryptography, this is the key used both for encryption and decryption.

secret sharing - Splitting a secret (*e.g*. a private key) into many pieces such that any specified subset of *N* pieces may be combined to form the secret.

secure channel - A communication medium safe from the threat of eavesdroppers.

seed - a typically random bit sequence used to generate another, usually longer pseudorandom bit sequence.

self-shrinking generator - A stream cipher where the output of an LFSR is allowed to feed back into itself.

self-synchronous - Referring to a stream cipher, when the keystream is dependent on the data and its encryption.

session key - A key for symmetric-key cryptosystems which is used for the duration of one message or communication session

SET - Secure Electronic Transaction. MasterCard and Visa developed (with some help from industry) this standard jointly to insure secure electronic transactions.

shared key - The secret key two (or more) users share in a symmetric-key cryptosystem.

shrinking generator - A stream cipher built around the interaction of the outputs of two LFSRs. See also stream cipher and linear feedback shift register.

Skipjack - The block cipher contained in the Clipper chip designed by the NSA.

SMTP - Simple Mail Transfer Protocol.

smartcard - A card, not much bigger than a credit card, that contains a computer chip and is used to store or process information.

special-purpose factoring algorithm - A factoring algorithm which is efficient or effective only for some numbers. See also factoring and prime factors.

standards - Conditions and protocols set forth to allow uniformity within communications and virtually all computer activity.

stream cipher - A secret key encryption algorithm that operates on a bit at a time.

stream cipher based MAC - MAC that uses linear feedback shift registers (LFSR's) to reduce the size of the data it processes.

strong prime - A prime number with certain properties chosen to defend against specific factoring techniques.

sub-exponential running time - The running time is less than exponential. Polynomial running time algorithms are sub-exponential, but not all sub-exponential algorithms are polynomial running time.

sub key - A value generated during the key scheduling of the key used during a round in a block cipher.

subset sum problem - A problem where one is given a set of numbers and needs to find a subset that sums to a particular value.

S/WAN - Secure Wide Area Network.

symmetric cipher - An encryption algorithm that uses the same key is used for encryption as decryption.

symmetric key - See secret key.

synchronous - A property of a stream cipher, stating that the keystream is generated independently of the plaintext and ciphertext.

tamper resistant - In cryptographic terms, this usually refers to a hardware device that is either impossible or extremely difficult to reverse engineer or extract information from.

TCSEC - Trusted Computer System Evaluation Criteria.

threshold cryptography - Splitting a secret (for example a private key) into many pieces such that only certain subsets of the *N* pieces may be combined to form the secret.

timestamp - see digital timestamp

tractable - A property of a problem, stating that it can be solved in a reasonable amount of time using a reasonable amount of space.

trap door one-way function - A one-way function that has an easy-to-compute inverse if you know certain secret information. This secret information is called the *trap door*.

traveling salesman problem - A hard problem. The problem is: given a set of cities, how does one tour all the cities in the minimal amount of distance traveled.

trustees - A common term for escrow agents.

Turing machine - A theoretical model of a computing device, devised by Alan Turing.

verification - The act of recognizing that a person or entity is who or what it claims to be.

Vernam cipher - See one-time pad.

weak key - A key giving a poor level in security, or causing regularities in encryption which can be used by cryptanalysts to break codes.

WWW - World Wide Web.

XOR - A binary bitwise operator yielding the result one if the two values are different and zero otherwise.

zero knowledge proofs - An interactive proof where the prover proves to the verifier that he or she knows certain information without revealing the information.

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.

[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 263277, Springer-Verlag, 1995.

[AHU74] Aho, Hopcroft, and Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.

[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.

[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.

[ARV95] *W*. Aiello, S. Rajagopalan, and *R*. Venkatesan. Design of practical and provably good random number generators (extended abstract). In Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 1-9, San Francisco, California, 22-24 January 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): 2531, July 1992.

[BBB92] C. Bennett, F. Bessette, G. Brassard, L. Savail, and J. Smolin. Experimental quantum cryptography. Journal of Cryptology, 5(1): 328, 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.

[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.

[BDN97] W. Burr, D. Dodson, N. Nazario, and W. T. Polk. MISPC, Minimum Interoperability Specification for PKI Components, Version 1. NIST, June 5, 1997.

[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 289299, 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 329334, Springer-Verlag, 1993.

[Bih95] E. Biham. Cryptanalysis of Multiple Modes of Operation. In Advances in Cryptology Asiacrypt '94, pages 278292, Springer-Verlag, 1995.

[BK98] A. Biryukov and E. Kushilevitz. Improved cryptanalysis of RC5. EuroCrypt '98. To appear.

[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.

[Bla94] Matt Blaze, Protocol Failure in the Escrowed Encryption Standard, Proceedings of the 2nd ACM Conference on Computer and Communications Security, pp 59-67, held in Fairfax, VA, Nov. 1994.

[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 bn ± 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.

[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.

[BR93] *M*. Bellare and *P*. Rogaway, "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols," Proceedings of the first Annual Conference on Computer and Communications Security, ACM, 1993.

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

[BR96] *M*. Bellare and *P*. Rogaway, "The Exact Security of Digital Signatures How to Sign with RSA and Rabin," Advances in Cryptology Eurocrypt `96 Proceedings, Lecture Notes in Computer Science Vol.1070, U. Maurer ed., Springer-Verlag, 1996.

[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): 1620, 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): 14, 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 221, 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.

[CGH97] Canetti, R. Gennaro, A. Herzberg and D. Naor, "Proactive Security: Long-term Protection Against Break-ins". CryptoBytes 3(1): 1-8, 1997.

[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.

[CJ98] F. Chabaud and A. Joux. Differential Collisions in SHA-0. In Advances in Cryptology Crypto '98, pages 56-71, Springer-Verlag, 1998.

[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: 115, 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): 901909, 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.

[DB96] - Denning, D. and Branstad, D. “A Taxonomy for Key Escrow Encryption Systems.” Communications of the ACM, Vol. 39, No. 3, March 1996, pp 34-40.

[DB96b] - H. Dobbertin. The Status of MD5 After a Recent Attack. CryptoBytes Vol.2 No.2, Summer 1996.

[DBP96] H. Dobbertin, A. Bosselaers, and B. Preneel. RIPEMD-160: A strengthened version of RIPEMD.In Proceedings of 3rd International Workshop on Fast Software Encryption, pages 71-82, Springer-Verlag, 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 ciphertextsTowards 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.

[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.

[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.

[EGM89] Even, S., Goldreich, *O*., and Micali, S., "On-Line/Off-Line Digital Signatures", Advances in Cryptology Crypto '89, *G*. Brassard (ed.), Springer-Verlag, 1990, pp. 263-275.

[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. http://www.netscape.com/newsref/std/credit.htm

[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.

[FGM97] *Y*. Frankel, *P*. Gemmel, *P*. *D*. MacKenzie and *M*. Yung, "Proactive RSA," Advances in Cryptology Crypto '97 Proceedings, Lecture Notes in Computer Science Vol. 1294, B. Kaliski ed., Springer-Verlag, 1997.

[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.

[GJ79] Michael *R*. Garey and David S. Johnson. Computers and Intractability - A Guide to the Theory of NP-Completeness. *W.H*. Freeman, New York, 1979.

[GJK96] *R*.Gennaro, S. Jarecki, *H*. Krawczyk and *T*. Rabin, "Robust Threshold DSS Signatures," Advances in Cryptology - Eurocrypt '96 Proceedings, Lecture Notes in Computer Science Vol. 1070, Ueli Maurer ed., Springer-Verlag, 1996.

[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): 124138, 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: 336-341, 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)

[HJJ97] A. Herzberg, *M*. Jakobsson, S. Jarecki, *H*. Krawczyk and *M*. Yung, "Proactive Public Key and Signature Systems," In 1997 ACM Conference on Computers and Communication Security, 1997.

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

[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.

[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.

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

[KM96] L.R. Knudsen and W. Meier. Improved differential attacks on RC5," Advances in Cryptology Crypto '96 Proceedings, Lecture Notes in Computer Science Vol. 1109, pages 216-228. Springer-Verlag 1996.

[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 International 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.

[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.

[Koc95] Ç.K. Koç. RSA Hardware Implementation. Technical Report TR-801, version 1.0, RSA Laboratories, August 1995.

[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 International 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: 336, 1988.

[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 1725, 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.

[LM91] *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.

[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.

[LP98] Harry *R*. Lewis and Christos *H*. Papadimitriou. Elements of the Theory of Computation, 2nd edition, Prentice Hall, Upper Saddle River, NJ, 1998.

[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 International 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 4244, 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.

[Men95] A. Menezes. Elliptic Curve Cryptosystems. CryptoBytes, 1(2): 1-4, 1995.

[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. EEE 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.

[MS95a] P. Metzger and W. Simpson. RFC 1828: IP Authentication using Keyed MD5. Piermont and Daydreamer, August 1995.

[MS95b] 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 43944, 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.

[OG97] Architecture for Public-Key Infrastructure (APKI) Draft 1. The Open Group, May 1997. http://www.opengroup.org/security/pki/apki_1-0.{ps, pdf}.

[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.

[PV95] B. Preneel and *P*.C. van Oorschot. MDx-MAC and Building Fast MACs from Hash Functions. In Advances in Cryptology Crypto '95, pages 1-14, Springer-Verlag, 1995.

[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 International 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 303311, Springer-Verlag, 1991.

[Riv92a] *R*.L. Rivest. Response to NIST's proposal. Communications of the ACM, 35: 4147, 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.

[RK96] Joe Kilian and Phillip Rogaway. How to protect DES against exhaustive key search. In Advances in Cryptology Crypto '96 Proceedings, pages 252-267, Springer-Verlag, 1996.

[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.

[Rob96] - MJ.B. Robshaw. On Recent Results for MD2, MD4 and MD5. RSA Laboratories Bulletin No. 4. November 12, 1996.

[Rog96] *P*. Rogaway. The security of DESX. CryptoBytes, 2(2):8-11, 1996.

[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.

[RY97] *M*. Robshaw and *Y*. Yin. Elliptic Curve Cryptosystems. An RSA Laboratories Technical Note. Revised June 27, 1997

[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 International 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.

[Sch96b] B. Schneier. Applied Cryptography: Protocols, Algorithms, and Source Code in C. Wiley, 2nd Edition, 1995.

[Sel98] A. A. Selcuk, New results in linear cryptanalysis of RC5. In Proceedings of 5th International Workshop on Fast Software Encryption, pages 1-16, Springer-Verlag, 1998.

[SH95] C.*P*. Schnorr and *H.H*. Hörner. Attacking the Chor-Rivest cryptosystem by improved lattice reduction. In Advances in Cryptology Eurocrypt '95, pages 112, 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.

[Sim83] *G*. J. Simmons, "The Prisoner's Problem and the Subliminal Channel," Advances in Cryptology Crypto '83 Proceedings, Plenum Press, *D*. Chaum.

[Sim92] *G*.J. Simmons, editor. Contemporary Cryptology The Science of Information Integrity. IEEE Press, 1992.

[Sim93a] *G*. J. Simmons, "Subliminal Communication is Easy Using DSA," Advances in Cryptology Eurocrypt '93 Proceedings, Lecture Notes in Computer Science Vol. 765, *T*. Helleseth ed., Springer-Verlag, 1993.

[Sim93b] *G*. J. Simmons, "The Subliminal Signatures in the U.S. Digital Signature Algorithm (DSA)," presented at the 3rd Symposium on State and Progress of Research in Cryptography, Rome, Italy, February 15-16, 1993.

[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.

[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 TR244, School of Computer Science, Carleton University, Ottawa, Canada, May 1994.

[Wie98] *M*.J. Wiener. Performance Comparison of Public-Key Cryptosytstems. CryptoBytes, 3(3): 1-5, 1998.

[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.

[Yin97] *Y*. Yin. The RC5 encryption algorithm: two years on. CryptoBytes, 2(3):14-16, 1997.

[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.

[Yuv79] *G*. Yuval. How to swindle Rabin. *Cryptologia*, July 1979.

[Yin97] *Y*. Yin. The RC5 encryption algorithm: two years on. *CryptoBytes*, 2(3):14-16, 1997.

[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.