NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

# MESSAGE AUTHENTICATION AND DIGITAL SIGNATURE

Graudation Project
COM – 400

Student: Tekin Tekin (20010672)

Supervisor: Prof.Dr.Fahreddin Mamedov SADIKOĞLU

Nicosia - 2004

## ACKNOWLEDGEMENTS

First of all, I would like to thank every one for their support and help they gave me during my preparation of these thesis.

I would like to thank my supervisor Prof. Dr. Fahreddin Mamedov Sadıkoğlu. Under his guidance, I successfully overcome many difficulties and learn a lot about Cryptography and Cryptosystems. I asked him many questions in Cryptography, he explained my questions patiently.

I would like to express my gratitude to Vice-President Prof. Dr. Şenol Bektaş, because he helped to me at each stage of my Undergraduate Education in Near East University.

I also wish to thank Mr. Ümit İlhan at my Undergraduate Education for his invaluable advices, for his help and for his patience also for his support.

I would like to say how grateful I am to Osman Tekin and Onur Taha Cananer for their help and support.

Finally, I want to thank especially to the people I have mentioned above and to my family, to Cemal Kavalcıoğlu, Ali Özgen and Ömer Gümüş for their endless support. I could never have prepared this thesis without the encouragement and support.

# LIST OF ABBREVIATIONS

| | | |
|---|---|---|
| **AECA** | **:** | Arms Export Control Act |
| **AES** | **:** | Advanced Encryption Standard |
| **CA** | **:** | Certificate Authorities |
| **CBC** | **:** | Cipher Block Chaining |
| **CFB** | **:** | Cipher Feedback |
| **CRL** | **:** | Certificate Revocation List |
| **DES** | **:** | Digital Encryption Standard |
| **DSA** | **:** | Digital Signature Algorithm |
| **DSS** | **:** | Digital Signature Standard |
| **EAR** | **:** | Export Administration Regulations |
| **ECB** | **:** | Electronic Code Book |
| **ECC** | **:** | Elliptic Curve Cryptosystem |
| **FCS** | **:** | Frame Check Sequence |
| **IDEA** | **:** | International Data Encryption Method |
| **ITAR** | **:** | International Traffic in Arms Regulations |
| **KDC** | **:** | Key Distribution Center |
| **MAC** | **:** | Message Authentication Code |
| **NIST** | **:** | National Institute of Standards and Technology |
| **NSA** | **:** | National Security Agency |
| **ODTC** | **:** | Office of Defense Trade Controls |
| **OFB** | **:** | Output Feedback |
| **PKI** | **:** | Public Key Infrastructure |
| **RA** | **:** | Registration Authorities |
| **SHA** | **:** | Secure Hash Algorithm |
| **SMTP** | **:** | Simple Mail Transfer Protocol |
| **SNMP** | **:** | Simple Network Management Protocol |
| **USML** | **:** | United States Munitions |

## ABSTRACT

People mean different things when they talk about cryptography. Children play with toy ciphers and secret languages. However, these have little to do with real security and strong encryption. Strong encryption is the kind of encryption that can be used to protect information of real value against organized criminals, multinational corporations, and major governments. Strong encryption used to be only military business; however, in the information society it has become one of the central tools for maintaining privacy and confidentiality.

As we move into an information society, the technological means for global surveillance of millions of individual people are becoming available to major governments. Cryptography has become one of the main tools for privacy, trust, access control, electronic payments, corporate security, and countless other fields.

Cryptography is no longer a military thing that should not be messed with. It is time to de-mystify cryptography and make full use of the advantages it provides for the modern society.

In the following, basic terminology and the main methods of cryptography are presented. Any opinions and evaluations neither presented here are speculative, and neither the authors nor SSH can be held responsible for their correctness although every attempt is made to make sure that this information is as correct and up-to-date as possible.

# TABLE OF CONTENTS

# INTRODUCTION

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

But today's cryptography is more than secret writing, more than encryption and decryption. Authentication is as fundamental a part of our lives as privacy.

The Thesis Consists of Introduction, Four chapter and Conclusion:

The Chapter one; Cryptography, to most people, is concerned with keeping communications private. Indeed, the protection of sensitive communications has been the emphasis of cryptography throughout much of its history.

Chapter two; Traditional cryptography is based on the sender and receiver of a message knowing and using the same secret key: the sender uses the secret key to encrypt the message, and the receiver uses the same secret key to decrypt the message. Asymmetric cryptosystems use one key (the public key) to encrypt a message and a different key (the private key) to decrypt it.

Chapter three; This chapter begins with an introduction to the requirements for authentication and digital signature and the types of attacks to be countered. Then the basic approaches are surveyed, including the increasingly important area of source hash functions.

Chapter four; This chapter begins with an overview of digital signatures. Then we look at authentication protocols, many of which depend on the use of the digital signature. Finally, we introduce the Digital Signature Standard (DSS).

# 1. CONSTITUTIONAL CHALLENGES TO CRYPTOGRAPHIC REGULATIONS

Our Founding Fathers penned the First Amendment over two hundred years ago, and its speech protections are applicable today to regulations of electronic speech. Although technology has radically changed since 1791, the Speech Clause has always kept pace with new technology and the free exchange of ideas and information. It is fitting that as we approach the twenty-first century--an era denoted as the Information Age that the First Amendment be given the opportunity to flex its muscles with regard to the Internet.

The Internet is a vast wealth of ideas and expression which draws its strength from its diversity. The Internet allows people from across the globe to come together to do business, debate worldly events, and share discoveries without regard to distances or borders. The accessibility of cyberspace has enabled more people to take active roles in communication because of the ease in placing information at the fingertips of others. Thus, people have become active producers and publishers of information on practically any topic imaginable.

Although technology has opened new First Amendment doors to promote free speech, it has also created new privacy concerns. Because much of today's electronic communication occurs in the form of e-mail, modern technology allows those messages to be tracked and stored by unintended recipients--namely the government. In addition, as more commerce takes place online, vital information about personal financial condition or personal tastes and preferences may become available to anyone with the motive to take advantage of the unsuspecting. To prevent Internet communication and commerce from becoming no more private than mailing a post card, technology has yet again delivered an answer.

Encryption technologies serve as the locks and keys of cyberspace. Cryptography has created new opportunities to protect our private communications and intimate information so that this electronic medium can continue to grow. Industry and commerce can prosper with the assurance that information and trade secrets can be transferred electronically with security. However, the increasing popularity of encryption technology has raised the ire of the government in the name of national security. In an effort to control the rapid growth of cryptography, the government has enacted laws controlling cryptography's development and dissemination. The laws have the effect of inhibiting the free flow of ideas among people who wish to communicate in this manner. The existing laws remove an entire area of communication from public debate and pose the potential to bar the First Amendment from electronic communication.

This Article focuses on the constitutional issues surrounding the development of cryptographic technology and suggests that existing regulations fail to pass constitutional muster. Three cases have arisen in the federal courts challenging governmental restrictions on the development and dissemination of cryptography, and the courts have taken contrasting views of the First Amendment issues involved. Because of the importance of these issues and the potential effects of divergent rulings in lower courts, the Supreme Court may have to make the final decision. This Article asserts that if this issue reaches the Supreme Court, the Court should find the cryptographic regulations to be an unconstitutional suppression of free speech. Moreover, this Article proposes that the current regulations be stricken in favor of pending legislation before Congress.

# 2. CRYPTOGRAPHY AND CRYPTOSYSTEMS

## 2.1 What Is Cryptography?

Cryptography is the art of creating and using methods of disguising messages, using codes, ciphers, and other methods, so that only certain people can see the real message. The process of disguising the substance of messages into incomprehensible data is called encryption. The encryption process converts the undisguised message, or plaintext, into unintelligible cipher text. After the message has been encrypted, it may be transformed back to plaintext in a process called decryption. The tool which performs the conversion is a cipher, which is a method of encryption that utilizes a mathematical algorithm to convert any text regardless of its content. As an added level of security, today's algorithms use a key which consists of a sequence of computer code to activate the algorithm to encrypt and decrypt messages. The key is input into the algorithm to successfully perform the desired conversion.

The strength of a coded communication is greatly dependent upon the key, for the algorithm itself is worthless without the key to decrypt the message. Early encryption techniques employed a single key system that was required to both encrypt and decrypt the message. This type of system was vulnerable because a separate key was needed for each pair of users who exchanged messages, and both sides had to keep the key secret to keep the system secure.

In the mid 1980s, a more secure key system was developed to solve the single key exchange problem. The system of public key cryptography was created to utilize a public and a private key to encrypt and decrypt messages. Under this scheme, each party establishes a unique private key which only the owner knows and a unique public key which everyone knows. Public keys may be published freely in directories similar to phone books to aid senders in locating a potential recipient's public key, but private keys must be kept secret by their owner.

Consider the following example: Sam completes a message to Ruth in plaintext form. Upon completion, Sam encodes the message with Ruth's public key. When Ruth receives the message in cipher text from Sam, she uses her private key to decode the message into plaintext. To send a message back to Sam, Ruth encodes her message with the use of Sam's public key. Sam then uses his private key to decode the message.

Ruth and Sam have not compromised their private keys. Knowledge of the public key in no way compromises the identity of the private key. The system is extremely secure, as virtually the only way to break security is for either Ruth or Sam to give away their private keys. Public key cryptographic technology has delivered military-grade cryptography with the level of security so high that even the ultra-secret, code-breaking computers at the National Security Agency cannot decipher the encrypted messages.

## 2.2 Who Uses Cryptography?

One of the earliest examples of cryptography was used by Julius Caesar when he sent military messages to his armies. Perhaps since that time, people have also tried to decode encrypted messages. Allies in World War II were able to break a secret German code called Enigma. This discovery enabled Allied forces to locate and sink many German U-boats; moreover, they were able to obtain advanced information about German military operations that was critical to the campaign in Europe. Similar code-breaking ability also allowed the United States Navy to intercept the Japanese fleet in one of the most decisive battles in the Pacific--The Battle of Midway. These are just a few examples of how cryptographic technology has played an important role in history.

Until recently, cryptography has primarily been the vital and exclusive tool of governments, not the public; however, a demand for private encryption technology has arisen with the growth of advanced computer technology. Today, many individuals and businesses want or need secure communications. For example, encryption is heavily used in the banking industry to ensure the security of electronic fund transfers. In 1994, an international group of criminals attempted to electronically steal twelve million dollars from Citicorp. As a result of the attempted heist, financial institutions around the world increased their authentication capabilities for electronic fund transfers. Banks also encrypt ATM customer identification numbers and the data on the cards to prevent unauthorized modification and forgery. As targets of industrial espionage, many U.S. corporations seek to secure communications to protect their intellectual property and other sensitive market information. Exponential growths in the Internet and the popularity of e-mail have given rise to encryption needs. Because cryptography can deliver secure transactions and communications on an unsecured worldwide computer network, the technology is essential to the commercial expansion of the Internet.

## 2.3 The Government's View of Cryptography

The early uses of cryptography were primarily for intelligence gathering and securing military communications, the Defense Department, through the National Security Agency (NSA), has played a key role in developing the science and controlling its use in the United States and abroad. The NSA has continuously attempted to control the development and expansion of cryptography in the private sector because it views the technology as a threat to national security. The NSA has tried to slow the growth and dissemination of cryptography by controlling public funding, patent publications, and presentation of scientific papers at academic conferences. To accomplish the NSA's task, the government has enacted export control laws to restrict the exportation and dissemination of encryption software.

One of the first laws enacted to regulate cryptography authorized the President, under the Arms Export Control Act (AECA), to control the export and import of defense articles and services by designating them as munitions on the United States Munitions List (USML). Regulatory responsibility for the AECA was vested in the Department of State, which instituted the International Traffic in Arms Regulations (ITAR) for administration of this task.

Once an item is placed on the USML, it must be licensed before it can be imported or exported. Requests to license items listed on the USML are made to the Office of Defense Trade Controls (ODTC), which considers requests on a case-by-case-basis. The ITAR provides for a commodity jurisdiction procedure allowing the ODTC to determine whether an article or service is covered by the USML. If an article is not listed on the USML, then it can be freely exported.

The USML's scope includes articles such as "military tanks, combat engineer vehicles, bridge launching vehicles, half-tracks and gun carriers." The USML also considers encryption technology as a "monition" having been "specifically designed, developed, configured, adapted, or modified for a military application. . . ."

The ITAR is not the only law controlling the development and dissemination of cryptography. In November 1996, President Clinton by Executive Order transferred jurisdiction over the export of nonmilitary encryption products to the Department of Commerce. The order removed encryption products that would qualify as defense articles under the USML and placed them on the Commerce Control List under the authority of the Export Administration Regulations (EAR).

Shortly after the President signed the order, the Commerce Department issued an interim rule regulating the export of encryption products. The Commerce Department declared that encryption items include all "encryption commodities, software, and technology that contain encryption features and are subject to the EAR." The EAR considers export as the downloading, or causing the downloading of software through Internet file transfer protocol locations, to bulletin boards, and on World Wide Web sites. To disseminate information subject to the EAR, one must obtain a license prior to any transmission.

Even with the EAR, encryption products with military application remain under the power of the ITAR. Because both the ITAR and EAR have control over cryptography, it is necessary to examine the constitutional ramifications of each to discover potential problems in the two laws.

## 2.4 Cryptosystems

There are two kinds of cryptosystems: symmetric *and* asymmetric. Symmetric cryptosystems use the same key (the secret key) to encrypt and decrypt a message, and asymmetric cryptosystems use one key (the public key) to encrypt a message and a different key (the private key) to decrypt it. Asymmetric cryptosystems are also called public key cryptosystems.

Symmetric cryptosystems have a problem: how do you transport the secret key from the sender to the recipient securely and in a tamperproof fashion? If you could send the secret key securely, then, in theory, you wouldn't need the symmetric cryptosystem in the first place because you would simply use that secure channel to send your message. Frequently, trusted couriers are used as a solution to this problem.

Another, more efficient and reliable solution is a public key cryptosystem, such as RSA, which is used in the popular security tool PGP.

## 2.4.1 Cryptanalysis and Attacks on Cryptosystems

Cryptanalysis is the art of deciphering encrypted communications without knowing the proper keys. There are many cryptanalytic techniques. Some of the more important ones for a system implementer are described below.

**Cipher text-only attack**: This is the situation where the attacker does not know anything about the contents of the message, and must work from cipher text only. In practice it is quite often possible to make guesses about the plaintext, as many types of messages have fixed format headers. Even ordinary letters and documents begin in a very predictable way. For example, many classical attacks use frequency analysis of the cipher text; however, this does not work well against modern ciphers.

Modern cryptosystems are not weak against cipher text-only attacks, although sometimes they are considered with the added assumption that the message contains some statistical bias.

**Known-plaintext attack**: The attacker knows or can guess the plaintext for some parts of the cipher text. The task is to decrypt the rest of the cipher text blocks using this information. This may be done by determining the key used to encrypt the data, or via some shortcut.

One of the best known modern known-plaintext attacks is linear cryptanalysis against block ciphers.

**Chosen-plaintext attack**: The attacker is able to have any text he likes encrypted with the unknown key. The task is to determine the key used for encryption. A good example of this attack is the differential cryptanalysis which can be applied against block ciphers.

Some cryptosystems, particularly RSA, are vulnerable to chosen-plaintext attacks. When such algorithms are used, care must be taken to design the application so that an attacker can never have chosen plaintext encrypted.

**Man-in-the-middle attack**: This attack is relevant for cryptographic communication and key exchange protocols. The idea is that when two parties, A and B, are exchanging keys for secure communication, an adversary positions himself between A and B on the communication line. The adversary then intercepts the signals that A and B send to each other, and performs a key exchange with A and B separately. A and B will end up using a different key, each of which is known to the adversary (hacker). The adversary can then decrypt any communication from A with the key he shares with A, and then resends the communication to B by encrypting it again with the key he shares with B. Both A and B will think that they are communicating securely, but in fact the adversary is hearing everything.

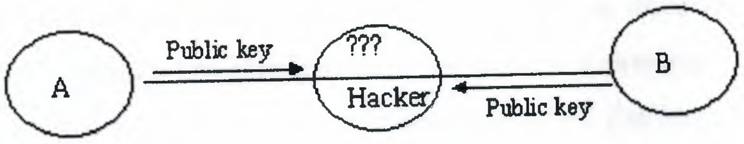


**Figure 2.1** Man in the middle attack

The usual way to prevent the man-in-the-middle attack is to use a public key cryptosystem capable of providing digital signatures. For set up, the parties must know each others public keys in advance. After the shared secret has been generated, the parties send digital signatures of it to each other. The man-in-the-middle can attempt to forge these signatures, but fails because he cannot fake the signatures.

Correlation between the secret key and the output of the cryptosystem is the main source of information to the cryptanalyst. In the easiest case, the information about the secret key is directly leaked by the cryptosystem. More complicated cases require studying the correlation between the observed information about the cryptosystem and the guessed key information.

For example, in linear attacks against block ciphers the cryptanalyst studies the known plain text and the observed cipher text. Guessing some of the key bits of the cryptosystem the analyst determines by correlation between the plaintext and the cipher text whether she guessed correctly. This can be repeated, and has many variations.

The differential cryptanalysis introduced by Eli Biham and Adi Shamir in late 1980's was the first attack that fully utilized this idea against block ciphers. Later Mitsuru Matsui came up with linear cryptanalysis which was even more effective against DES. More recently, new attacks using similar ideas have been developed.

The correlation idea is fundamental to cryptography and several researchers have tried to construct cryptosystems which are provably secure against such attacks.

**Attack against or using the underlying hardware**: in the last few years as more and smaller mobile crypto devices have come into widespread use, a new category of attacks has become relevant which aim directly at the hardware implementation of the cryptosystem.

The attacks use the data from very fine measurements of the crypto device doing, say, encryption and compute key information from these measurements. The basic ideas are then closely related to those in other correlation attacks. For instance, the attacker guesses some key bits and attempts to verify the correctness of the guess by studying correlation against her measurements.

Several attacks have been proposed such as using careful timings of the device, fine measurements of the power consumption, and radiation patterns. These measurements can be used to obtain the secret key or other kinds information stored on the device.

This attack is generally independent of the used crypto graphical algorithms and can be applied to any device that is not explicitly protected against it.

Faults in cryptosystems can lead to cryptanalysis and even the discovery of the secret key. The interests in crypto graphical devices lead to the discovery that some algorithms behaved very badly with the introduction of small faults in the internal computation.

For example, the usual implementations of RSA private key operations are very susceptible to fault attacks. It has been shown that by causing one bit of error at a suitable point can reveal the factorization of the modulus.

Similar ideas have been applied to a wide range of algorithms and devices. It is thus necessary that crypto graphical devices are designed to be highly resistant against faults.

**DNA cryptography**: Leonard Adleman (one of the inventors of RSA) came up with the idea of using DNA as computers. DNA molecules could be viewed as a very large computer capable of parallel execution. This parallel nature could give DNA computers exponential speed-up against modern serial computers.

There are unfortunately problems with DNA computers, one being that the exponential speed-up requires also exponential growth in the volume of the material needed. Thus in practice DNA computers would have limits on their performance. Also, it is not very easy to build one.

There are many other cryptographic attacks and cryptanalysis techniques. However, these are probably the most important ones for an application designer. Anyone contemplating to design a new cryptosystem should have a much deeper understanding of these issues.

## 2.5 Basic Terminology

Suppose that someone wants to send a message to a receiver, and wants to be sure that no-one else can read the message. However, there is the possibility that someone else opens the letter or hears the electronic communication.

In cryptographic terminology, the message is called plaintext or clear text. Encoding the contents of the message in such a way that hides its contents from outsiders is called encryption. The encrypted message is called the cipher text. The process of retrieving the plaintext from the cipher text is called decryption. Encryption and decryption usually make use of a key, and the coding method is such that decryption can be performed only by knowing the proper key.

Cryptography is the art or science of keeping messages secret. Cryptanalysis is the art of breaking ciphers, i.e. retrieving the plaintext without knowing the proper key. People who do cryptography are cryptographers, and practitioners of cryptanalysis are cryptanalysts.

Cryptography deals with all aspects of secure messaging, authentication, digital signatures, electronic money, and other applications. Cryptology is the branch of mathematics that studies the mathematical foundations of cryptographic methods.

## 2.6 Basic Cryptographic Algorithms

A method of encryption and decryption is called a cipher. Some cryptographic methods rely on the secrecy of the algorithms; such algorithms are only of historical interest and are not adequate for real-world needs. All modern algorithms use a key to control encryption and decryption; a message can be decrypted only if the key matches the encryption key.

There are two classes of key-based encryption algorithms, symmetric (or secret-key) and asymmetric (or public-key) algorithms. The difference is that symmetric algorithms use the same key for encryption and decryption (or the decryption key is easily derived from the encryption key), whereas asymmetric algorithms use a different key for encryption and decryption, and the decryption key cannot be derived from the encryption key.

Symmetric algorithms can be divided into stream ciphers and block ciphers. Stream ciphers can encrypt a single bit of plaintext at a time, whereas block ciphers take a number of bits (typically 64 bits in modern ciphers), and encrypt them as a single unit. Many symmetric ciphers are described on the algorithms page. Asymmetric ciphers (also called public-key algorithms or generally public-key cryptography) permit the encryption key to be public, allowing anyone to encrypt with the key, whereas only the proper recipient can decrypt the message. The encryption key is also called the public key and the decryption key the private key or secret key.

Modern cryptographic algorithms are no longer pencil-and-paper ciphers. Strong cryptographic algorithms are designed to be executed by computers or specialized hardware devices. In most applications, cryptography is done in computer software. Generally, symmetric algorithms are much faster to execute on a computer than asymmetric ones. In practice they are often used together, so that a public-key algorithm is used to encrypt a randomly generated encryption key, and the random key is used to encrypt the actual message using a symmetric algorithm. This is sometimes called hybrid encryption.

### 2.6.1 Types of Ciphers

**Block cipher:** Manipulate a group of bits. Typically implemented with software using substitution-box (S-box),

**Stream cipher:** Manipulate bit or byte. Typically in hardware.

Typically, a block-cipher method is implemented with software while a stream-cipher is in hardware format.

Here we will talk about some basic cipher types. Be aware that any simple ciphers are vulnerable to frequency analysis, which means often used words are easy to be guesses out. As a result, a polyalphabetic cipher is better than one alphabetic cipher to defeat frequency analysis

A strong cipher algorithm;

- Long period of no repeating pattern within key stream values,
- Statically unpredictable,
- The key stream is not linearly related to the key,
- Statically unbiased key stream (as many 0's as 1's)

**Substitution cipher:** It replaces the original information with other in formations

### 2.6.2 Strength of Cryptographic Algorithms

Good cryptographic systems should always be designed so that they are as difficult to break as possible. It is possible to build systems that cannot be broken in practice. This does not significantly increase system implementation effort; however, some care and expertise is required. There is no excuse for a system designer to leave the system breakable. Any mechanisms that can be used to circumvent security must be made explicit, documented, and brought into the attention of the end users.

In theory, any cryptographic method with a key can be broken by trying all possible keys in sequence. If using *brute force* to try all keys is the only option, the required computing power increases exponentially with the length of the key. A *32* bit key takes $2^{32}$ (about $10^{9}$) steps. This is something anyone can do on his/her home computer. A system with *40* bit keys takes $2^{40}$ steps - this kind of computation requires something like a week (depending on the efficiency of the algorithm) on a modern home computer. A system with *56* bit keys (such as DES) takes a substantial effort, but is easily breakable with special hardware. The cost of the special hardware is substantial but easily within reach of organized criminals, major companies, and governments. Keys with *64* bits are probably breakable now by major governments, and within reach of organized criminals, major companies, and lesser governments in few years. Keys with *80* bits appear good for a few years, and keys with *128* bits will probably remain unbreakable by brute force for the foreseeable future. Even larger keys are sometimes used.

However, key length is not the only relevant issue. Many ciphers can be broken without trying all possible keys. In general, it is very difficult to design ciphers that could not be broken more effectively using other methods. Designing your own ciphers may be fun, but it is not recommended for real applications unless you are a true expert and know exactly what you are doing.

One should generally be very wary of unpublished or secret algorithms. Quite often the designer is then not sure of the security of the algorithm, or its security depends on the secrecy of the algorithm. Generally, no algorithm that depends on the secrecy of the algorithm is secure. Particularly in software, anyone can hire someone to disassemble and reverse-engineer the algorithm. Experience has shown that the vast majority of secret algorithms that have become public knowledge later have been pitifully weak in reality.

The key lengths used in public-key cryptography are usually much longer than those used in symmetric ciphers. This is caused by the extra structure that is available to the cryptanalyst. There the problem is not that of guessing the right key, but deriving the matching secret key from the public key. In the case of RSA, this could be done by factoring a large integer that has two large prime factors. In the case of some other cryptosystems it is equivalent to computing the discrete logarithm modulo a large integer (which is believed to be roughly comparable to factoring when the module is a large prime number). There are public key cryptosystems based on yet other problems.

To give some idea of the complexity for the RSA cryptosystem, a *256* bit modulus is easily factored at home, and *512* bit keys can be broken by university research groups within a few months. Keys with *768* bits are probably not secure in the long term. Keys with *1024* bits and more should be safe for now unless major crypto graphical advances are made against RSA; keys of *2048* bits are considered by many to be secure for decades.

It should be emphasized that the strength of a cryptographic system is usually equal to its weakest link. No aspect of the system design should be overlooked, from the choice algorithms to the key distribution and usage policies.

### 2.6.3 Key Exchange Algorithm

Sometimes, people need secure communication to exchange keys. A couple of suggested algorithms is listed below.

**Algorithm 1: Diffie-Hellman,**

This is the first public-key algorithm. It involves exchanging keys. Alice and Bob know a large integer n and g (less then n, greater then 1). Assume these numbers are known by anyone.

1. Alice- generates large integer x, solves A=g^x mod n, sends A to Bob.
2. Bob- generates large integer y, solves B=g^y mod n, sends B to Alice.
3. Alice- solves K(A)=B^x mod n.
4. Bob- solves K(B)=A^y mod n
5. Both Alice and Bob have K(A)=K(B)

One thing to remember is that n has to be no smaller then 512 bits.

**Algorithm 2: Public-key cryptography**

This is an easy algorithm, and Alice can either ask Bob for his public key or get it from a database.

1. Alice- asks Bob for his public key (or gets it from a database), generates a *session key*, encrypts it with Bob's public key, sends it to Bob
2. Bob decrypts the session key with his private key
3. Alice and Bob share the same session key with which they can encrypt messages to each other

**Algorithm 3: Public-key cryptography**

This is probably the easiest public-key exchange algorithm created. It does not involve any session keys and is really straightforward.

1. Alice- Asks Bob for his public key
2. Bob- Asks Alice for her public key
3. Alice- Encrypts her message with Bob's public key, sends it to Bob
4. Bob decrypts Alice's message using his private key, encrypts his reply with Alice's public key
5. Alice decrypts Bob's reply with her private key and reads the message

**WARNING:** *Although, this is an easy algorithm, it is not at all safe. There is an attack that can destroy this algorithm's purpose. It is called man-in-the-middle attack. In a nutshell, when Alice and Bob exchange their public keys, the interceptor can substitute their public keys for his own.*

**Algorithm 4: Fooling man-in-the-middle attack**

There is one algorithm created by Ron Rivest and Adi Shamir that prevents the man-in-the-middle attack. It is called the interlock protocol. Although not fully secure, this algorithm has a good chance to prevent the man-in-the-middle.

1. Alice- Sends Bob her public key
2. Bob- Sends Alice his public key
3. Alice- Encrypts her message with Bob's private key, sends half of the message to Bob
4. Bob- Encrypts his message sends half of it to Alice.
5. Alice- Sends second part of her message to Bob
6. Bob- Decrypts Alice's message sends second half of his message to Alice
7. Alice- Decrypts Bob's message

**Algorithm 5: Symmetric Cryptography**

This algorithm requires a Key Distribution Center (KDC) to generate a random session key for Bob and Alice.

1. Alice- requests a session key from KDC
2. KDC- generates a session key, encrypts it with Alice's and Bob's public keys, sends both copies to Alice
3. Alice- decrypts her session key with her private key, sends Bob's copy to Bob
4. Bob- decrypts the received session key with his private key
5. Now Alice and Bob have the same session key to communicate with

Considering that Bob does not know Alice, she might want to include some info about her in Bob's copy of the session key.

**Algorithm 6: Message + key sending**

Alice can send Bob her message and the key in the same message.

1. Alice- Generates a random session key, encrypts her message with it, finds Bob's public key, encrypts session key with Bob's public key. Sends all of it to Bob.
2. Bob- decrypts the session key, decrypts message.

This algorithm can fall to a man-in-the-middle attack, if Alice gets the key of not Bob, but an impostor.

## 2.7 Cryptographic Hash Functions

Cryptographic hash functions are used in various contexts, for example to compute the message digest when making a digital signature. A hash function compresses the bits of a message to a fixed-size *hash value* in a way that distributes the possible messages evenly among the possible hash values. A cryptographic hash function does this in a way that makes it extremely difficult to come up with a message that would hash to a particular hash value.

Cryptographic hash functions typically produce hash values of *128* or more bits. This number ($2^{128}$) is vastly larger than the number of different messages likely to ever be exchanged in the world. The reason for requiring more than *128* bits is based on the birthday paradox. The birthday paradox roughly states that given a hash function mapping any message to an *128*-bit hash digest, we can expect that the same digest will be computed twice when $2^{64}$ randomly selected messages have been hashed. As cheaper memory chips for computers become available it may become necessary to require larger than *128* bit message digests (such as *160* bits as has become standard recently).

Many good cryptographic hash functions are freely available. The most famous cryptographic hash functions are those of the MD family, in particular MD4 and MD5. MD4 has been broken, and MD5, although still in widespread use, should be considered insecure as well.

## 2.8 Encryption Methods

### 2.8.1 Symmetric (secret key)

Symmetric encryption means a *secret key* is shared by a peer. It is faster than the asymmetric methods and is hard to break if the key size is large. But it has some weaknesses:

- Key distribution: how to deliver the secret keys? It might be very unsafe
- Scalability: if a person has lots of person to talk to, he has to maintain a large key data set
- Limited security: no way to do authentication and no repudiation.

Often used symmetric algorithms include:

- DES (64 bits block, 64 bits key (56 bits - 8 bits parity), 16 rounds of transposition and substitution)
- 2DES(112 bit key, same work factor as DES)
- 3DES (168 bits key, 48 rounds, it takes 3 times longer than DES to encrypt and decrypt, $2^{56}$ times stronger than DES)
- AES (128, 192 or 256-bit key)
- Blowfish
- IDEA
- RC4, RC5, RC6

**DES, Double DES, 3DES**

DES originated from IBM, which was known as the Lucifer project, it became the data encryption standard in 1978 and was broken in 1998 in 3 days with a $250,000 computer. After that, the algorithm has been evolved to double-DES and 3DES. But the new versions are not admitted as standard, which is replaced by Rijndael algorithm, and is known as Advanced Encryption Standard (AES).

DES has four operation modes:

- Electronic Code Book (ECB) mode:
    - It is the native method for DES
    - It adds padding to neat and tidy 64-bit blocks
    - Code book provides the recipe of substitution and permutation
    - It doesn't require encrypt on order, the part after another part could be encrypted first
    - Not for large file, because it could reveal the encryption pattern, same plaintext--> same cipher text
    - Usually used for challenge-response operation and key management, PIN in ATM machine
- Cipher Block Chaining (CBC) mode
    - Not reveal pattern
    - The encryption of each block is dependent on all the blocks before it
    - It uses key and a value generated by previous blocks to calculate
- Cipher Feedback (CFB) mode
    - like CBC, but the previous cipher block is used to calculate the new cipher text
- Output Feedback (OFB) mode
    - Like CBC, but treat new block as stream

**AES**

- block cipher
- used to protect unclassified US government information

**IDEA** (International Data Encryption Algorithm)

- block cipher
- 64-bit block is divided into 16 sub-blocks, each with 8 rounds
- used in PGP

**Blowfish**

- 64 bit block, key length up to 448 bits, 16 rounds

**RC5**

- changeable block size and key size
- block size: 32, 64 or 128
- key size up to 2048

### 2.8.2 Asymmetric (public key)

It is the well known public key and private key method. Although it is slower than the symmetric method, but it does provide better key distribution security (confidentiality, authentication, no repudiation), and it is more scalable. It has three formats:

- ***Secure message format:*** Encrypted with receiver's public key, so only the receiver can decrypt it. It protect the confidentiality of a message, but not authentication.
- ***Open Message format:*** Encrypted with sender's private key, so anybody who has his public key can decrypt the message. So it provides authentication but no confidentiality
- ***Secure and signed format:*** It is a double encryption method which encrypts a message with the sender's private key at first then with the receiver's public key.

Some algorithms falls into this category are:

- RSA
- ECC
- Diffie-Hellman
- EL Gamal
- Digital Signature Standard (DSS)

**RSA**

- A pair of large prime numbers
- Used for encryption and digital signature
- Running in SSL in web browser
- PGP also uses it

**El Gamal**: Digital signature key exchange

**Elliptic Curve Cryptosystems (ECCs)**: Same functionality with RSA, more efficient

**Diffe-Hellman**

- It is the first algorithm came up with public key / private key concepts
- It is used only for key distribution, not encrypting message

## 2.9 What are the Advantages and Disadvantages of Public-Key Cryptography Compared with Secret-Key Cryptography?

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

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

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

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

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

Public-key cryptography is not meant to replace secret-key cryptography, but rather to supplement it, to make it more secure. The first use of public-key techniques was for secure key exchange in an otherwise secret-key system; 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.

## 2.10 Public Key Infrastructure (PKI)

Rather than being an encryption algorithm, PKI is a framework that uses public key cryptography and X.509 standard protocols.

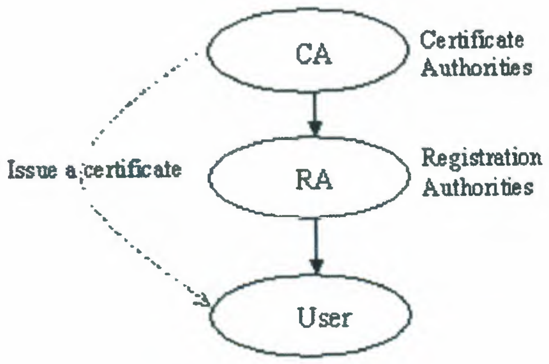


**Figure 2.2** X.509 Standard Protocols

Only a CA can issue certificate to user, a RA can hand out the certificate on behalf of a CA. Currently, most of the certificates are X.509 V3.

- Encrypt → confidentiality
- Hash → integrity
- Digital sign → integrity + authentication
- Encrypt + Digital sign → confidentiality + integrity + authentication

A certificate could be revoked under some circumstance. Revoked certificates are kept on the Certificate Revocation List (CRL). One user can have multiple keys under PKI for different levels strength.

**Some other terms**

- **One-way function:** a function computer easier in one way than its opposite direction. i.e., encryption is easier than decryption;
- **Trap door one-way function**: It is almost impossible to do the calculation in the opposite way unless you have a trapdoor, i.e., a private key

### 2.10.1 Message Integrity

Parity is used to deal with unintentional modification, such as disturbance in wire...Hash is used to protect message's integrity.

By hash algorithms, different message should produce different hash value; this is called collision free, repetitive free or resistant to birthday attack.

**One-way hash**: Takes a file and transfers it into a fixed-length value, aka, hash value, or message digest.

**Message Authentication Code (MAC)**: One-way hash value that is encrypted with a symmetric key. It is expected to be never performed in reverse.

One-time pad random number used only once, same length with message. It is impractical.

**Rules for key management;**

- Key should be long enough
- Stored and transmitted by secure means
- Extremely random and use full spectrum of the key space
- key lifetime is corresponded with message sensitivity
- The more a key is used, the shorter its life should be
- Backup or escrowed
- Destroyed after lifetime

# 3. MESSAGE AUTHENTICATION AND HASH FUNCTIONS

## 3.1 Overview

Perhaps the most confusing area of network security is that of message authentication and the related topics of digital signatures. The attacks and countermeasures become so convoluted that practitioners in this area begin to remind one of the astronomers of old, who built epicycles on top epicycles in an attempt to account for all contingencies. Fortunately, it appears that today's designers of cryptographic protocols, unlike those long-forgotten astronomers, are working from a fundamentally sound model.

It would be impossible, in anything less than book length, to exhaust all the cryptographic functions and protocols that have been proposed or implemented for message authentication and digital signatures. Instead, the purpose of this chapter and the next two is to provide a broad overview of the subject and to a develop a systematic means of describing the various approaches.

## 3.2 Authentication Requirements

In the context of communications across a network, the following attacks can be identified:

1. **Disclosure:** Release of message contents to any person or process not possessing the appropriate cryptographic key.
2. **Traffic analysis:** Discovery of the pattern of traffic between parties. In a connection-oriented application, the frequency and duration of connections could be determined. In either a connection-oriented or connectionless environment, the number and length of messages between parties could be determined,
3. **Masquerade**: Insertion of messages into the network from a fraudulent source. This includes the creation of messages by an opponent that are purported to come from an authorized entity. Also included are fraudulent acknowledgements of messages receipt or no receipt by someone other than the message recipient.

4. **Content modification:** Changes to the contents of a message, including insertion, deletion, transposition and modification.
5. **Sequence modification:** Any modification to a sequence of messages between parties, including insertion, deletion and reordering.
6. **Timing modification:** Delay or replay of messages. In a connection-oriented application, an entire session or sequence of messages could be a replay of some previous valid session, or individual messages in the sequence could be delayed or replayed. In a connectionless application, an individual message (e.g., datagram) could be delayed or replayed.
7. **Repudiation:** Denial of receipt of message by destination or denial of transmission of message by source.

Measured to deal with the first two attacks are in the realm of message confidentiality and are dealt with in Part One. Measures to deal with items 3 through 6 in the foregoing list are generally regarded as message authentication. Mechanisms for dealing specifically with item 7 come under the heading of digital signatures. Generally, a digital signature technique will also counter some or all the attacks listed under items 3 through 6.

## 3.3 Authentication Functions

Any message authentication or digital signature mechanism can be viewed as having fundamentally two levels. At the lower level, there must be some sort of function that produces an authenticator: a value to be used to authenticate a message. This lower level function is then used as primitive in a higher-level authentication protocol that enables a receiver to verify the authenticity of a message.

This section is concerned with the types of functions that may be used to produce an authenticator. These may be grouped into three classes, as follows:

- **Message encryption:** The cipher text of the entire message serves as its authenticator
- **Message authentication code (MAC):** A public function of the message and a secret key that produces a fixed-length value that serves as the authenticator

- **Hash function:** A public function that maps a message of any length into a fixed-length hash value, which serves as the authenticator

We now briefly examine each of these topics; MACs and hash functions are examined in greater detail in Section 3.3 and 3.4.

**Message Encryption**

Message encryption by itself can provide a measure of authentication. The analysis differs for conventional and public-key encryption schemes.

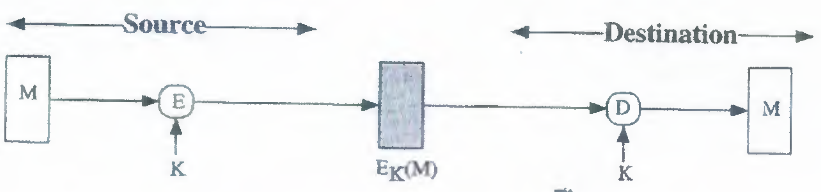
**Conventional Encryption**

Consider the straightforward use of conventional encryption (Figure 3.1a). A message transmitted from source A to destination B is encrypted using a secret key K shared by A and B. If no other party knows the key, then confidentiality is provided: No other party can recover the plaintext of the message.

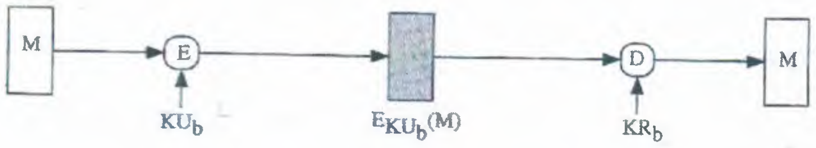
In addition, we may say that B is assured that the message came was generated by A. Why? The message must have come from A because A is the only other party that possesses K and therefore the only other party with the information is recovered, B knows that none of the bits of M have been altered, because an opponent that does not know K would not know how to alter bits in the cipher text to produce desired changes in the plaintext.

So we may say that conventional encryption provides authentication as well as confidentiality. However, this flat statement needs to be qualified. Consider exactly what is happening at B. Given a decryption function D and secret key K, the destination will accept any input X and produce output $Y = D_K(X)$. If X is the cipher text of a legitimate message M produced by the corresponding encryption function, then Y is some plaintext message M. Otherwise, Y will be meaningless sequence of bits. There may need to be some automated means of determining at B whether Y is legitimate plaintext and therefore must have come from A.

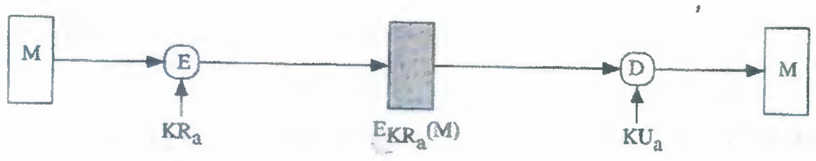
The implications of the line of reasoning in the preceding paragraph are profound from the point of view of authentication. Suppose the message M can be any arbitrary bit pattern. In that case, there is no way to determine automatically, at the destination, whether an incontrovertible: If M can be any bit pattern, then regardless of the value of X, Y = $D_K(X)$ is some bit pattern and therefore must be accepted as authentic plaintext.



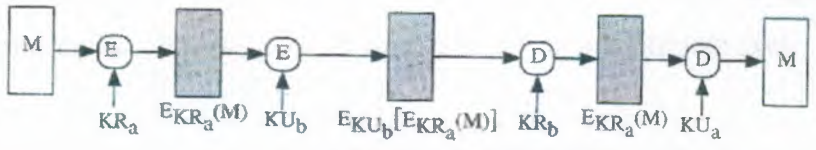
(a) Conventional encryption: confidentiality and authentication



(b) Public-key encryption: confidentiality



(c) Public-key encryption: authentication and signature



(d) Public-key encryption: confidentiality, authentication, and signature

**Figure 3.1** Basics Uses of Message Encryption

Thus, in general, we require that only a small subset of all possible bit patterns is considered legitimate plaintext. In that case, any spurious cipher text is unlikely to produce legitimate plaintext. For example, suppose that only one bit pattern in $10^6$ is legitimate plaintext. Then the probability that any randomly chosen bit pattern, treated as cipher text, will produce a legitimate plaintext message is only $10^{-6}$.

For a number of applications and encryption schemes, the desired conditions prevail as a matter of course. For example, suppose that we are transmitting English-language messages using a Ceaser cipher with a shift of one (K = 1). A sends the following legitimate cipher text:

```
nbsftfbupbutboeepftfbupbutboemjuumfmbnctfbujwz
```

B decrypts to produce the following plaintext:

```
mareseatoatsanddoeseatoatsandlittlelambseatitvy
```

A simple frequency analysis confirms that this message has the profile of ordinary English. On the other hand, if an opponent generates the following random sequence of letters:

```
zuvrsoevgqxlzwigamdvnmhpmccxiuureosfbcebtqxsxq
```

this decrypts to

```
ytuqrndufpwkyvhfzlcumlgolbbwhttqdnreabdaspwrwp
```

which does not fit the profile of ordinary English.

It may be difficult to determine automatically if incoming cipher text decrypts to intelligible plaintext. If the plaintext is, say, a binary object file or digitized X-rays, determination of properly formed and therefore authentic plaintext may be difficult. Thus, an opponent could achieve a certain level of disruption simply by issuing messages with random content purporting to come from a legitimate user.

One solution to this problem is to force the plaintext to have some structure that is easily recognized but that cannot be replicated without recourse to the encryption function. We could, for example, append an error-detecting code, also known as a frame check sequence (FCS) or checksum, to each message before encryption, as illustrated in Figure 3.2a. A prepares a plaintext message M and then provides this as input to a function F that produces an FCS. The FCS is appended to M and the entire block is then encrypted. At the destination, B decrypts the incoming block and treats the results as a message with an appended FCS. B applies the same function F to attempt to reproduce the FCS. If the calculated FCS is equal to the incoming FCS, then the message is considered authentic. It is unlikely that any random sequence of bits would exhibit the desired relationship.

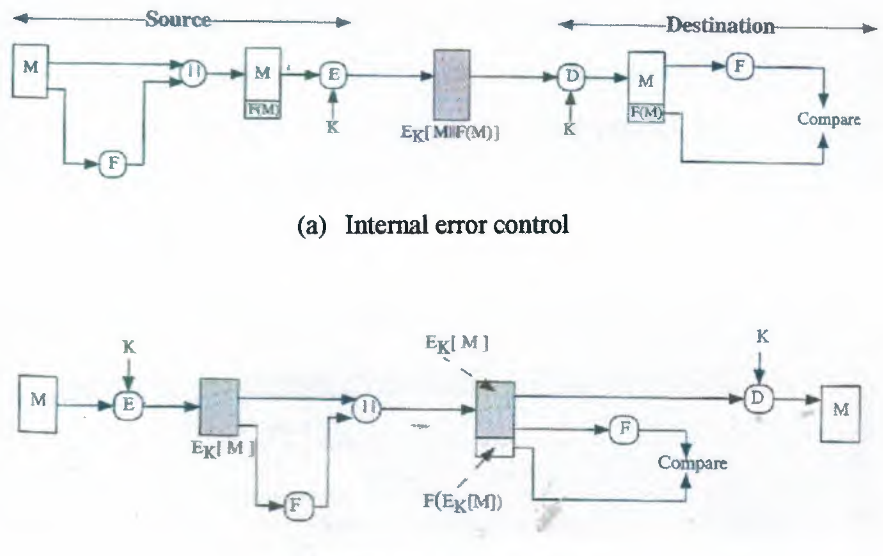


**Figure 3.2** Internal and External Error Control

Note that the order in which the FCS and encryption functions are performed is critical. The sequence illustrated in Figure 3.2a is referred to as internal error control, which the authors contrast with external error control (Figure 3.2b). With internal error control, authentication m-provided because an opponent would have difficulty generating cipher text that, when decrypted, would have valid error control bits. If instead the FCS is the outer code, an opponent can construct messages with valid error control codes. Although the opponent cannot know what the decrypted plaintext will be, he or she can still hope to create confusion and disrupt operations.

An error-control code is just one example; in fact, any sort of structuring added to the transmitted message serves to strengthen the authentication capability. Such structure is provided by the use of a communications architecture consisting of layered protocols. As an example, consider the structure of messages transmitted using the TCPI/IP protocol architecture. Figure 3.3 shows the format of a TCP segment, illustrating the TCP header. Now suppose that each pair of hosts shared a unique secret key, so that all exchanges between a pair of hosts used the same key, regardless of application. Then one could simply encrypt all of the data- gram except the IP header Again, if an opponent substituted some arbitrary bit pattern for the encrypted TCP segment, the resulting plaintext would not include a meaningful header. In this case, the header includes not only a check-sum (which covers the header) but other useful information, such as the sequence number. Because successive TCP segments on a given connection are numbered sequentially, encryption assures that an opponent does not delay, misorder, or delete any segments.

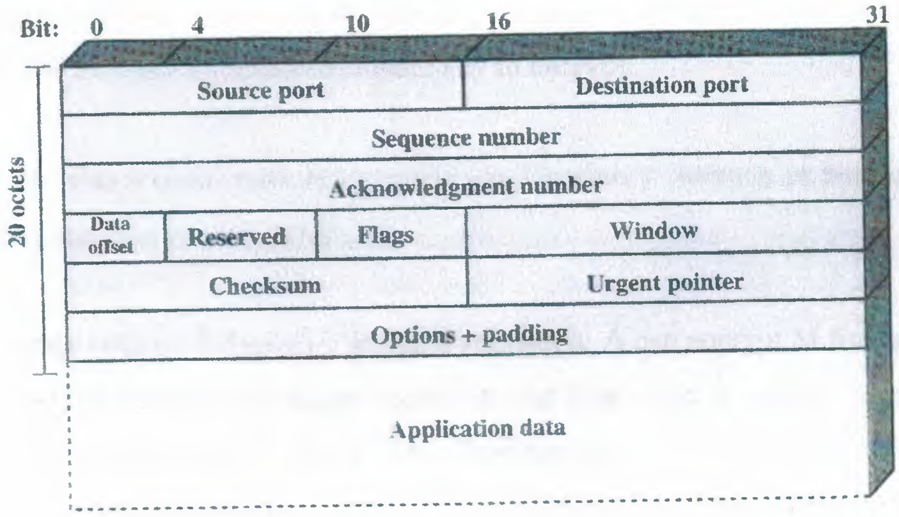


**Figure 3.3** TCP Segment

## Public-Key Encryption

The straightforward use of public-key encryption (Figure 3.1b) provides confidentiality but not authentication. The source (A) uses the public key $KU_b$ of the destination (B) to encrypt M. Because only B has the corresponding private key $KR_b$, only B can decrypt the message. This scheme provides no authentication because any opponent could also use B's public key to encrypt a message, claiming to be A.

To provide authentication, A uses its private key to encrypt the message, and B uses A's public key to decrypt (Figure 3.1c). This provides a measure of authentication using the same type of reasoning as in the conventional encryption case: The message must have come from A because A is the only party that possesses $KR_a$ and therefore the only party with the information necessary to construct cipher text that can be decrypted with $KU_a$. Again, the same reasoning as before applies: There must be some internal structure to the plaintext so that the receiver can distinguish between well formed plaintext and random bits.

Assuming there is such structure, then the scheme of Figure 3.1c does provide authentication. It also provides what is known as digital signature. Only A could have constructed the cipher text because only A possesses $KR_a$. Not even B, the recipient, could have constructed the cipher text. Therefore, if B is in possession of the cipher text, B has the means to prove that the message must have come from A. In effect, A has "signed" the message by using its private key to encrypt.

Note that this scheme does not provide confidentiality. Anyone in possession of A's public key can decrypt the cipher text.

To provide both confidentiality and authentication, A can encrypt M first using its private key, which provides the digital signature, and then using B's public key, which provides confidentiality (Figure 3.1d). The disadvantage of this approach is that the public-key algorithm, which is complex, must be exercised four times rather than two in each communication.

Table 3.1 summarizes the confidentiality and authentication implications of these various approaches to message encryption.

**Message Authentication Code**

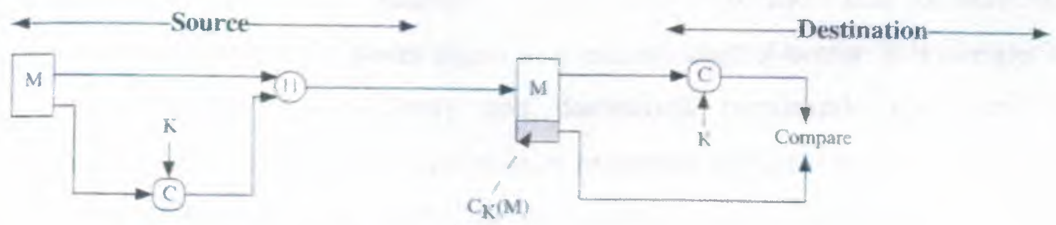
An alternative authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as a cryptographic checksum or MAC that is appended to the message. This technique assumes that two communicating parties, say A and B, share a common secret key K. When A has a message to send to B, it calculates the MAC as a function of the message and the key: MAC = $C_K(M)$. The message plus MAC are transmitted to the intended recipient. The recipient performs the same calculation on the received message, using the same secret key, to generate a new MAC. The received MAC is compared to the calculated MAC (Figure 3.4a). if we assume that only the receiver and the sender know the identity of the secret key, and if the received MAC matches the calculated MAC, then

**Table 3.1** Confidentiality and Authentication Implications of Messages Encryption

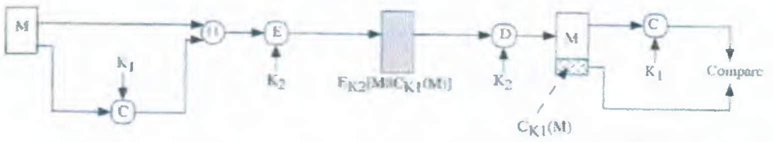
| **(a) Conventional (symmetric) Encryption** |
|---|
| A $\rightarrow$ B: $E_K[M]$<br>• Provides confidentiality<br>– Only A and B share K<br>• Provides a degree of authentication<br>– Could come only from A<br>– Has not been altered in transit<br>– Requires some formatting/redundancy<br>• Dose not provide signature<br>– Receiver could forge message<br>– Sender could deny message |
| **(b) Public-Key (asymmetric) Encryption** |
| A $\rightarrow$ B: $E_{KUb}[M]$<br>• Provides confidentiality<br>– Only B has $KR_b$ to decrypt<br>• Provides no authentication<br>– Any party could use $KU_b$ to encrypt message and claim to be A |
| A $\rightarrow$ B: $E_{KRa}[M]$<br>• Provides authentication and signature<br>– Only A has $KR_a$ to encrypt<br>– Has not been altered in transit<br>– Requires some formatting/redundancy<br>– Any party can use $KU_a$ to verify signature |
| A $\rightarrow$ B: $E_{KUb}[E_{KRa}(M)]$<br>• Provides confidentiality because of $KU_b$<br>• Provides authentication and signature because of $KR_a$ |

1. The receiver is assured that the message has not been altered. If an attacker alters the message but does not alter the MAC, then the receiver's calculation of the MAC will differ from the received MAC. Because the attacker is assumed not to know the secret key, the attacker cannot alter the MAC to correspond to the alterations in the message.
2. The receiver is assured that the message is from the alleged sender .Because no one else knows the secret key, no one else could prepare a message with a proper MAC.
3. If the message includes a sequence number, then the receiver can be assured of the proper sequence because an attacker cannot successfully alter the sequence number.

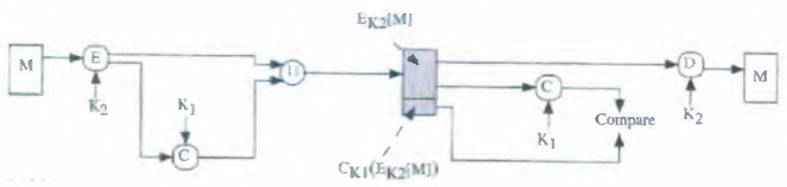
A MAC function is similar to encryption. One difference is that the MAC algorithm need not be reversible, as it must for decryption. It turns out that because of the mathematical properties of the authentication function, it is less vulnerable to being broken than encryption.



(a) Message authentication



(b) Message authentication and confidentiality; authentication tied to plaintext



(c) Message authentication and confidentiality; authentication tied to cipher text

**Figure 3.4** Basic Uses of Message Authentication Code (MAC)

The process just described provides authentication but not confidentiality, because the message as a whole is transmitted in the clear .Confidentiality can be provided by performing message encryption either after (Figure 3.4b) or before (Figure 3.4c) the MAC algorithm. In both these cases, two separate keys are needed, each of which is shared by the sender and the receiver. In the first case, the MAC is calculated with the message as input and is then concatenated to the message. The entire block is then encrypted. In the second case, the message is encrypted first. Then the MAC is calculated using the resulting cipher text and is concatenated to the cipher text to form the transmitted block. Typically, it is preferable to tie the authentication directly to the plaintext, so the method of Figure 3.4b is used.

Because conventional encryption will provide authentication and because it is widely used with readily available products, why not simply use this instead of a separate message authentication code? Three situations in which a message authentication code is used:

1. There are a number of applications which the same message is broadcast to a number of destinations. Examples are notification to users that the network is now unavailable or an alarm signal in a military control center. It is cheaper and more reliable to have only one destination responsible for monitoring authenticity. Thus, the message must be broadcast in plaintext with an associated message authentication code. The responsible system has the secret key and performs authentication. If a violation occurs, the other destination systems are alerted by a general alarm.
2. Another possible scenario is an exchange in which one side has a heavy load and cannot afford the time to decrypt all incoming messages. Authentication is carried out on a selective basis, messages being chosen at random for checking.
3. Authentication of a computer program in plaintext is an attractive service. The computer program can be executed without having to decrypt it every time; which would be wasteful of processor resources. However, if a message authentication code were attached to the program, it could be checked whenever assurance was required of the integrity of the program.

Three other rationales may be added, as follows:

4. For some applications, it may not be of concern to keep messages secret, but it is important to authenticate messages. An example is the Simple Network Management Protocol Version 3 (SNMPv3), which separates the functions of confidentiality and authentication. For this application, it is usually important for a managed system to authenticate incoming SNMP messages, particularly if the message contains a command to change parameters at the managed system. On the other hand, it may not be necessary to conceal the SNMP traffic.
5. Separation of authentication and confidentiality functions affords architectural flexibility. For example, it may be desired to perform authentication at the application level but to provide confidentiality at a lower level, such as the transport layer.
6. A user may wish to prolong the period of protection beyond the time of reception and yet allow processing of message contents. With message encryption, the protection is lost when the message is decrypted, so the message is protected against fraudulent modifications only in transit but not within the target system.

Finally, note that the MAC does not provide a digital signature because both sender and receiver share the same key.

Table 3.2 summarizes the confidentiality and authentication implications of the approaches illustrated in Figure 3.4.

A variation on the message authentication code is the one-way hash function. As with the message authentication code, a hash function accepts a variable-size message *M* as input and produces a fixed-size hash code H*(M)*, sometimes called a message digest, as output. The hash code is a function of all the bits of the message and provides an error-detection capability: A change to any bit or bits in the message results in a change to the hash code.

Figure 3.5 illustrates a variety of ways in which a hash code can be used to provide message authentication, as follows:

**Table 3.2** Basic Uses of Message Authentication Code C

| |
|---|
| (a) A$\rightarrow$ B: M \| \| $C_K$ (M)<br>• Provides authentication<br>– Only A and B share K |
| (b) A $\rightarrow$ B: $E_{K2}$ [M \| \| $C_{K1}$ (M)]<br>• Provides authentication<br>– Only A and B share $K_1$<br>• Provides confidentiality<br>– Only A and B share $K_2$ |
| (c) A $\rightarrow$ B: $E_{K2}$ [M] \| \| $C_{K1}$ ( $E_{K2}$[M] )<br>• Provides authentication<br>– Using $K_1$<br>• Provides confidentiality<br>– Using $K_2$ |

**a.** The message plus concatenated hash code is encrypted using conventional encryption. This is identical in structure to the internal error-control strategy shown in Figure 3.2a. The same line of reasoning applies: Because only A and B share the secret key, the message must have come from A and has not been altered. The hash code provides the structure or redundancy required to achieve authentication. Because encryption is applied to the entire message plus hash code, confidentiality is also provided.

**b.** Only the hash code is encrypted, using conventional encryption. This reduces the processing burden for those applications that do not require confidentiality. Note that the combination of hashing and encryption results in an overall function that is, in fact, a MAC (Figure 3.4a). That is, $E_K[H(M)]$ *is* a function of a variable-length message *M* and a secret key K, and it produces a fixed-size output that is secure against an opponent who does not know the secret key.

**c.** Only the hash code is encrypted, using public-key encryption and using the sender's private key. As with (b), this provides authentication. It also provides a digital signature, because only the sender could have produced the encrypted hash code. In fact, this is the essence of the digital signature technique.

**d.** If confidentiality as well as a digital signature is desired, then the message plus the public-key-encrypted hash code can be encrypted using a conventional secret key.

**e.** This technique uses a hash function but no encryption for message authentication. The technique assumes that the two communicating parties share a common secret value S. A computes the hash value over the concatenation of M and S and appends the resulting hash value to M. Because B possesses S, it can recomputed the hash value to verify. Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.

**f.** Confidentiality can be added to the approach of (e) by encrypting the entire message plus the hash code.

When confidentiality is not required, methods (b) and (c) have an advantage over those that encrypt the entire message in that less computation is required. Nevertheless, there has been growing interest in techniques that avoid encryption (Figure 3.5e). Several reasons for this interest are pointed out in:

- Encryption software is quite slow. Even though the amount of data to be encrypted per message is small, there may be a steady stream of messages into and out of a system.
- Encryption hardware costs are not negligible. Low-cost chip implementations of DES (Digital Encryption Standard) are available, but the cost adds up if all nodes in a network must have this capability.
- Encryption hardware is optimized toward large data sizes. For small blocks of data, a high proportion of the time is spent in initialization/invocation overhead.
- Encryption algorithms may be covered by patents. Some encryption algorithms, such as the RSA public-key algorithm, are patented and must be licensed, adding a cost.
- Encryption algorithms are subject to U .S. export control.

Table 3.3 summarizes the confidentiality and authentication implications of the approaches illustrated in Figure 3.5.

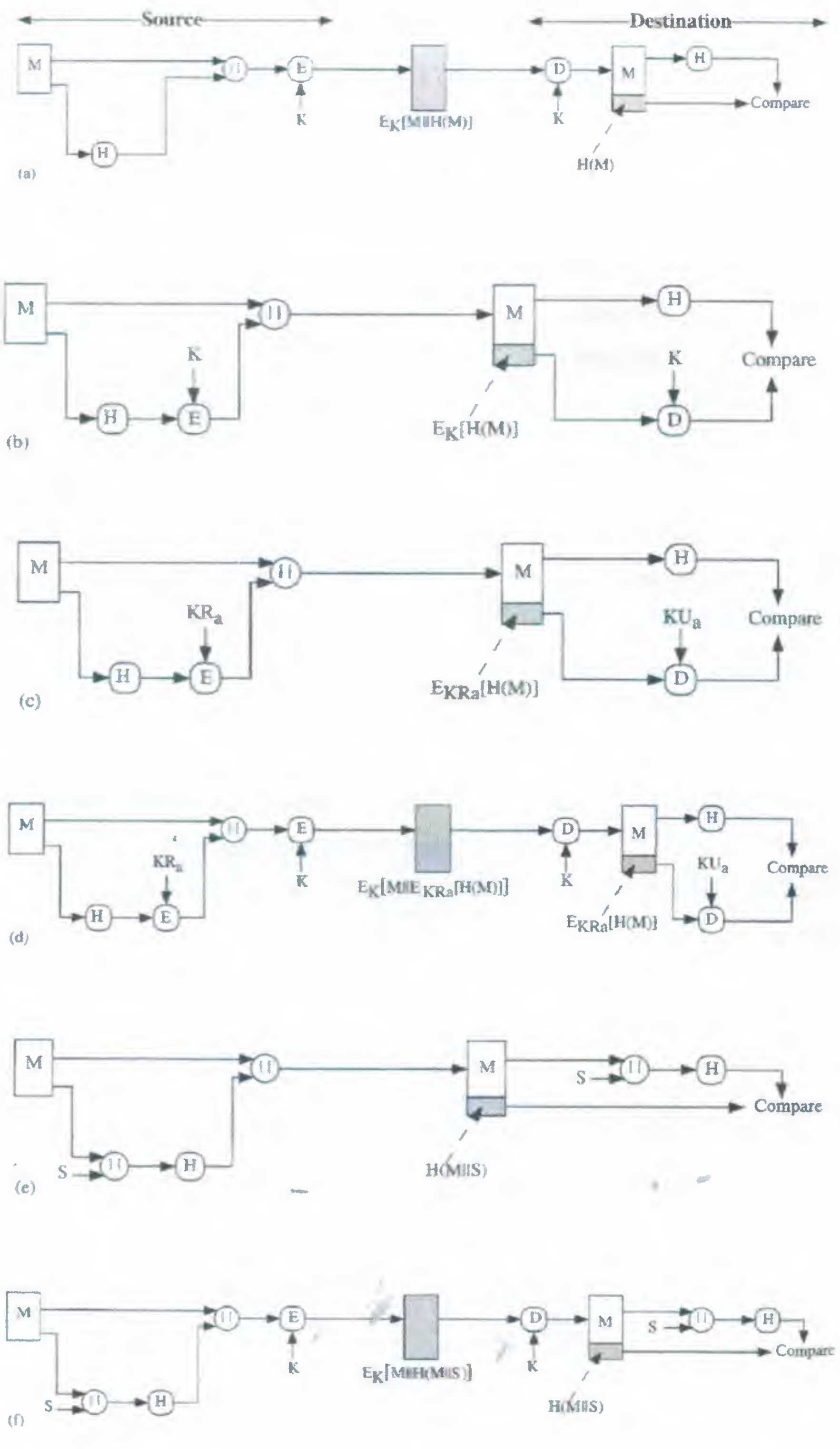


**Figure 3.5** Basic Uses of Hash Function

## 3.4 Message Authentication Codes

A MAC, also known as a cryptographic checksum, is generated by a function C of the form

$$MAC = C_K(M)$$

**Table 3.3** Basic Uses of Hash Function H

| | |
|---|---|
| (a) $A \rightarrow B: EK [ M \| H(M) ]$<br>• Provides confidentiality<br>— Only A and B share K<br>• Provides authentication<br>— H(M) is cryptographically protected | (d) $A \rightarrow B: E_K [ M \| E_{KRa} [ H(M) ] ]$<br>• Provides authentication and digital signature<br>• Provides confidentiality<br>— Only A and B share K |
| (b) $A \rightarrow B: M \| E_K [ H(M) ]$<br>• Provides authentication<br>— H(M) is cryptographically protected | (e) $A \rightarrow B: M \| E_K H( M \| S )$<br>• Provides authentication<br>— Only A and B share S |
| (c) $A \rightarrow B: M \| E_{KRa} [ H(M) ]$<br>• Provides authentication and digital signature<br>— H(M) is cryptographically protected<br>— Only A could create $E_{KRa}$ [H(M)] | (f) $A \rightarrow B: M \| E_K [ M \| H(M) \| S ]$<br>• Provides authentication<br>— Only A and B share S<br>• Provides confidentiality<br>— Only A and B share K |

Where M is a variable-length message, K is a secret key shared only by sender and receiver, and $C_K(M)$ is the fixed-length authenticator. The MAC is appended to the message at the source at a time when the message is assumed or known to be correct. The receiver authenticates that message by recomputing the MAC.

**Requirements for MACs**

When an entire message is encrypted for confidentiality, using either symmetric or asymmetric encryption, the security of the scheme generally depends on the bit length of the key. Barring some weakness in the algorithm, the opponent must resort to a brute-force attack using all possible keys. On average, such an attack will require $2^{(k-l)}$ attempts for a k-bit key .In particular, for a cipher text-only attack, the opponent, given cipher text C, would perform $P_i = D_{Ki}(C)$ for all possible key values $K_i$ until a $P_i$ was produced that matched the form of acceptable plaintext.

In the case of a MAC, the considerations are entirely different. In general, the MAC function is a many-to-one function. The domain of the function consists of messages of some arbitrary length, whereas the range consists of all possible MACs and all possible keys. If an n-bit MAC is used, then there are $2^n$ possible MACs, whereas there are N possible messages with $N >> 2^n$. Furthermore, with a k-bit key, there are 2k possible keys.

Using brute-force methods, how would an opponent attempt to discover a key? If confidentiality is not employed, the opponent has access to plaintext messages and their associated MACs. Suppose $k > n$; that is, suppose that the key size is greater than the MAC size, Then, given a known $M_1$ and $MAC_1$, with $MAC_1 = C_{K1}(M_1)$, the cryptanalyst can perform $MAC_i = C_{K\,i}(M_1)$for all possible key values $K_i$, At least one key is guaranteed to produce a match of $MAC_i = MAC_1$. Note that a total of $2^k$ MACs will be produced but there are only $2^n < 2^k$ different MAC values. Thus, a number of keys will produce the correct MAC, and the opponent has no way of knowing which the correct key. On average, a total of $2^k / 2^n = 2^{(k-n)}$ keys will produce a match. Thus, the opponent must iterate the attack:

- **Round 1**
  - Given: $M_1$, $MAC_1 = C_K(M_1)$
  - Compute $MAC_i = C_{Ki}(M_1)$ for all 2k keys
  - Number of matches $\approx 2^{(k-n)}$

- **Round 2**
    - Given: $M_2$, $MAC_2 = C_K(M_2)$
    - Compute $MAC_i = C_{Ki}(M_2)$ for the remaining $2^{(k-n)}$ keys
    - Number of matches $\approx 2^{(k-2 \times n)}$

and so on. On average, a rounds will be needed if $k = \alpha \times n$. For example, if an 80-bit key is used and the MAC is 32 bits long, then the first round will produce about $2^{48}$ possible keys. The second round will narrow the possible keys to about $2^{16}$ possibilities. The third round should produce only a single key, which must be the one used by the sender.

If the key length is less than or equal to the MAC length, then it is likely that a first round will produce a single match. It is possible that more than one key will produce such a match, in which case the opponent would need to perform the same test on a new (message, MAC) pair.

Thus, a brute-force attempt to discover the authentication key is no less effort and may be more effort than that required to discover a decryption key of the same length. However, other attacks that do not require the discovery of the key are possible.

**Message Authentication Code Based on DES**

One of the most widely used MACs, referred to as the Data Authentication Algorithm, and is based on DES. The algorithm is both a FIPS publication (FIPS PUB 113) and an ANSI standard (X9.17).

The algorithm can be defined as using the cipher block chaining (CBC) mode of operation of DES with an initialization vector of zero. The data (e.g., message, record, file, or program) to be authenticated is grouped into contiguous 64-bit blocks: $D_1, D_2, \ldots\ldots D_N$. If necessary, the final block is padded on the right with zeroes to form a full 64-bit block. Using the DES encryption algorithm, E, and a secret key, K, a data authentication code (DAC) is calculated as follows (Figure 3.6):

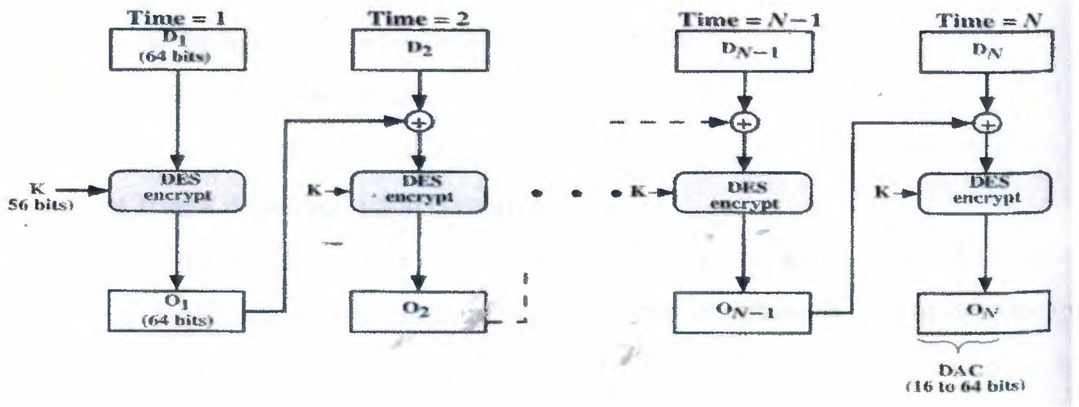


**Figure 3.6** Data Authentication Algorithm

The DAC consists of either the entire block $O_N$ or the leftmost M bits of the block, with $16 \leq M \leq 64$.

## 3.5 Hash Functions

A hash value is generated by a function h of the form

$$h = H(M)$$

where *M* is a variable-length message and *H(M)* is the fixed-length hash value. The hash value is appended to the message at the source at a time when the message is assumed or known to be correct. The receiver authenticates that message by recomputing the hash value. Because the hash function itself is not considered to be secret, some means is required to protect the hash value (Figure 3.5).

We begin by examining the requirements for a hash function to be used for message authentication. Because hash functions are, typically, quite complex, it is useful to examine next some very simple hash functions to get a feel for the issues involved. We then look at several approaches to hash function design.

### Requirements for a Hash Function

The purpose of a hash function is to produce a "fingerprint" of a file, message, or other block of data. To be useful for message authentication, a hash function H must have the following properties:

1. H can be applied to a block of data of any size.
2. H produces a fixed-length output.
3. H(x) is relatively easy to compute for any given x, making both hardware and software implementations practical.
4. For any given code h, it is computationally infeasible to find x such that H(x) = h. This is sometimes referred to in the literature as the one-way property.
5. For any given block x, it is computationally infeasible to find $y \neq x$ with H(y) = H(x). This is sometimes referred to as weak collision resistance.
6. It is computationally infeasible to find any pair (x, y) such that H(x) = H(Y). This is sometimes referred .to as strong collision resistance.

The first three properties are requirements for the practical application of a hash function to message authentication.

The fourth property is the one-way property: It is easy to generate a code given a message but virtually impossible to generate a message given a code. This property is important if the authentication technique involves the use of a secret value (Figure 3.5e). The secret value itself is not sent; however, if the hash function is not one way, an attacker can easily discover the secret value: If the attacker can observe or intercept a transmission, the attacker obtains the message *M* and the hash code $C = H(S_{AB} \| M)$. The attacker then inverts the hash function to obtain $S_{AB} \| M = H^{-1}(C)$. Because the attacker now has both *M* and $S_{AB} \| M$, it is a trivial matter to recover $S_{AB}$.

The fifth property guarantees that an alternative message hashing to the same value as a given message cannot be found. This prevents forgery when an encrypted hash code is used (Figure 3.5b and 3.5c). For these cases, the opponent can read the message and therefore generate its hash code. However, because the opponent does not have the secret key, the opponent should not be able to alter the message without detection. If this property were not true, an attacker would be capable of the following sequence: First, observe or intercept a message plus its encrypted hash code; second, generate an unencrypted hash code from the message; third, generate an alternate message with the same hash code.

The sixth property refers to how resistant the hash function is to a class of attack known as the birthday attack, which we examine shortly.

## 3.6 Security of Hash Functions and MACs

Just as with conventional and public-key encryption, we can group attacks on hash functions and MACs into two categories: brute-force attacks and cryptanalysis.

### Brute-Force Attacks

The nature of brute-force attacks differs somewhat for hash functions and MACS.

#### Hash Functions

The strength of a hash function against brute-force attacks depends solely on the length of the hash code produced by the algorithm. Recall from our discussion of hash functions that there are three desirable properties:

- **One-way:** For any given code *h*, it is computationally infeasible to find x such that H (x ) =h.
- **Weak collision resistance:** For any given block x, it is computationally infeasible to find $y \neq x$ with H(y) = H(x).
- **Strong collision resistance:** It is computationally infeasible to find any pair *(x,y)* such that H(x) = H(y).

For a code of length *n*, the level of effort required, as we have seen, is proportional to the following:

| | |
|---|---|
| One way | $2^n$ |
| Weak collision resistance | $2^n$ |
| Strong collision resistance | $2^{n/2}$ |

If strong collision resistance is required (and this is desirable for a general-purpose secure hash code), then the value $2^{n/2}$ determines the strength of the hash code against brute-force attacks. Oorschot and Wiener presented a design for a $10 million collision search machine for MD5, which has a 128-bit hash length that could find a collision in 24 days. Thus a 128-bit code may be viewed as inadequate. The next step up, if a hash code is treated as a sequence of 32 bits, is a 160-bit hash length. With a hash length of 160 bits, the same search machine would require four thousand years to find a collision.

**Message Authentication Codes**

A brute-force attack on a MAC is a more difficult undertaking because it requires known message-MAC pairs. Let us see why this is so. To attack a hash code, one can proceed in the following way. Given a fixed message x with n-bit hash code h = H(x), a brute-force method of finding a collision is to pick a random bit string y and check if H(y) = H(x). The attacker can do this repeatedly off line. Whether an off-line attack can be used on a MAC algorithm depends on the relative size of the key and the MAC.

To proceed, we need to state the desired security property of MAC algorithm, which can be expressed as follows:

- **Computation resistance:** Given one or more text-MAC pairs ( $x_i$ , $C_K( x_i )$), it is computationally infeasible to compute any text-MAC pair ( x , $C_K( x )$) for any new input $x \neq x_i$.

In other words, the attacker would like to come up with the valid MAC, code for a given message x. There are two lines of attack possible: Attack the key space and attack the MAC value. We examine each of these in turn.

If an attacker can determine the MAC key, then it is possible to generate a valid MAC value for any input x. suppose the key size is k bits and that the attacker has one known text-MAC pair. Then the attacker can compute the n-bit MAC on the known text for all possible keys. At least one key is guaranteed to produce the correct MAC — namely, the valid key that was initially used to produce the known text-MAC pair. This phase of the attack takes a level of effort proportional to $2^k$ (that is, one operation for each of the $2^k$ possible key values). However, as was described earlier, because the MAC is a many-to-one mapping, there may be other keys that produce the correct value. Thus, if more than one key is found to produce the correct value, additional text-MAC pairs must be tested. It can be shown that the level of effort drops off rapidly with each additional text-MAC pair and that the overall level of effort is roughly *2k*.

An attacker can also work on the MAC value without attempting to recover the key. Here, the objective is to generate a valid MAC value for a given message or to find a message that matches a given MAC value. In either case, the level of effort is comparable to that for attacking the one-way or weak collision resistant property of a hash code, or *2n*. In the case of the MAC, the attack cannot be conducted off line without further input; the attacker will require chosen text-MAC pairs or knowledge of the key.

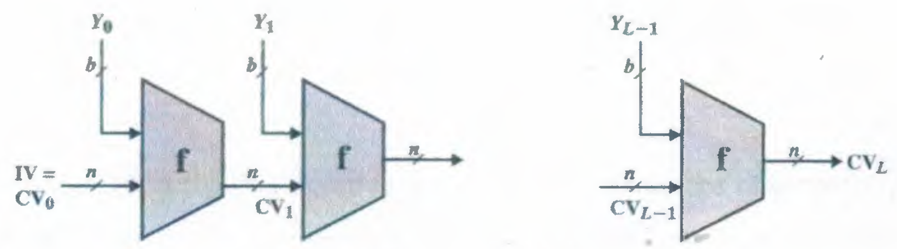
To summarize, the level of effort for brute-force attack on a MAC algorithm can be expressed as min(2k, 2n). The assessment of strength is similar to that for symmetric encryption algorithms. It would appear reasonable to require that the key length and MAC length satisfy a relationship such as min(k, n) ~ N, where N is perhaps in the range of 128 bits.

**Cryptanalysis**

As with encryption algorithms, cryptanalytic attacks on hash functions and MAC algorithms seek to exploit some property of the algorithm to perform some attack other than an exhaustive search. The way to measure the resistance of a hash or MAC algorithm to cryptanalysis is to compare its strength to the effort required for a brute force attack. That is, an ideal hash of MAC algorithm will require a cryptanalytic effort greater than or equal to the brute-force effort.

**Hash Function**

In recent years, there has been considerable effort, and some successes, in developing cryptanalytic attacks on hash functions. To understand these, we need to look at the overall structure of a typical secure hash function, indicated in Figure 3.10.



| | | | | | |
|---|---|---|---|---|---|
| **IV** | = | initial value | **CV** | = | chaining variable |
| **$Y_i$** | = | i th input block | **f** | = | compression algorithm |
| **L** | = | number of input blocks | | | |
| **n** | = | length of hash function | | | |
| **b** | = | length of input block | | | |

**Figure 3.10** General Structure of Secure Hash Function

This structure, referred to as an iterated hash function, was proposed by Merkle and is the structure of most hash functions in use today, including MDS, SHA-l, and RIPEMD-160. The hash function takes an input message and partitions it into L — 1 fixed-sized block of *b* bits each. If necessary, the final block is padded to *b* bits. The final block also includes the value of the .total length of the input to the hash function. The inclusion of the length makes the job of the opponent more difficult. Either the opponent must find two messages of equal length that hash to the same value or two messages of differing lengths that, together with their length values, hash to the same value.

The hash algorithm involves repeated use of a compression function, f that takes two inputs (an n-bit input from the previous step, called the chaining variable, and a b-bit block) and produces an n-bit output. At the start of hashing, the chaining variable has an initial value that is specified as part of the algorithm. The final value of the chaining variable is the hash value. Usually, b > n; hence the term compression. The hash function can be summarized as follows:

$$\begin{aligned} CV_0 &= IV = \text{initial n-bit value} \\ CV_1 &= f(CV_{i-1}, Y_{i-1}) \qquad 1 \leq i \leq L \\ H(M) &= CV_L \end{aligned}$$

where the input to the hash function is a message $M$ consisting of the blocks $Y_0, Y_1, \ldots, Y_{L-1}$.

The motivation for this iterative structure stems from the observation by Merkle and Damgard that if the compression functions is collision resistant, then so is the resultant iterated hash function. Therefore, the structure can be used produce a secure hash function to operate on a message of any length. The problem of designing a secure hash function reduces to that of designing a collision-resistant compression function that operates on inputs of some fixed size.

Cryptanalysis of hash functions focuses on the internal structure of f and is based on attempts to find efficient techniques for producing collisions for a single execution of f. Once that is done, the attack must take into account the fixed value of IV .The attack on f depends on exploiting its internal structure. Typically, as with symmetric block ciphers, f consists of a series of rounds of processing, so that the attack involves analysis of the pattern of bit changes from round to round.

Keep in mind that for any hash function there must exist collisions, because we are mapping a message of length at least equal to the block size $b$ into a hash code of length $n$, where $b < n$. What is required is that it is computationally infeasible to find collisions. The attacks that have been mounted on hash functions are rather complex and, beyond our scope here.

# 4. DIGITAL SIGNATURE AND AUTHENTICATION PROTOCOLS

## 4.1 Overview

As the Internet becomes more and more a part of our everyday lives and as online commerce continues to grow, Internet security becomes more of an issue. One of the great features of the Internet is also its greatest drawback, anonymity. You can pretend to be anyone that you want to be when you are in a chat room, posting messages to a message board, or even when sending email. The person reading your information on the other end does not need to know who you really are. However, when you want to purchase something online, whether it be a book or a stereo, being able to verify who you say you are is very important for the company that you are buying from.

The most important development from the work on public-key cryptography is the digital signature. The digital signature provides a set of security capabilities that would be difficult to implement in any other way.

The best and most common way to provide this sort of verification is via the concept of a digital signature. This signature works the same way as your real signature does. It is unique to you, and no one else has it.

## 4.2 Digital Signatures

**Requirements**

Message authentication protects two parties who exchange messages from any third party. However, it does not protect the two parties against each other. Several forms of dispute between the two are possible.

For example, suppose that John sends an authenticated message to Mary, using one of the schemes of Figure 3.4. Consider the following disputes that could arise:

1. Mary may forge a different message and claim that it came from John. Mary would simply have to create a message and append an authentication code using the key that John and Mary share.

2. John can deny sending the message. Because it is possible for Mary to forge a message, there is no way to prove that John did in fact send the message.

Both scenarios are of legitimate concern. Here is an example of the first scenario: An electronic funds transfer takes place, and the receiver increases the amount of funds transferred and claims that the larger amount had arrived from the sender. An example of the second scenario is that an electronic mail message contains instructions to a stockbroker for a transaction that subsequently turns out badly .The sender pretends that the message was never sent.

In situations where there is not complete trust between sender and receiver, something more than authentication is needed. The most attractive solution to this problem is the digital signature. The digital signature is analogous to the handwritten signature. It must have the following properties:

- It must be able to verify the author and the date and time of the signature.
- It must be able to authenticate the contents at the time of the signature.
- The signature must be verifiable by third parties, to resolve disputes.

Thus, the digital signature function includes the authentication function.

On the basis of these properties, we can formulate the following requirements for a digital signature:

- The signature must be a bit pattern that depends on the message being signed.
- The signature must use some information unique to the sender, to prevent both forgery and denial.
- It must be relatively easy to produce the digital signature.
- It must be relatively easy to recognize and verify the digital signature.
- It must be computationally infeasible to forge a digital signature, either by constructing a new message for an existing digital signature or by constructing a fraudulent digital signature for a given message.
- It must be practical to retain a copy of the digital signature in storage.

A secure hash function, embedded in a scheme such as that of Figure 3.5c or 3.5d, satisfies these requirements.

A variety of approaches has been proposed for the digital signature function. These approaches fall into two categories: direct and arbitrated.

**Direct Digital Signature**

The direct digital signature involves only the communicating parties (source, destination). It is assumed that the destination knows the public key of the source. A digital signature may be formed by encrypting the entire message with the sender's private key (Figure 3.1c) or by encrypting a hash code of the message with the sender's private key (Figure 3.5c).

Confidentiality can be provided by further encrypting the entire message plus signature with either the receiver's public key (public-key encryption) or a shared secret key (conventional encryption); for example, see Figures 3.1d and 3.5d. Note that it is important to perform the signature function first and then an outer confidentiality function. In case of dispute, some third party must view the message and its signature. If the signature is calculated on an encrypted message, then the third party also needs access to the decryption key to read the original message. However, if the signature is the inner operation, then the recipient can store the plaintext message and its signature for later use in dispute resolution.

All direct schemes described so far share a common weakness: The validity of the scheme depends on the security of the sender's private key. If a sender later wishes to deny sending a particular message, the sender can claim that the private key was lost or stolen and that someone else forged his or her signature. Administrative controls relating to the security of private keys can be employed to thwart or at least weaken this ploy, but the threat is still there, at least to some degree. One example is to require every signed message to include a timestamp (date and time) and to require prompt reporting of compromised keys to a central authority.

Another threat is that some private key might actually be stolen from X at time T. The opponent can then send a message signed with X's signature and stamped with a time before or equal to T.

**Arbitrated Digital Signature**

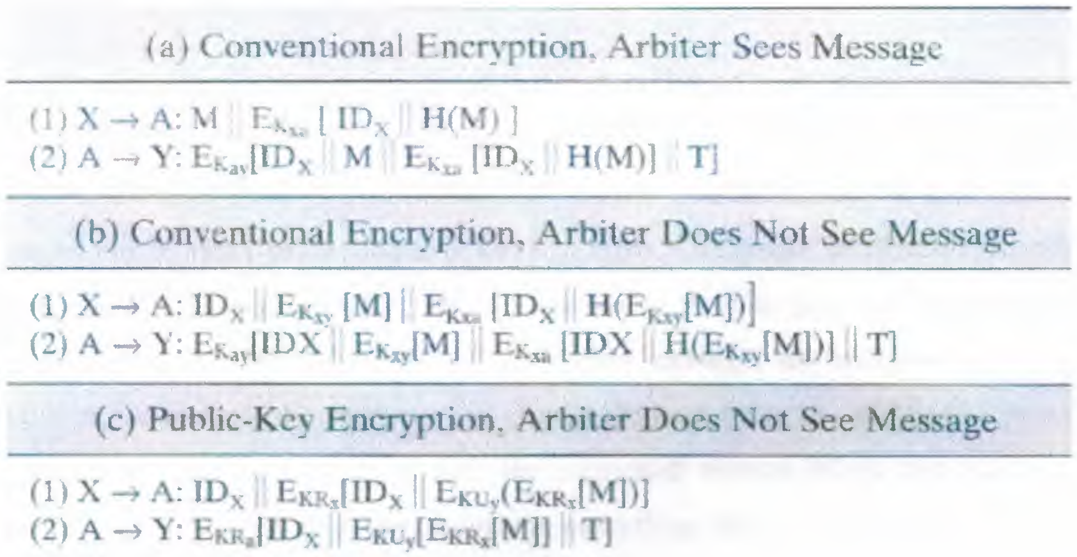
The problems associated with direct digital signatures can be addressed by using an arbiter.

As with direct signature schemes, there is a variety of arbitrated signature schemes. In general terms, they all operate as fol1ows: Every signed message from a sender X to a receiver Y goes first to an arbiter A, who subjects the message and its signature to a number of tests to check its origin and content. The message is then dated and sent to Y with an indication that it has been verified to the satisfaction of the arbiter. The presence of A solves the problem faced by direct signature schemes: that X might disown the message.

The arbiter plays a sensitive and crucial role in this sort of scheme, and all parties must have a great deal of trust that the arbitration mechanism is working properly.

Table 4.1, gives several examples of arbitrated digital signatures. In the first, conventional encryption is used. It is assumed that the sender X and the arbiter A share a secret key $K_{xa}$ and that A and Y share secret key $K_{ay}$. X constructs a message M and computes its hash value H(M). Then X transmits the message plus a signature to A. The signature consists of an identifier of X plus the hash value, all encrypted using $K_{xa}$. A decrypts the signature and checks the hash value to validate the message. Then A transmits a message to Y, encrypted with $K_{ay}$. The message includes $ID_X$, the original message

**Table 4.1** Arbitrated Digital Signature Techniques



| (a) Conventional Encryption, Arbiter Sees Message |
|---|
| (1) $X \rightarrow A$: $M \| E_{K_{xa}}[ID_X \| H(M)]$ |
| (2) $A \rightarrow Y$: $E_{K_{ay}}[ID_X \| M \| E_{K_{xa}}[ID_X \| H(M)] \| T]$ |
| **(b) Conventional Encryption, Arbiter Does Not See Message** |
| (1) $X \rightarrow A$: $ID_X \| E_{K_{xy}}[M] \| E_{K_{xa}}[ID_X \| H(E_{K_{xy}}[M])]$ |
| (2) $A \rightarrow Y$: $E_{K_{ay}}[IDX \| E_{K_{xy}}[M] \| E_{K_{xa}}[IDX \| H(E_{K_{xy}}[M])] \| T]$ |
| **(c) Public-Key Encryption, Arbiter Does Not See Message** |
| (1) $X \rightarrow A$: $ID_X \| E_{KR_x}[ID_X \| E_{KU_y}(E_{KR_x}[M])]$ |
| (2) $A \rightarrow Y$: $E_{KR_a}[ID_X \| E_{KU_y}[E_{KR_x}[M]] \| T]$ |

from X, the signature, and a timestamp. Y can decrypt this to recover the message and the signature. The timestamp informs Y that this message is timely and not a replay. Y can store M and the signature. In case of dispute, Y, who claims to have received M from X, sends the following message to A:

$$E_{Kay}[ID_X \| M \| E_{Kxa}[ID_X \| H(M)]]$$

The arbiter uses $K_{ay}$ to recover $ID_X$, M, and the signature, and then uses $K_{xa}$ to decrypt the signature and verify the hash code. In this scheme, Y cannot directly check X's signature; the signature is there solely to settle disputes. Y considers the message from X authentic because it comes through A. In this scenario, both sides must have a high degree of trust in A:

- X must trust A not to reveal $K_{xa}$ and not to generate false signatures of the form $E_{Kxa}[ID_x \| H(M)]$.
- Y must trust A to send $E_{Kay}[ID_X \| M \| E_{Kxa}[ID_X \| H(M)] \| T]$ only if the hash value is correct and the signature was generated by X.
- Both sides must trust A to resolve disputes fairly.

If the arbiter does live up to this trust, then X is assured that no one can forge his signature and Y is assured that X cannot disavow his signature.

The preceding scenario also implies that A is able to read messages from X to Y and, indeed, that any eavesdropper is able to do so. Table 10.1b shows a scenario that provides the arbitration as before but also assures confidentiality. In this case it is assumed that X and Y share the secret key $K_{xy}$. Now X transmits an identifier, a copy of the message encrypted with $K_{xy}$, and a signature to A. The signature consists of the identifier plus the hash value of the encrypted message, all encrypted using $K_{xa}$. As before, A decrypts the signature and checks the hash value to validate the message. In this case, A is working only with the encrypted version of the message and is prevented from reading it. A then transmits everything that it received from X, plus a timestamp, all encrypted with $K_{ay}$, to Y.

Although unable to read the message, the arbiter is still in a position to prevent fraud on the part of either X or Y. A remaining problem, one shared with the first scenario, is that the arbiter could form an alliance with the sender to deny a signed message, or with the receiver to forge the sender's signature.

All the problems just discussed can be resolved by going to a public-key scheme, one version of which is shown in Table 10.1c. In this case, X double encrypts a message M first with X's private key, $KR_X$, and then with Y's public key, $KU_y$. This is a signed, secret version of the message. This signed message, together with X's identifier, is encrypted again with $KR_x$ and, together with $ID_x$, is sent to A. The inner, double encrypted message is secure from the arbiter (and everyone else except Y). However, A can decrypt the outer encryption to assure that the message must have come from X (because only X has $KR_X$). A checks to make sure that X's private/public key pair is still valid and, if so, verifies the message. Then A transmits a message to Y, encrypted with $KR_a$. The message includes $ID_X$, the double-encrypted message, and a timestamp.

This scheme has a number of advantages over the preceding two schemes. First, no information is shared among the parties before communication, preventing alliances to defraud. Second, no incorrectly dated message can be sent, even if $KR_X$ is compromised, assuming that $KR_a$ is not compromised. Finally, the content of the message from X to Y is secret from A and anyone else.

## 4.3 Authentication Protocols

The basic tools described in Chapter 3 are used in a variety of applications, including the digital signature discussed in Section 4.1. Other uses are numerous and growing. In this section, we focus on two general areas (mutual authentication and one-way authentication) and examine some of the implications of authentication techniques in both.

**Mutual Authentication**

An important application area is that of mutual authentication protocols. Such protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys. There, the focus was key distribution. We return to this topic here to consider the wider implications of authentication.

Central to the problem of authenticated key exchange are two issues: confidentiality and timeliness. To prevent masquerade and to prevent compromise of session keys, essential identification and session key information must be communicated in encrypted form. This requires the prior existence of secret or public keys that can be used for this purpose. The second issue, timeliness, is important because of the threat of message replays. Such replays, at worst, could allow an opponent to compromise a session key or successfully impersonate another party. At minimum, a successful replay can disrupt operations by presenting parties with messages that appear genuine but are not.

The following examples of replay attacks:

- **Simple replay:** The opponent simply copies a message and replays it later.
- **Repetition that can be logged:** An opponent can replay a timestamp message within the valid time window.

- **Repetition that cannot be detected:** This situation could arise because the original message could have been suppressed and thus did not arrive at its destination; only the replay message arrives.
- **Backward replay without modification:** This is a replay back to the message sender. This attack is possible if conventional encryption is used and the sender cannot easily recognize the difference between messages sent and messages received on the basis of content.

One approach to coping with replay attacks is to attach a sequence number to each message used in an authentication exchange. A new message is accepted only if its sequence number is in the proper order. The difficulty with this approach is that it requires each party to keep track of the last sequence number for each claimant it has dealt with. Because of this overhead, sequence numbers are generally not used for authentication and key exchange. Instead, one of the following two genera1 approaches is used:

- **Timestamps:** Party A accepts a message as fresh only if the message contains a timestamp that, in A's judgment, is close enough to A's knowledge of current time. This approach requires that clocks among the various participants be synchronized.
- **Challenge/response:** Party A, expecting a fresh message from B, first sends B a nonce (challenge) and requires that the subsequent message (response) received from B contain the correct nonce value.

It can be argued that the timestamp approach should not be used for connection-oriented applications because of the inherent difficulties with this technique. First, some sort of protocol is needed to maintain synchronization among the various processor clocks. This protocol must be both fault tolerant, to cope with network errors, and secure, to cope with hostile attacks. Second, the opportunity for a successful attack will arise if there is a temporary loss of synchronization resulting from a fault in the clock mechanism of one of the parties. Finally, because of the variable and unpredictable nature of network delays, distributed clocks cannot be expected to maintain, precise synchronization. Therefore, any timestamp-based procedure must allow for a window of time sufficiently large to accommodate network delays yet sufficiently small to minimize the opportunity for attack.

On the other hand, the challenge-response approach is unsuitable for a connectionless type of application because it requires the overhead of a handshake before any connectionless transmission, effectively negating the chief characteristic of a connectionless transaction. For such applications, reliance on some sort of secure time server and a consistent attempt by each party to keep its clocks in synchronization may be the best approach.

**One-Way Authentication**

One application for which encryption is growing in popularity is electronic mail (e-mail). The very nature of electronic mail, and its chief benefit, is that it is not necessary for the sender and receiver to be on line at the same time. Instead, the e-mail message is forwarded to the receiver's electronic mailbox, where it is buffered until the receiver is available to read it.

The "envelope' or header of the e-mail message must be in the clear so that the message can be handled by the store-and-forward e-mail protocol, such as the Simple Mail Transfer Protocol (SMTP) or X.400. However, it often desirable that the mail-handling protocol not require access to the plaintext forms of the message, because that would require trusting the mail-handling mechanism. Accordingly, the e-mail message should be encrypted such that the mail-handling system is not in possession of the decryption key.

A second requirement is that of authentication. Typically, the recipient wants some assurance that the message is from the alleged sender.

**Public-Key Encryption Approaches**

We have already presented public-key encryption approaches that are suited to electronic mail, including the straightforward encryption of the entire message for confidentiality (Figure 4.1b), authentication (Figure 4.1c), or both (Figure 4.1d). These approaches require that either the sender know the recipient's public key (confidentiality) or that the recipient know the sender's public key (authentication) or both (confidentiality plus authentication). In addition, the public-key algorithm must be applied once or twice to what may be a long message.

If confidentiality is the primary concern, then the following may be more efficient:

$$A \rightarrow B \quad : \quad E_{KUb}[K_S] \,\|\, E_{KS}[M]$$

In this case, the message is encrypted with a one-time secret key. A also encrypts this one-time key with B's public key. Only B will be able to use the corresponding private key to recover the one-time key and then use that key to decrypt the message. This scheme is more efficient than simply encrypting the entire message with B's public key.

If authentication is the primary concern, then a digital signature may suffice, as was illustrated in Figure 3.5c:

$$A \rightarrow B \quad : \quad M \,\|\, E_{KRa}[H(M)]$$

This method guarantees that A cannot later deny having sent the message. However, this technique is open to another kind of fraud. Bob composes a message to his boss Alice that contains an idea that will save the company money. He appends his digital signature and sends it into the e-mail system. Eventually, the message will get delivered to Alice's mailbox. But suppose that Max has heard of Bob's idea and gains access to the mail queue before delivery. He finds Bob's message, strips off his signature, appends his, and requires the message to be delivered to Alice. Max gets credit for Bob's idea.

To counter such a scheme, both the message and signature can be encrypted with the recipient's public key:

$$A \rightarrow B \quad : \quad E_{KUb}[M \,\|\, E_{KRa}[H(M)]]$$

The latter two schemes require that B know A's public key and be convinced that it is timely. An effective way to provide this assurance is the digital certificate. Now we have

$$A \rightarrow B \quad : \quad M \| E_{KRa}[H(M)] \| E_{KRas}[T \| ID_A \| K_{Ua}]$$

In addition to the message, A sends B the signature, encrypted with A's private key, and A's certificate, encrypted with the private key of the authentication server .The recipient of the message first uses the certificate to obtain the sender's public key and verify that it is authentic and then uses the public key to verify the message itself. If confidentiality is required, then the entire message can be encrypted with B's public key. Alternatively, the entire message can be encrypted with a one-time secret key; the secret key is also transmitted, encrypted with B's public key.

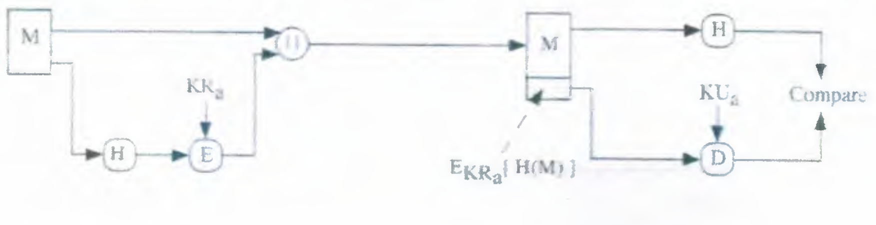
## 4.4 Digital Signature Standard

The National Institute of Standards and Technology (NIST) have published Federal Information Processing Standard FIPS PUB 186, known as the Digital Signature Standard (DSS). The DSS makes use of the Secure Hash Algorithm (SHA) and presents a new digital signature technique, the Digital Signature Algorithm (DSA). The DSS was originally proposed in 1991 and revised in 1993 in response to public feedback concerning the security of the scheme. A further minor revision occurred in 1996.

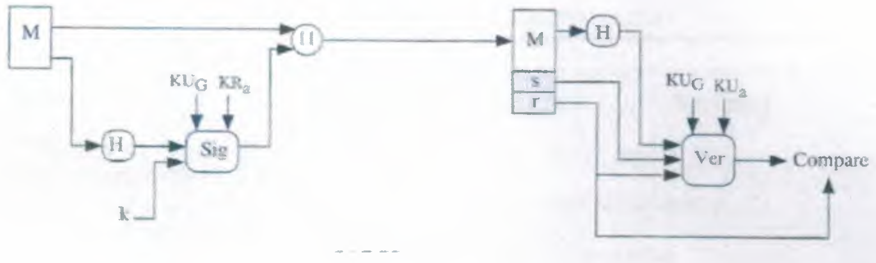
### The DSS Approach

The DSS uses an algorithm that is designed to provide only the digital signature function. Unlike RSA, it cannot be used for encryption or key exchange. Nevertheless, it is a public-key technique.

Figure 4.1 contrasts the DSS approach for generating digital signatures to that used with RSA. In the RSA approach, the message to be signed is input to a hash function that produces a secure hash code of fixed length. This hash code is then encrypted using the sender's private key to form the signature. Both the message and the signature are then transmitted. The recipient takes the message and produces a hash code. The recipient also decrypts the signature using the sender's public key. If the calculated hash code matches the decrypted signature, the signature is accepted as valid. Because only the sender knows the private key, only the sender could have produced a valid signature.



**(a) RSA Approach**



**(b) DSS Approach**

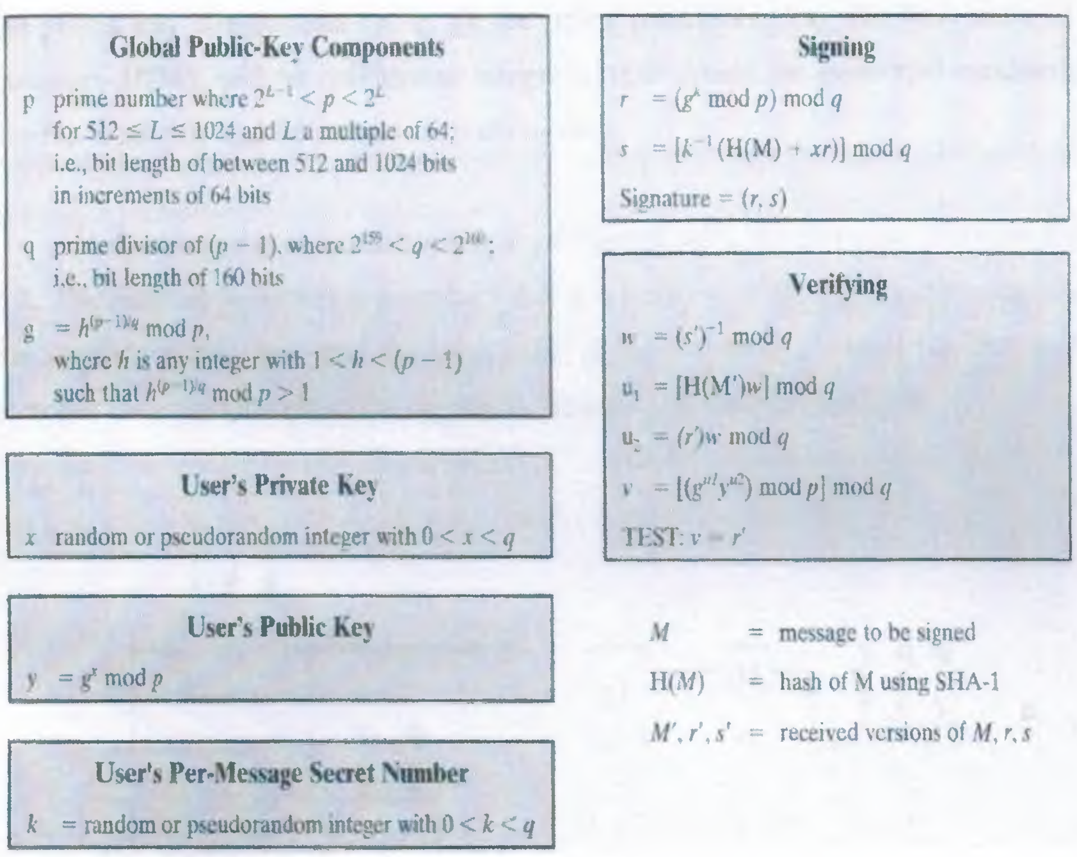
**Figure 4.1** Two Approaches to Digital Signatures

The DSS approach also makes use of a hash function. The hash code is provided as input to a signature function along with a random number *k* generated for this particular signature. The signature function also depends on the sender's private key ($KR_a$) and a set of parameters known to a group of communicating principals. We can consider this set to constitute a global public key ($KU_G$). The result is a signature is consisting of two components, labeled *s* and *r*.

At the receiving end, the hash code of the incoming message is generated. This plus the signature is input to a verification function. The verification function also depends on the global public key as well as the sender's public key ($KU_a$), which is paired with the sender's private key. The output of the verification function is a value that is equal to the signature component r if the signature is valid. The signature function is such that only the sender, with knowledge of the private key, could have produced the valid signature.

**The Digital Signature Algorithm**

The DSA is based on the difficulty of computing discrete logarithms and is based on schemes originally presented by ElGamal and Schnorr.



**Global Public-Key Components**

p prime number where $2^{L-1} < p < 2^L$ for $512 \le L \le 1024$ and $L$ a multiple of 64; i.e., bit length of between 512 and 1024 bits in increments of 64 bits

q prime divisor of $(p-1)$, where $2^{159} < q < 2^{160}$; i.e., bit length of 160 bits

g $= h^{(p-1)/q} \bmod p$, where $h$ is any integer with $1 < h < (p-1)$ such that $h^{(p-1)/q} \bmod p > 1$

**User's Private Key**

$x$ random or pseudorandom integer with $0 < x < q$

**User's Public Key**

$y = g^x \bmod p$

**User's Per-Message Secret Number**

$k$ = random or pseudorandom integer with $0 < k < q$

**Signing**

$r = (g^k \bmod p) \bmod q$

$s = [k^{-1}(H(M) + xr)] \bmod q$

Signature = $(r, s)$

**Verifying**

$w = (s')^{-1} \bmod q$

$u_1 = [H(M')w] \bmod q$

$u_2 = (r')w \bmod q$

$v = [(g^{u1} y^{u2}) \bmod p] \bmod q$

TEST: $v = r'$

| | | |
|---|---|---|
| $M$ | = | message to be signed |
| $H(M)$ | = | hash of M using SHA-1 |
| $M', r', s'$ | = | received versions of $M, r, s$ |

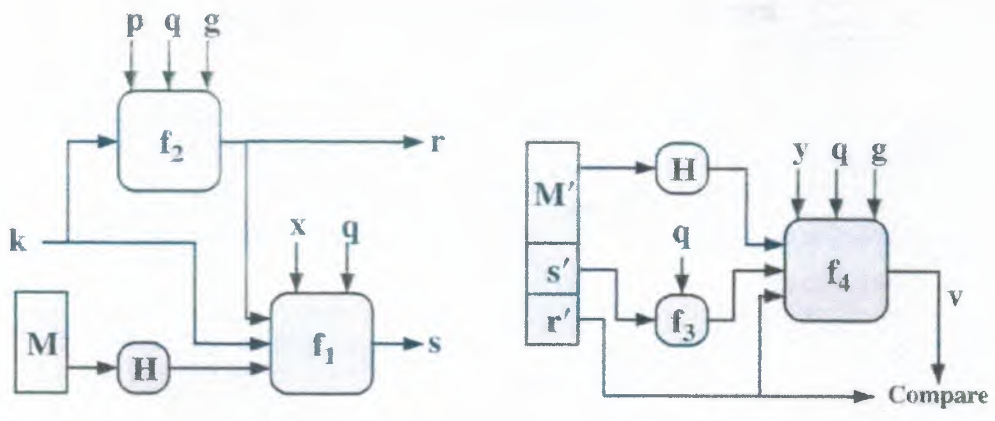
**Figure 4.2** The Digital Signature Algorithm (DSA)

Figure 4.2 summarizes the algorithm. There are three parameters that are public and can be common to a group of users. A 160-bit prime number q is chosen. Next, a prime number p is selected with a length between 512 and 1024 bits such that q divides (p-1). Finally, g is chosen to be of the form h $^{(p-1)/q}$ mod p, where h is an integer between 1 and (p-1) with the restriction that g must be greater than 1.

With these numbers in hand, each user selects a private key and generates a public key. The private key x must be a number from 1 to (q-1) and should be chosen randomly or pseudo randomly. The public key is calculated from the private key as $y = g^x \bmod p$. The calculation of y given x is relatively straightforward. However, given the public key y, it is believed to be computationally infeasible to determine x, which is the discrete logarithm of y to the base g, mod p.

To create a signature, a user calculates two quantities, r and s, that are functions of the public key components (p, q, g), the user's private key (x), the hash code of the message, H(M), and an additional integer k that should be generated randomly or pseudo randomly and be unique for each signing.

At the receiving end, verification is performed using the formulas shown in Figure 4.2. The receiver generates a quantity v that is a function of the public-key components, the sender's public key, and the hash code of the incoming message. If this quantity matches the r component of the signature, then the signature is validated.



$s = f_1(H(M)), k, x, r, q) = (k^{-1}(H(M) + xr)) \bmod q$

$r = f_2(k,p,q,g) = (g^k \bmod p) \bmod q$

$w = f_3(s', q) = (s')^{-1} \bmod q$

$v = f_4(y, q, g, H(M)'), w, r')$

**(a) Signing** **(b) Verifying**

**Figure 4.3** DSS Signing and Verifying

The structure of the algorithm, as revealed in Figure 4.3, is quite interesting. Note that the test at the end is on the value *r*, which does not depend on the message at all. Instead, *r* is a function of *k* and the three global public-key components. The multiplicative inverse of *k* (mod *p*) is passed to a function that also has as inputs the message hash code and the user's private key. The structure of this function is such that the receiver can recover *r* using the incoming message and signature, the public key of the user, and the global public key. It is certainly not obvious from Figure 4.2 or Figure 4.3 that such a scheme would work.

Given the difficulty of taking discrete logarithms, it is infeasible for an opponent to recover *k* from *r* or to recover x from s.

Another point worth noting is that the only computationally demanding task in signature generation is the exponential calculation $g^k$ mod *p*. Because this value does not depend on the message to be signed, it can be computed ahead of time. Indeed, a user could precalculate a number of values of *r* to be used to sign documents as needed. The only other somewhat demanding task is the determination of a multiplicative inverse, $k^{-1}$. Again, a number of these values can be precalculated.

# CONCLUSION

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 and a private signing function (encrypting with the signer's private key), but there are other methods.

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.

## REFERENCES

[1] Stalling, Williams, "Cryptography and Network Security Principles and Practice", ISBN: 0-13-869017-0, Prentice-Hall Inc., 1999

[2] Trappe, Wade & Washington, C., Lawrence, "Introduction to Cryptography with coding theory", ISNB: 0-13-061814-4, Prentice-Hall Inc., 2002

[3] Bernstein, Karn, *and* Junger, Constitutional Challenges to Cryptographic Regulations, *"http://www.law.ua.edu/lawreview/crain.htm "*

[4] SSH Communications Security, Cryptography A-Z, *"http://www.ssh.com/support/cryptography/index.html"*

[5] The Security Portal for Information System Security Professionals, Cryptographic Algorithms, *"http://www.ghostship.com/infosyssec/cryptalgorithms.html"*

## APPENDIX A

Comparison of Asymmetric, Symmetric and Hash Algorithm Methods

| | | Encryption | Digital Signature | Hashing | Key Distribution |
|---|---|---|---|---|---|
| **Asymmetric** | RSA | X | X | | X |
| | ECC | X | X | | X |
| | Diffie-Hellman | | | | X |
| | El Gamal | | X | | X |
| **Symmetric** | DES | X | | | |
| | 3DES | X | | | |
| | Blowfish | X | | | |
| | IDEA | X | | | |
| | RC4 | X | | | |
| | SAFER | X | | | |
| **Hash** | RSA message digest used within RSA operation | | | X | |
| | Ronald Rivest family of hashing function MD2, MD4, MD5 | | | X | |
| | Secure Hash Algorithm (SHA) used with Digital Signature Algorithm (DSA) | | X | X | |
| | HAVAL (variable length hash values using a one-way function design) | | | X | |