CRYPTOGRAPHY AND NETWORK SECURITY

(DMCA302)

(MCA)

ACHARYA NAGARJUNA UNIVERSITY

CENTRE FOR DISTANCE EDUCATION

NAGARJUNA NAGAR,

GUNTUR

ANDHRA PRADESH

CONTENTS

UNIT/LESSON	PARTICULARS	PAGE NO
UNIT – I		**1 – 81**
LESSON 1	CRYPTOGRAPHY	2 – 13
LESSON 2	CONVENTIONAL ENCRYPTION TECHNIQUES	14 - 35
LESSON 3	BLOCK CIPHERS AND DES	36 - 61
LESSON 4	ADVANCED ENCRYTION STANDARD	62 – 81
UNIT – II		**82 – 125**
LESSON 1	NUMBER THEORY	83 - 92
LESSON 2	DISCRETE LOGARITHMS AND ALGORITHM COMPLEXITY	93 – 99
LESSON 3	PUBLIC KEY CRYPTOSYSTEMS	100 – 108
LESSON 4	RSA AND DEFFIE-HELLMAN KEY EXCHANGE ALGORITHMS	109 – 125
UNIT – III		**126 – 183**
LESSON 1	ELLIPTICAL CURVE CRYPTOGRAPHY	127 – 136
LESSON 2	MESSAGE AUTHENTICATION AND HASH FUNCTIONS	137 – 158
LESSON 3	DIGITAL SIGNATURES	159 – 175
LESSON 4	KEY MANAGEMENT SCHEMES	176 - 183
UNIT – IV		**184 – 328**
LESSON 1	KERBEROS	185 – 203
LESSON 2	DIRECTORY AUTHENTICATION SERVICES	204 – 215
LESSON 3	ELECTRONIC SECURITY	216 - 251
LESSON 4	WEB SECURITY	252 – 278
LESSON 5	SYSTEM SECURITY	279 – 315
LESSON 6	FIREWALLS	316 - 328

UNIT- I

1. CRYPTOGRAPHY

2. CONVENTIONAL ENCRYPTION TECHNIQUES

3. BLOCK CIPHERS AND DES

4. ADVANCED ENCRYPTION STANDARD

1. CRYTOGRAPHY

OBJECTIVE

This chapter describes classical symmetric encryption techniques. It provides a gentle and interesting introduction to cryptography and cryptanalysis and highlights important concepts.

CRYTOGRAPHY

The term C**ryptology** is derived from the Greek *kryptós* (“hidden”) and *lógos* (“word”). Cryptology is the science of coding and decoding secret messages. It is usually divided into two parts

- **Cryptography**, which concerns designing cryptosystems for coding and decoding messages, and the more glamorous
- **Cryptanalysis**, which is concerned with ``breaking'' cryptosystems, or deciphering messages without prior detailed knowledge of the cryptosystem.

Cryptography (from the Greek *kryptós* and *gráphein*, “to write”) was originally the study of the principles and techniques by which information could be concealed in ciphers and later revealed by legitimate users employing the secret key. It now encompasses the whole area of key-controlled transformations of information into forms that are either impossible or computationally infeasible for unauthorized persons to duplicate or undo.

Cryptanalysis (from the Greek *kryptós* and *analýein*, “to loosen” or “to untie”) is the science (and art) of recovering or forging cryptographically secured information without knowledge of the key. Cryptology is often—and mistakenly—considered a synonym for cryptography and occasionally for cryptanalysis, but specialists in the field have for years adopted the convention that cryptology is the more inclusive term, encompassing both cryptography and cryptanalysis.

Cryptography was initially only concerned with providing secrecy for written messages, especially in times of war. Figure 1.1.1 shows the German Lorenz cipher machine used for encryption during the World War II. Its principles apply equally well, however, to securing data flowing between computers or data stored in them, to encrypting facsimile and television signals, to verifying the identity of participants in electronic commerce (e-commerce) and providing legally acceptable records of

those transactions. Because of this broadened interpretation of cryptography, the field of cryptanalysis has also been enlarged.

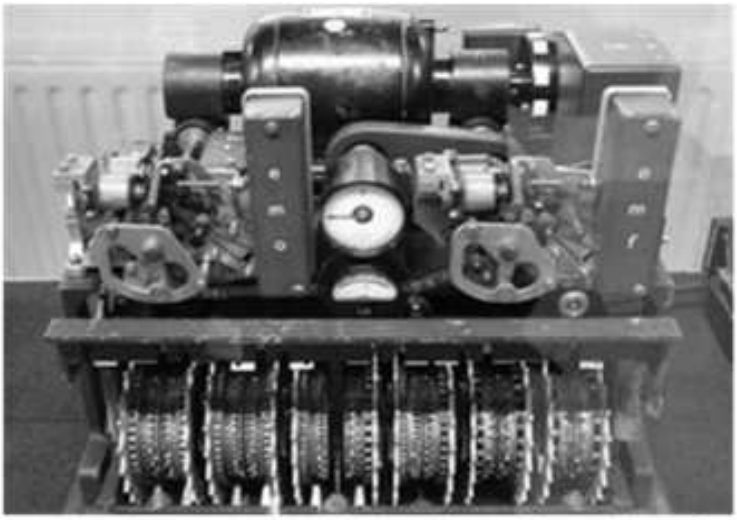


Fig1.1.1 The German Lorenz cipher machine, used in World War II for encryption of very high-level general staff messages

Attacks, Services and Mechanisms

To effectively identify the information security needs of an organization and to evaluate various security products and policies, we consider three aspects:

- **Security Attack:** Any action that compromises the security of information owned by an organization (Interruption, Interception, Modification, Fabrication).
- **Security Mechanism:** A mechanism that is designed to detect, prevent, or recover from a security attack (Encryption, Digital Signature, SSL etc.).
- **Security Service:** A service that enhances the security of the data processing systems and the information transfers of an organization. The services are intended to counter security attacks, and they make use of one or more security mechanisms to provide the service.

Security Attacks

Let us consider an information source and information destination as shown in fig 1.1.2. The normal flow is depicted in fig 1.1.2a. The following are the four general categories of attacks.

- **Interruption:** This is an *attack on availability*. An asset of the system is destroyed or becomes unavailable or unusable. This is depicted in fig 1.1.2b.

  Examples: Breaking communication lines, destructing hardware such as hard disks etc.
- **Interception:** This is *attack on confidentiality*. An unauthorized party gains access to an asset. The unauthorized party could be a person, a program, or a computer. This is depicted in fig 1.1.2c.

  Examples: wire tapping to access data in a network, unauthorized copying of files, programs etc.
- **Modification:** This is *attack on integrity*. An unauthorized party not only gains access but tampers with an asset. This is depicted in fig 1.1.2d.

  Examples: changing values in a data file, altering a program so that it functions differently and modifying content of messages being transmitted in a network.
- **Fabrication:** This is *attack on Authenticity*. An unauthorized party inserts counterfeit objects into the system. This is depicted in fig. 1.1.2e.

  Examples: Adding records to a file

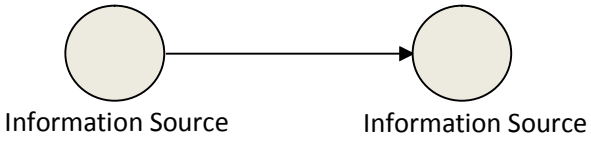


Fig. 1.1.2a Normal

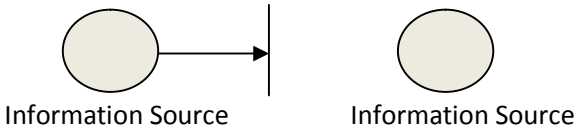


Fig 1.1.2b Interruption

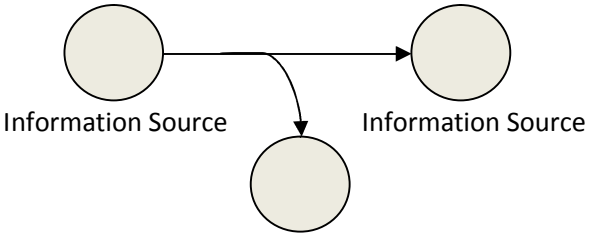


Fig 1.1.2c Interception

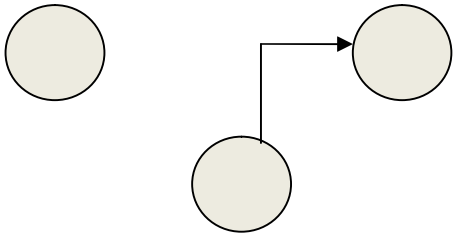


Fig.1.1.2e Fabrication

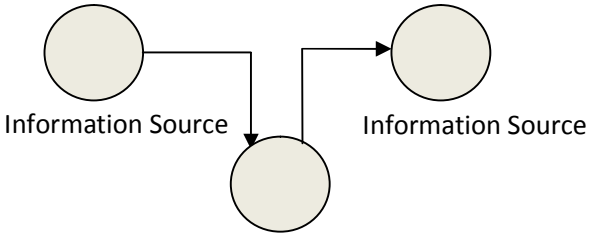


Fig. 1.1.2d Modification

Fig 1.1.2 Security Attacks

Security Attacks can also be characterized as

- Passive Attacks
- Active Attacks

Passive Attacks

Passive attack is an attack where an unauthorized attacker monitors or listens in on the communication between two parties. Figure 1.1.3 illustrates a passive attack where Eve monitors the communication

between Alice and Bob. The goal of the opponent is to obtain information that is being transmitted. Two types of passive attacks are

- Release of Message Contents
- Traffic Analysis

Release of Messages Contents: A telephone conversation, an Electronic mail message and a transferred file may contain sensitive or confidential information. We would like to prevent the opponent from learning the contents of these transmissions.

Traffic Analysis: Traffic analysis is the process of intercepting and examining messages in order to deduce information from patterns in communication. It can be performed even when the messages are encrypted and cannot be decrypted. In general, the greater the number of messages observed, or even intercepted and stored, the more can be inferred from the traffic. This information might be useful in guessing the nature of the communication that was taking place. The opponent could determine the location and identity of communicating hosts.

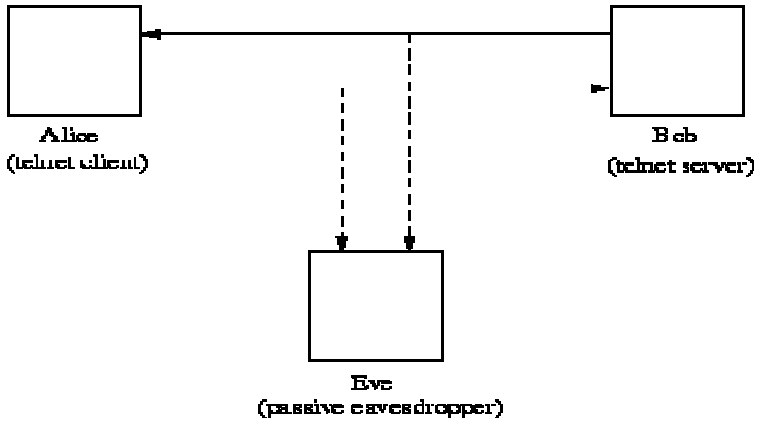


Fig 1.1.3 Illustration of Passive Attack

Since they do not involve any alteration of the data, the passive attacks are very difficult to detect. It is feasible to prevent them. Therefore, they are hard to detect and easy to prevent.

Active Attacks:

These attacks involve some modification of the data stream or creation of a false stream. Active attacks are subdivided into four categories.

- Masquerade
- Replay
- Modification of messages
- Denial of Service

Masquerade

Masquerade is a type of attack where the attacker pretends to be an authorized user of a system in order to gain access to it or to gain greater privileges than they are authorized for.

A masquerade may be attempted through the use of stolen logon IDs and passwords, through finding security gaps in programs, or through bypassing the authentication mechanism. The attempt may come from within an organization, for example, from an employee; or from an outside user through some connection to the public network. Weak authentication provides one of the easiest points of entry for a masquerade, since it makes it much easier for an attacker to gain access. Once the attacker has been authorized for entry, they may have full access to the organization's critical data, and (depending on the privilege level they pretend to have) may be able to modify and delete software and data, and make changes to network configuration and routing information.

Replay

A replay attack is a form of network attack in which a valid data transmission is maliciously or fraudulently repeated or delayed. This is carried out either by the originator or by an adversary who intercepts the data and retransmits it, possibly as part of a masquerade attack by IP packet substitution (such as stream cipher attack).

As an example, suppose A wants to prove its identity to B. B requests A's password as proof of identity, which A dutifully provides (possibly after some transformation like a hash function); meanwhile, some person X is eavesdropping the conversation and keeps the password. After the interchange is over, X connects to B posing as A; when asked for a proof of identity, X sends A's password read from the last session, which B must accept.

Modification of messages

Modification of messages means that some portion of a legitimate message is altered, or that messages are delayed or reordered, to produce an unauthorized effect. For example, a message meaning "Allow John Smith to read confidential file accounts" is modified to mean "Allow Fred Brown to read confidential file accounts."

Denial of Service

The denial of service prevents or inhibits the normal use or management of communications facilities. This attack may have a specific target; for example, an entity may suppress all messages directed to a particular destination (e.g., the security audit service). Another form of service denial is the disruption of an entire network, either by disabling the

network or by overloading it with messages so as to degrade performance.

Security Services

Security services are the base-level services that are used to combat the security attacks. The services defined here should not be confused with security mechanisms, which are the actual implementations of these services. These services include

- Confidentiality
- Authentication
- Integrity
- Nonrepudiation
- Access Control
- Availability

Confidentiality

The confidentiality service provides for the secrecy of information. When properly used, confidentiality only allows authorized users to have access to information. In order to perform this service properly, the confidentiality service must work with the accountability service to properly identify individuals. In performing this function, the confidentiality service protects against the access attack. The confidentiality service must take into account the fact that information may reside in physical form in paper files, in electronic form in electronic files, and in transit. For example, a credit card transaction on the Internet requires the credit card number to be transmitted from the buyer to the merchant and from the merchant to a transaction processing network. The system attempts to enforce confidentiality by encrypting the card number during transmission, by limiting the places where it might appear (in databases, log files, backups, printed receipts, and so on), and by restricting access to the places where it is stored. If an unauthorized party obtains the card number in any way, a breach of confidentiality has occurred.

Breaches of confidentiality take many forms. Permitting someone to look over your shoulder at your computer screen while you have confidential data displayed on it could be a breach of confidentiality. If a laptop computer containing sensitive information about a company's employees is stolen or sold, it could result in a breach of confidentiality. Giving out confidential information over the telephone is a breach of confidentiality if the caller is not authorized to have the information.

Confidentiality is necessary (but not sufficient) for maintaining the privacy of the people whose personal information a system holds.

Authentication

Authentication is the act of establishing or confirming something (or someone) as authentic, i.e. that claims made by or about the thing are true. This might involve confirming the identity of a person, the origins of an artifact, or assuring that a computer program is a trusted one.

In private and public computer networks (including the Internet), authentication is commonly done through the use of logon passwords. Knowledge of the password is assumed to guarantee that the user is authentic. Each user registers initially (or is registered by someone else), using an assigned or self-declared password. On each subsequent use, the user must know and use the previously declared password. The weakness in this system for transactions that are significant (such as the exchange of money) is that passwords can often be stolen, accidentally revealed, or forgotten. For this reason, Internet business and many other transactions require a more stringent authentication process. The use of digital certificates issued and verified by a Certificate Authority (CA) as part of a public key infrastructure is considered likely to become the standard way to perform authentication on the Internet.

Integrity

Integrity, in terms of data and network security, is the assurance that information can only be accessed or modified by those authorized to do so.

A connection-oriented integrity service deals with a stream of messages, assures that messages are received as sent, with no duplication, insertion, modification, reordering, or replays. The destruction of data is also covered under this service. Thus, the connection-oriented integrity service addresses both message stream modification and denial of service. On the other hand, a connectionless integrity service deals with individual messages without regard to any larger context. It provides protection against message modification only.

Non-repudiation

In general, nonrepudiation is the ability to ensure that a party to a contract or a communication cannot deny the authenticity of their signature on a document or the sending of a message that they originated. On the Internet, the digital signature is used not only to ensure that a message or document has been electronically signed by the person that purported to sign the document, but also, since a digital signature can only be created by one person, to ensure that a person cannot later deny that they furnished the signature.

Since no security technology is absolutely fool-proof, some experts warn that the digital signature alone may not always guarantee non-repudiation. It is suggested that multiple approaches be used, such as capturing unique biometric information and other data about the sender or signer that collectively would be difficult to repudiate.

Access Control

Access control is the ability to permit or deny the use of a particular resource by a particular entity. Access control mechanisms can be used in managing physical resources (such as a movie theater, to which only ticketholders should be admitted), logical resources (a bank account, with a limited number of people authorized to make a withdrawal), or digital resources (for example, a private text document on a computer, which only certain users should be able to read).

Availability

Availability is the property of a system or a system resource being accessible and usable upon demand by an authorized system entity, according to performance specifications for the system (i.e., a system is available if it provides services according to the system design whenever users request them). This refers to whether the network, system, hardware, and software are reliable and can recover quickly and completely in the event of an interruption in service. Ideally, these

elements should not be susceptible to denial of service attacks.

Active attacks present the opposite characteristics of passive attacks. Whereas passive attacks are difficult to detect, measures are available to prevent their success. On the other hand, it is quite difficult to prevent active attacks absolutely, because of the wide variety of potential physical, software, and network vulnerabilities. Instead, the goal is to detect active attacks and to recover from any disruption or delays caused by them. If the detection has a deterrent effect, it may also contribute to prevention.

A Model for Network Security

A model for generalized network security is shown in Figure 1.1.4. A message is to be transferred from one party to another across some sort of internet. The two parties, who are the principals in this transaction, must cooperate for the exchange to take place. A logical information channel is established by defining a route through the internet from source to destination and by the cooperative use of communication protocols (e.g., TCP/IP) by the two principals.

Security aspects come into play when it is necessary or desirable to protect the information transmission from an opponent who may present

a threat to confidentiality, authenticity, and so on. All the techniques for providing security have two components:

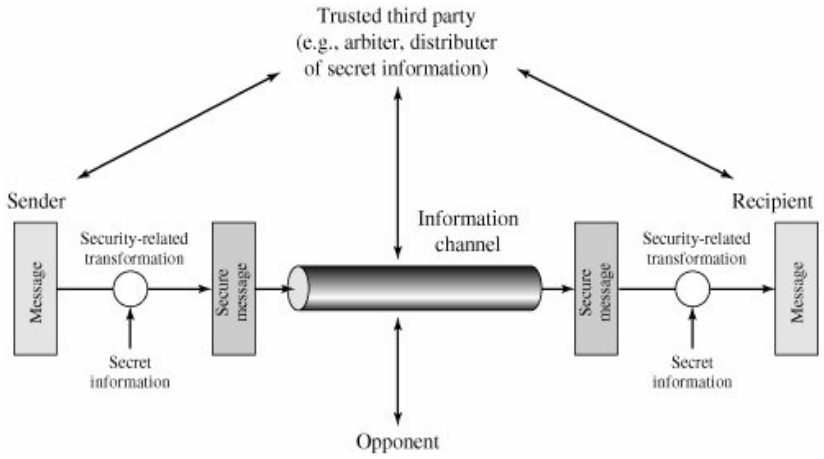


Fig 1.1.4 Model for Network Security

- A security-related transformation on the information to be sent. Examples include the encryption of the message, which scrambles the message so that it is unreadable by the opponent, and the addition of a code based on the contents of the message, which can be used to verify the identity of the sender
- Some secret information shared by the two principals and, it is hoped, unknown to the opponent. An example is an encryption key used in conjunction with the transformation to scramble the message before transmission and unscramble it on reception

A trusted third party may be needed to achieve secure transmission. For example, a third party may be responsible for distributing the secret information to the two principals while keeping it from any opponent. Or a third party may be needed to arbitrate disputes between the two principals concerning the authenticity of a message transmission.

This general model shows that there are four basic tasks in designing a particular security service:

1. Design an algorithm for performing the security-related transformation. The algorithm should be such that an opponent cannot defeat its purpose.

2. Generate the secret information to be used with the algorithm.
3. Develop methods for the distribution and sharing of the secret information.
4. Specify a protocol to be used by the two principals that makes use of the security algorithm and the secret information to achieve a particular security service.

A general model of these other situations is illustrated by Fig 1.1.5, which reflects a concern for protecting an information system from unwanted access.

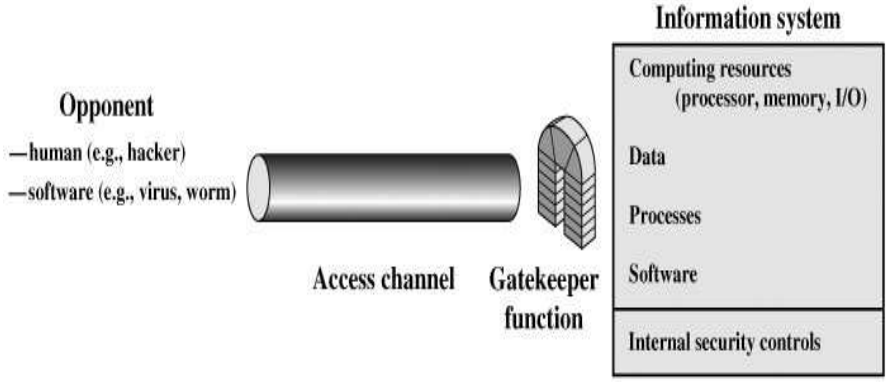


Fig 1.1.5 Network Access Security Model

The hacker can be someone who, with no malign intent, simply gets satisfaction from breaking and entering a computer system. Or, the intruder can be a disgruntled employee who wishes to do damage, or a criminal who seeks to exploit computer assets for financial gain (e.g., obtaining credit card numbers or performing illegal money transfers).

Another type of unwanted access is the placement in a computer system of logic that exploits vulnerabilities in the system and that can affect application programs as well as utility programs, such as editors and compilers. Programs can present two kinds of threats:

- Information access threats intercept or modify data on behalf of users who should not have access to that data.
- Service threats exploit service flaws in computers to inhibit use by legitimate users.

REFERENCES

1. William Stallings, Cryptography and Network Security, PHI Publishers
2. www.wikipedia.org

Review Questions:

1. Write short notes on different types of security attacks.
2. Explain the model for Network security?
3. What are the different types of security services?

2. CONVENTIONAL ENCRYPTION TECHNIQUES

OBJECTIVE

This lesson describes classical symmetric encryption techniques. It provides a gentle and interesting introduction to cryptography and cryptanalysis and highlights important concepts.

INTRODUCTION

Symmetric encryption also referred to as conventional encryption or single-key encryption was the only type of encryption in use prior to the development of public-key encryption in the 1970s. It is the most widely used of the two types of encryption.

Basic Definitions

Plain Text : An original message is known as Plain Text

Cipher Text : The coded message is called the cipher text.

Encryption : The process of converting from plaintext to ciphertext is known as enciphering or encryption

Decryption : restoring the plaintext from the ciphertext is deciphering or decryption.

The many schemes used for encryption constitute the area of study known as **cryptography**. Such a scheme is known as a **cryptographic system** or a **cipher**. Techniques used for deciphering a message without any knowledge of the enciphering details fall into the area of **cryptanalysis**. Cryptanalysis is what the layperson calls "breaking the code."

Symmetrical Cipher Model

A symmetric encryption scheme has five ingredients (**Fig 1.2.1**)

- **Plaintext:** This is the original intelligible message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The encryption algorithm performs various substitutions and transformations on the plaintext.
- **Secret key:** The secret key is also input to the encryption algorithm. The key is a value independent of the plaintext and of the algorithm. The algorithm will produce a different output depending on the specific key being used at the time. The exact

substitutions and transformations performed by the algorithm depend on the key.

- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different cipher texts. The ciphertext is an apparently random stream of data and, as it stands, is unintelligible.
- **Decryption algorithm:** This is essentially the encryption algorithm run in reverse. It takes the ciphertext and the secret key and produces the original plaintext.

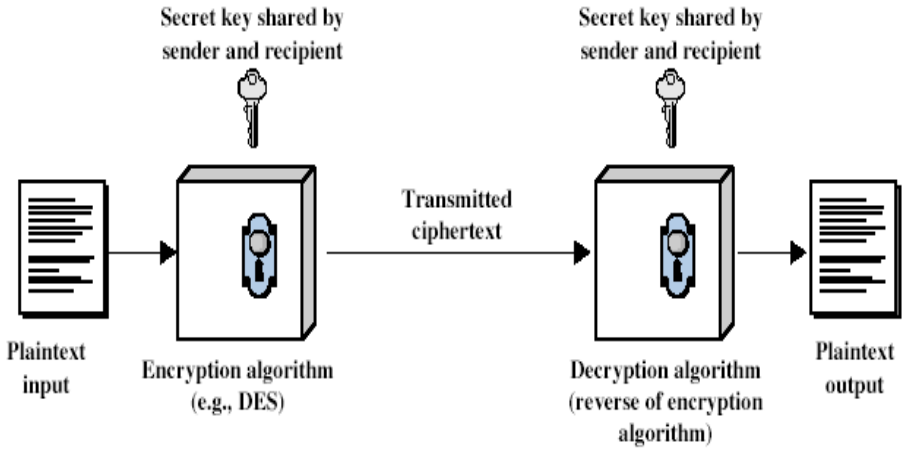


Fig 1.2.1 Simplified Model of Conventional Encryption

There are two requirements for secure use of conventional encryption:

1. We need a strong encryption algorithm. At a minimum, we would like the algorithm to be such that an opponent who knows the algorithm and has access to one or more ciphertexts would be unable to decipher the ciphertext or figure out the key. This requirement is usually stated in a stronger form: The opponent should be unable to decrypt ciphertext or discover the key even if he or she is in possession of a number of ciphertexts together with the plaintext that produced each ciphertext.
2. Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure. If someone can discover the key and knows the algorithm, all communication using this key is readable.

We assume that it is impractical to decrypt a message on the basis of the ciphertext plus knowledge of the encryption/decryption algorithm. In other

words, we do not need to keep the algorithm secret; we need to keep only the key secret. This feature of symmetric encryption is what makes it feasible for widespread use.

Conventional Cryptosystem

Consider a model conventional cryptosystem as shown in **Fig** 1.2.2.

A source produces a message in plaintext, $X = [X_1, X_2, ..., X_M]$. The M elements of X are letters in some finite alphabet. Traditionally, the alphabet usually consisted of the 26 capital letters. Nowadays, the binary alphabet {0, 1} is typically used. For encryption, a key of the form $K = [K_1, K_2, ..., K_J]$ is generated. If the key is generated at the message source, then it must also be provided to the destination by means of some secure channel. Alternatively, a third party could generate the key and securely deliver it to both source and destination. With the message X and the encryption key K as input, the encryption algorithm forms the ciphertext $Y = [Y_1, Y_2, ..., Y_N]$. We can write this as

$$Y = \mathbf{E}(K, X)$$

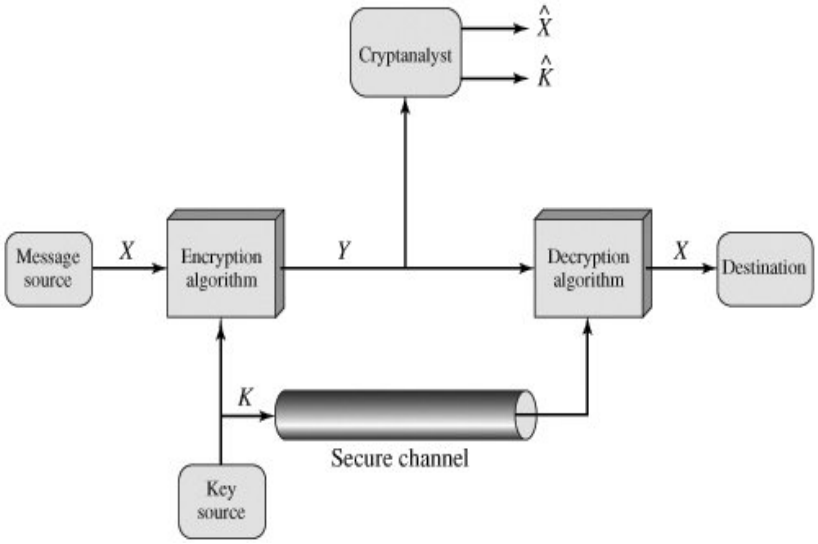


Fig 1.2.2 Model Conventional Cryptosystem

This notation indicates that Y is produced by using encryption algorithm E as a function of the plaintext X, with the specific function determined by the value of the key K.

The intended receiver, in possession of the key, is able to invert the transformation:

$$X = \mathbf{D}(K, Y)$$

An opponent, observing Y but not having access to K or X, may attempt to recover X or K or both X and K. It is assumed that the opponent knows the encryption (E) and decryption (D) algorithms. If the opponent is interested in only this particular message, then the focus of the effort is to recover X by generating a plaintext estimate $\hat{X}$. Often, however, the opponent is interested in being able to read future messages as well, in which case an attempt is made to recover K by generating an estimate $\hat{K}$

Cryptography

Cryptographic systems are characterized along three independent dimensions:

1. The type of operations used for transforming plaintext to ciphertext. All encryption algorithms are based on two general principles: *substitution*, in which each element in the plaintext (bit, letter, group of bits or letters) is mapped into another element, and *transposition*, in which elements in the plaintext are rearranged. The fundamental requirement is that *no information be lost* (that is, that *all operations are reversible*). Most systems, referred to as product systems, involve multiple stages of substitutions and transpositions.
2. The number of keys used. If both sender and receiver use the same key, the system is referred to as symmetric, single-key, secret-key, or conventional encryption. If the sender and receiver use different keys, the system is referred to as asymmetric, two-key, or public-key encryption.
3. The way in which the plaintext is processed. A block cipher processes the input one block of elements at a time, producing an output block for each input block. A stream cipher processes the input elements continuously, producing output one element at a time, as it goes along.

ATTACKING CRYPTOSYSTEMS

The objective of attacking an encryption system is to recover the key in use rather than simply to recover the plaintext of a single ciphertext. There are two general approaches to attacking a conventional encryption scheme:

- **Cryptanalysis:** Cryptanalytic attacks rely on the nature of the algorithm plus perhaps some knowledge of the general characteristics of the plaintext or even some sample plaintext-ciphertext pairs. This type of attack exploits the characteristics of

the algorithm to attempt to deduce a specific plaintext or to deduce the key being used.

- **Brute-force attack:** The attacker tries every possible key on a piece of ciphertext until an intelligible translation into plaintext is obtained. On average, half of all possible keys must be tried to achieve success.

If either type of attack succeeds in deducing the key, all future and past messages encrypted with that key are compromised.

Cryptanalysis

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

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

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

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

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

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

An ***adaptive-chosen-ciphertext*** is the adaptive version of the above attack. A cryptanalyst can mount an attack of this type in a scenario in which he has free use of a piece of decryption hardware, but is unable to extract the decryption key from it. All the attacks are tabled in Table 1.2.1

Type of Attack	Known to Cryptanalyst
Ciphertext only	• Encryption algorithm • Ciphertext
Known plaintext	• Encryption algorithm • Ciphertext • One or more plaintext-ciphertext pairs formed with the secret key
Chosen plaintext	• Encryption algorithm • Ciphertext • Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key
Chosen ciphertext	• Encryption algorithm • Ciphertext • Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key
Chosen text	• Encryption algorithm • Ciphertext • Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key • Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key

Table 1.2.1 Types of Attacks on Encrypted Messages

NOTE

1) An encryption scheme is ***unconditionally secure*** if the ciphertext generated by the scheme does not contain enough information to determine uniquely the corresponding plaintext, no matter how much ciphertext is available. That is, no matter how much time an opponent has, it is impossible for him or her to decrypt the ciphertext, simply because the required information is not there. **Except** a scheme known as the ***one-time pad***, there is **no encryption algorithm** that **is *unconditionally secure***. Therefore,

all that the users of an encryption algorithm can strive for is an algorithm that meets one or both of the following criteria:

- The cost of breaking the cipher exceeds the value of the encrypted information.
- The time required to break the cipher exceeds the useful lifetime of the information.

2) An encryption scheme is said to be ***computationally secure*** if either of the foregoing two criteria are met.

- All forms of cryptanalysis for symmetric encryption schemes are designed to exploit the fact that traces of structure or pattern in the plaintext may survive encryption and be visible in the ciphertext. Cryptanalysis for public-key schemes proceeds from a fundamentally different premise, namely, that the mathematical properties of the pair of keys may make it possible for one of the two keys to be deduced from the other.
- A **brute-force attack** involves trying every possible key until an intelligible translation of the ciphertext into plaintext is obtained. On average, half of all possible keys must be tried to achieve success. Table 1.2.2 shows how much time is involved for various key spaces. Results are shown for four binary key sizes. The 56-bit key size is used with the DES (Data Encryption Standard) algorithm, and the 168-bit key size is used for triple DES. The minimum key size specified for AES (Advanced Encryption Standard) is 128 bits. Results are also shown for what are called substitution codes that use a 26-character key (discussed later), in which all possible permutations of the 26 characters serve as keys. For each key size, the results are shown assuming that it takes 1 μs to perform a single decryption, which is a reasonable order of magnitude for today's machines. With the use of massively parallel organizations of microprocessors, it may be possible to achieve processing rates many orders of magnitude greater. The final column of Table 1.2.2 considers the results for a system that can process 1 million keys per microsecond. As you can see, at this performance level, DES can no longer be considered computationally secure.

Key size (bits)	Number of alternative keys		Time required at 1 decryption/μs		Time required at 10^6 decryption/μs
32	2^{32}	= 4.3 x 10^9	2^{31} μs	= 35.8 minutes	2.15 milliseconds
56	2^{56}	= 7.2 x 10^{16}	2^{55} μs	= 1142 years	10.01 hours
128	2^{128}	= 3.4 x 10^{38}	2^{127} μs	= 5.4 x 10^{24} years	5.4 x 10^{18} years
168	2^{168}	= 3.7 x 10^{50}	2^{167} μs	= 5.9 x 10^{36} years	5.9 x 10^{30} years
26 characters (permutation)	26!	= 4 x 10^{26}	2 x 10^{26} μs	= 6.4 x 10^{12} years	6.4 x 10^6 years

Table 1.2.2 Average Time Required for Exhaustive Key Search

ENCRYPTION TECHNIQUES

The two basic building blocks of all encryption techniques are

- Substitution
- Transposition.

Substitution Techniques

In cryptography, a substitution cipher is a method of encryption by which units of plaintext are substituted with ciphertext according to a regular system; the "units" may be single letters (the most common), pairs of letters, triplets of letters, mixtures of the above, and so forth. The receiver deciphers the text by performing an inverse substitution.

There are a number of different types of substitution cipher. If the cipher operates on single letters, it is termed a simple substitution cipher; a cipher that operates on larger groups of letters is termed polygraphic. A monoalphabetic cipher uses fixed substitution over the entire message, whereas a polyalphabetic cipher uses a number of substitutions at different times in the message.

CAESAR CIPHER

In cryptography, a Caesar cipher, also known as a Caesar's cipher, the shift cipher, Caesar's code or Caesar shift, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some

fixed number of positions down the alphabet. For example, with a shift of 3, A would be replaced by D, B would become E, and so on (see **Fig 1.2.3**). The method is named after Julius Caesar, who used it to communicate with his generals.

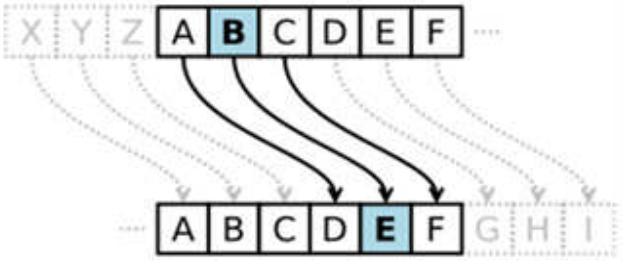


Fig 1.2.3 The action of a Caesar cipher is to replace each plaintext letter with one a fixed number of places down the alphabet. This example is with a shift of three, so that a B in the plaintext becomes E in the ciphertext

The encryption step performed by a Caesar cipher is often incorporated as part of more complex schemes, such as the Vigenère cipher, and still has modern application in the ROT13 system (**Fig 1.2.4**).

The encryption can also be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1... Z = 25.

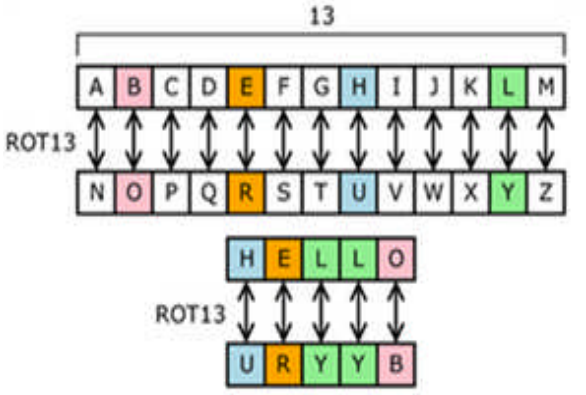


Fig 1.2.4 ROT13 is a Caesar cipher, in which the alphabet is rotated 13 steps.

Encryption of a *letter* **x** by a *shift* **n** can be described mathematically as,

$$E_n(x) = (x + n) \bmod 26$$

Decryption is performed similarly,

$$D_n(x) = (x - n) \bmod 26$$

If it is known that a given ciphertext is a Caesar cipher, then a brute-force cryptanalysis is easily performed: Simply try all the 25 possible keys. Table 1.3 shows the results of applying this strategy to the example ciphertext.

Let us suppose that the plain text 'HELLO' is encrypted using ROT13 and the resultant cipher text is URYYB. The cryptanalyst will try to decrypt the message as follows

Shifting by	Resultant Plain text
1	VSZZC
2	WTAAD
3	XU BBE
4	YVCCF
5	Z WDDG
6	AXEEH
7	BYFFI
8	CZGGJ
9	DAHHK
10	EBIIL
11	FCJJM
12	GDKKN
13	**HELLO**
14	IFMMP
15	JGNNQ
16	KHOOR
17	LIPPS
18	MJQQT
19	NKRRU
20	OLSSV
21	PMTTW
22	QNUUX
23	ROVVY
24	SPWWZ
25	TQXX

Table 1.2.3 Cryptanalysis of message cipher text URYYB

As with all single alphabet substitution ciphers, the Caesar cipher is easily broken and in practice offers essentially no communication security. With only 25 possible keys, the Caesar cipher is far from secure. A dramatic

increase in the key space can be achieved by allowing an arbitrary substitution.

MONOALPHABETIC CIPHER

Ciphers in which the cipher alphabet remains unchanged throughout the message are called Monoalphabetic Substitution Ciphers. The Caesar Shift Cipher is one example of a *Monoalphabetic Cipher*. In a monoalphabetic cipher, every letter in the alphabet is represented by exactly one other letter in the key. The monoalphabetic cipher is not restricted to using only letters, but can use symbols and numbers as well. Let's explore another example of a monoalphabetic cipher

To construct the key, first select a keyword. We'll use "Butterfly". Now rewrite the codeword but get rid of any duplicate letters. So Butterfly" becomes "Buterfly", leaving the second "t" out. Write out the alphabet as before and underneath write out the new keyword. Write out the rest of the alphabet in order, leaving out any letters in the keyword.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

B U T E R F L Y A C D G H I J K M N O P Q S V W X Z

You have now constructed the key for the cipher. A will be replaced by B, B with U, C with T and so on.

Consider the plain text **:** NETWORK SECURITY IS RULING THE WORLD

Corresponding Cipher **:** IRPVJND ORTQNAPX AO NQGAIL PYR VJNGE

The same key is also available at the receiver end and he performs the reverse mapping on the cipher text to obtain the plain text. Encrypting and decrypting works exactly the same for all monoalphabetic ciphers, the only difference is how the key is constructed.

Frequency Analysis

Frequency analysis exploits the known patterns of the language used in a cryptographic system. In our case, this will be English, but the same principles could be applied to any language.

Certain letters and combinations of letters are more likely than others to occur in any arbitrary English sentence. For example, the letter "e" is the most common single letter and the digram "th" is the most common two letter combination. Fig 1.2.5 shows the percentage of occurrences (i.e. frequency) of each letter in the alphabet for English text.

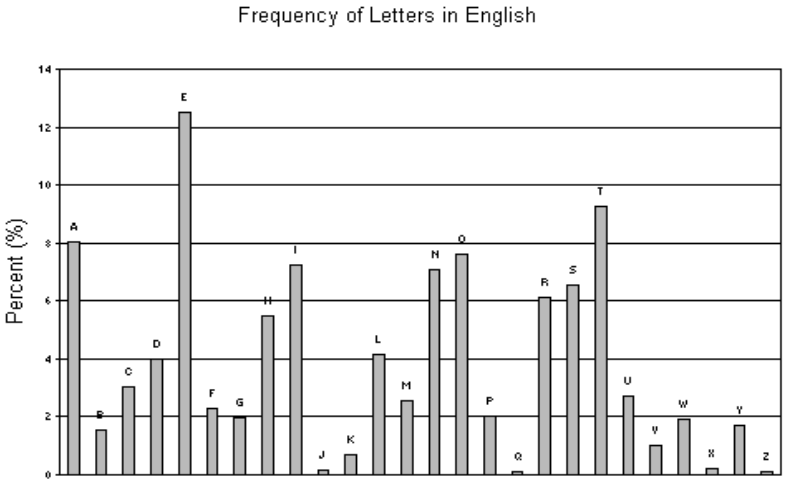


Fig 1.2.5 Frequency of Letters in English

This knowledge can be used to attack monoalphabetic ciphers because all occurrences of the same letter in a message will be encrypted the same way.

The Attack

1. Calculate the frequency of letters in the ciphertext and compare this to the known frequencies in the language.
2. Then make an intelligent guess of how to decrypt the message.

For example, if we analysed a ciphertext and found that the most commonly occurring letter was H, there is a good chance that all H's would be E's in the plaintext. This is not guaranteed to be the case, of course. The shorter the message, the more likely the frequencies will be unusual. This method is still much faster than trying all possibilities and on longer messages, it will be extremely effective.

PLAYFAIR CIPHER

The Playfair cipher encrypts pairs of letters (digraphs), instead of single letters. This is significantly harder to break since the frequency analysis used for simple substitution ciphers is considerably more difficult.

The Playfair cipher uses a 5 by 5 table containing a key word or phrase. To generate the table, one would first fill in the spaces of the table with the letters of the keyword (dropping any duplicate letters), then fill the remaining spaces with the rest of the letters of the alphabet in order (to reduce the alphabet to fit you can either omit "Q" or replace "J" with "I").

To encrypt a message, one would break the message into groups of 2 letters. If there is a dangling letter at the end, we add an X. For example,

"Secret Message" becomes "SE CR ET ME SS AG EX". We now take each group and find them out on the table. Noticing the location of the two letters in the table, we apply the following rules, in order.

1. If both letters are the same, add an X between them. Encrypt the new pair, re-pair the remaining letters and continue.
2. If the letters appear on the same row of your table, replace them with the letters to their immediate right respectively, wrapping around to the left side of the row if necessary. For example, using the table above, the letter pair GJ would be encoded as HF.
3. If the letters appear on the same column of your table, replace them with the letters immediately below, wrapping around to the top if necessary. For example, using the table above, the letter pair MD would be encoded as UG.

K	E	Y	W	O
R	D	A	B	C
F	G	H	I/J	L
M	N	P	Q	S
T	U	V	X	Z

Table 1.2.4 5 X 5 Table generated for the key KEYWORD

4. If the letters are on different rows and columns, replace them with the letters on the same row respectively but at the other pair of corners of the rectangle defined by the original pair. The order is important - the first letter of the pair should be replaced first. For example, using the table above, the letter pair EB would be encoded as WD.

To decipher, ignore rule 1. In rules 2 and 3 shift up and left instead of down and right. Rule 4 remains the same. Once you are done, drop any extra Xs that don't make sense in the final message and locate any missing Qs or any Is that should be Js.

Example: (see Table 1.2.4)

Key : keyword

Plaintext : thisisasecretmessage

Cipher Text : vf jp jp cn od dk ul om nc md

POLYALPHABETIC CIPHER

Frequency analysis was a devastating attack on monoalphabetic ciphers and they could no longer be considered secure. The next development in cryptography was Polyalphabetic ciphers. The idea behind the polyalphabetic cipher is that a single letter can be encrypted to several different letters instead of just one.

In a Caesar cipher, each letter of the alphabet is shifted along some number of places; for example, in a Caesar cipher of shift 3, A would become D, B would become E and so on. The Vigenère cipher consists of several Caesar ciphers in sequence with different shift values.

To encipher, a table of alphabets can be used, termed a **tabula recta, Vigenère square,** or **Vigenère table**. It consists of the alphabet written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar ciphers. At different points in the encryption process, the cipher uses a different alphabet from one of the rows. The alphabet used at each point depends on a repeating keyword.

For example, suppose that the plaintext to be encrypted is:

ATTACKATDAWN

The person sending the message chooses a keyword and repeats it until it matches the length of the plaintext, for example, the keyword "LEMON":

LEMONLEMONLE

The first letter of the plaintext, A, is enciphered using the alphabet in row L, which is the first letter of the key. This is done by looking at the letter in row L and column A of the Vigenère square, namely L. Similarly, for the second letter of the plaintext, the second letter of the key is used; the letter at row E and column T is X. The rest of the plaintext is enciphered in a similar fashion:

Plaintext: ATTACKATDAWN

Key: LEMONLEMONLE

Ciphertext: LXFOPVEFRNHR

The keyspace consists of all ordered permutations of the alphabet and we call it a Vigenère square (Tab. 1.2.5). You can see there are 25 rows that can be used as keys, each numbered with the amount they are shifted.

Decryption is performed by finding the position of the ciphertext letter in a row of the table, and then taking the label of the column in which it appears as the plaintext. For example, in row L, the ciphertext L appears in column A, which taken as the first plaintext letter. The second letter is

decrypted by looking up X in row E of the table; it appears in column T, which is taken as the plaintext letter.

Vigenère can also be viewed algebraically. If the letters A–Z are taken to be the numbers 0–25, and addition is performed modulo 26, then Vigenère encryption can be written,

$$C_i \equiv P_i + K_i \ (mod\ 26)$$

and decryption,

$$P_i \equiv C_i - K_i \ (mod\ 26)$$

Cryptanalysis

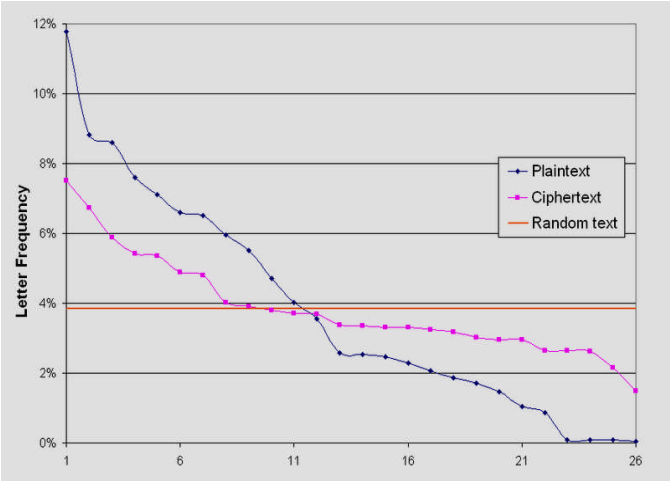
The idea behind the Vigenère cipher, like all polyalphabetic ciphers, is to disguise plaintext letter frequencies, which interferes with a straightforward application of frequency analysis. **Fig** 1.2.6 shows the Vigenere letter frequencies. For instance, if P is the most frequent letter in a ciphertext whose plaintext is in English, one might suspect that P corresponds to E, because E is the most frequently used letter in English. However, using the Vigenère cipher, E can be enciphered as different ciphertext letters at different points in the message, thus defeating simple frequency analysis.

The primary weakness of the Vigenère cipher is the repeating nature of its key. If a cryptanalyst correctly guesses the key's length, then the cipher text can be treated as interwoven Caesar ciphers, which individually are easily broken.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Table 1.2.5 Vigenère square

Fig 1.2.6



Vigenere letter frequencies

Transposition Techniques

In classical cryptography, a transposition cipher changes one character from the plaintext to another (to decrypt the reverse is done). That is, the order of the characters is changed. Mathematically a bijective function is used on the characters' positions to encrypt and an inverse function to decrypt.

Rail Fence cipher

The Rail Fence cipher is a form of transposition cipher that gets its name from the way in which it is encoded. In the rail fence cipher, the plaintext is written downwards on successive "rails" of an imaginary fence, then moving up when we get to the bottom. The message is then read off in rows. For example, using **three "rails"** (i.e., Depth = 3) and a message of 'WE ARE DISCOVERED. FLEE AT ONCE', the cipherer writes out:

W . . . E . . . C . . . R . . . L . . . T . . . E

. E . R . D . S . O . E . E . F . E . A . O . C .

. . A . . . I . . . V . . . D . . . E . . . N . .

Then reads off:

WECRLTEERDSOEEFEAOCAIVDEN

Columnar transposition

In a columnar transposition, the message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order. Both the width of the rows and the permutation of the columns are usually defined by a keyword. For

example, the word ZEBRAS is of length 6 (so the rows are of length 6), and the permutation is defined by the alphabetical order of the letters in the keyword. In this case, the order would be

6 3 2 4 1 5

In a regular columnar transposition cipher, any spare spaces are filled with nulls; in an irregular columnar transposition cipher, the spaces are left blank. Finally, the message is read off in columns, in the order specified by the keyword. For example, suppose we use the keyword ZEBRAS and the message WE ARE DISCOVERED. FLEE AT ONCE. In a regular columnar transposition, we write this into the grid as:

6 3 2 4 1 5
W E A R E D
I S C O V E
R E D F L E
E A T O N C
E

This results in the following ciphertext:

EVLNACDTESEAROFODEECWIREE

To decipher it, the recipient has to work out the column lengths by dividing the message length by the key length. Then he can write the message out in columns again, and then re-order the columns by reforming the key word.

ONE TIME PAD

A one-time pad is a system in which a private key generated randomly is used only once to encrypt a message that is then decrypted by the receiver using a matching one-time pad and key. Messages encrypted with keys based on randomness have the advantage that there is theoretically no way to "break the code" by analyzing a succession of messages. Each encryption is unique and bears no relation to the next encryption so that some pattern can be detected. With a one-time pad, however, the decrypting party must have access to the same key used to encrypt the message and this raises the problem of how to get the key to the decrypting party safely or how to keep both keys secure. One-time pads have sometimes been used when the both parties started out at the same physical location and then separated, each with knowledge of the keys in the one-time pad. The key used in a one-time pad is called a secret key because if it is revealed, the messages encrypted with it can easily be deciphered. One-time pads figured prominently in secret

message transmission and espionage before and during World War II and in the Cold War era.

Suppose Alice wishes to send the message 'HELLO' to Bob. Assume two pads of paper containing identical random sequences of letters were somehow previously produced and securely issued to both. Alice chooses the appropriate unused page from the pad. The way to do this is normally arranged for in advance, as for instance 'use the 12th sheet on Labor Day', or 'use the next available sheet for the next message'. The material on the selected sheet is the *key* for this message. Each letter from the pad will be combined in a predetermined way with one letter of the message. It is common, but not required, to assign each letter a numerical value: e.g., "A" is 0, "B" is 1, and so on through "Z", 25. In this example, the technique is to combine the key and the message using modular addition. The numerical values of corresponding message and key letters are added together, modulo 26. If key material begins with,

X M C K L

and the message is "HELLO", then the coding would be done as follows:

```
   7(H) 4 (E) 11(L) 11(L)  14 (O)message
+ 23(X)12 (M)  2 C) 10(K)  11 (L)key
= 30   16     13    21     25    message + key
 = 4(E) 16 (Q) 13(N) 21 (V) 25(Z)message + key (mod 26)
                                    >> ciphertext
```

Note that if a number is larger than 25, then in modular arithmetic fashion, the remainder after subtraction of 26 is taken. This simply means that, if your computations "go past" Z, you start again at A.

The ciphertext to be sent to Bob is thus "EQNVZ." Bob uses the matching key page and the same process, but in reverse, to obtain the plaintext. Here, the key is *subtracted* from the ciphertext, again using modular arithmetic:

```
    4 (E) 16(Q) 13 (N)  21 (V)  25 (Z) ciphertext
-  23 (X) 12(M)  2 (C)  10 (K)  11 (L) key
=  -19      4    11      11      14   ciphertext - key
=  7(H)  4(E) 11(L) 11(L) 14 (O)ciphertext-key(mod 26)
                                    >> message
```

Similar to above, if a number is negative, 26 is added to make the number positive.

Thus, Bob recovers Alice's plaintext, the message "HELLO". Both Alice and Bob destroy the key sheet immediately after use, thus preventing reuse and an attack against the cipher. The KGB often issued its agents

one-time pads printed on tiny sheets of "flash paper", paper chemically converted to nitrocellulose, which burns almost instantly and leaves no ash.

ROTOR MACHINES

In cryptography, a **rotor machine** is an electro-mechanical device used for encrypting and decrypting secret messages. Rotor machines were the cryptographic state-of-the-art for a brief but prominent period of history; they were in widespread use in the 1930s–1950s.

The primary component is a set of *rotors*, also termed *wheels* or *drums*, which are rotating disks with an array of electrical contacts on either side. The wiring between the contacts implements a fixed substitution of letters, scrambling them in some complex fashion. On its own, this would offer little security; however, after encrypting each letter, the rotors advance positions, changing the substitution. By this means, a rotor machine produces a complex polyalphabetic substitution cipher.

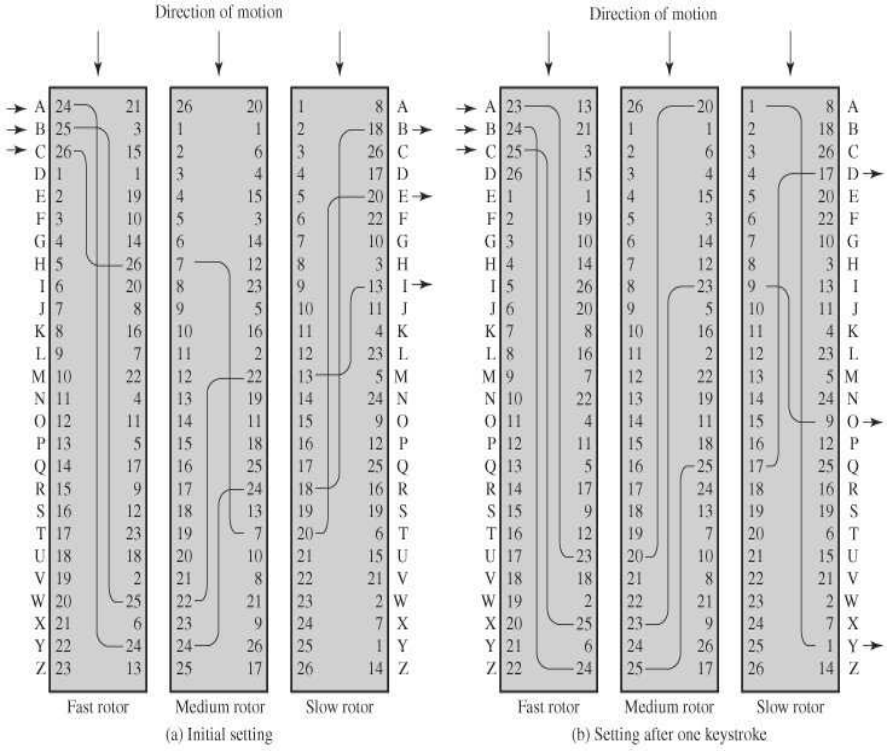


Fig 1.2.7 Three-Rotor Machine with Wiring Represented by Numbered Contacts

The basic principle of the rotor machine is illustrated in **Fig 1.2.7**. The machine consists of a set of independently rotating cylinders through which electrical pulses can flow. Each cylinder has 26 input pins and 26 output pins, with internal wiring that connects each input pin to a unique output pin. For simplicity, only three of the internal connections in each cylinder are shown.

If we associate each input and output pin with a letter of the alphabet, then a single cylinder defines a monoalphabetic substitution. For example, in Fig 1.2.7, if an operator depresses the key for the letter A, an electric signal is applied to the first pin of the first cylinder and flows through the internal connection to the twenty-fifth output pin.

Consider a machine with a single cylinder. After each input key is depressed, the cylinder rotates one position, so that the internal connections are shifted accordingly. Thus, a different monoalphabetic substitution cipher is defined. After 26 letters of plaintext, the cylinder would be back to the initial position. Thus, we have a polyalphabetic substitution algorithm with a period of 26.

A single-cylinder system is trivial and does not present a formidable cryptanalytic task. The power of the rotor machine is in the use of multiple cylinders, in which the output pins of one cylinder are connected to the input pins of the next. Fig 1.2.7 shows a three-cylinder system. The left half of the figure shows a position in which the input from the operator to the first pin (plaintext letter a) is routed through the three cylinders to appear at the output of the second pin (ciphertext letter B).

With multiple cylinders, the one closest to the operator input rotates one pin position with each keystroke. The right half of Fig 1.2.7 shows the system's configuration after a single keystroke. For every complete rotation of the inner cylinder, the middle cylinder rotates one pin position. Finally, for every complete rotation of the middle cylinder, the outer cylinder rotates one pin position. This is the same type of operation seen with an odometer. The result is that there are 26 x 26 x 26 = 17,576 different substitution alphabets used before the system repeats. The addition of fourth and fifth rotors results in periods of 456,976 and 11,881,376 letters, respectively.

The significance of the rotor machine today is that it points the way to the most widely used cipher ever: the **Data Encryption Standard (DES).**

STEGANOGRAPHY

Steganography is the art and science of writing hidden messages in such a way that no one apart from the sender and intended recipient even realizes there *is* a hidden message. By contrast, cryptography obscures

the meaning of a message, but it does not conceal the fact that there *is* a message. Today, the term steganography includes the concealment of digital information within computer files. For example, the sender might start with an ordinary-looking image file, then adjust the color of every 100th pixel to correspond to a letter in the alphabet, a change so subtle that someone who isn't actively looking for it is unlikely to notice it.

The word *steganography* is of Greek origin and means *"covered, or hidden writing"*. Generally, a steganographic message will appear to be something else: a picture, an article, a shopping list, or some other message. This apparent message is the *covertext*. For instance, a message may be hidden by using invisible ink between the visible lines of innocuous documents.

Consider the following message.

Fi**s**hing fr**e**shwater be**n**ds an**d** sa**l**twater
co**a**sts re**w**ards an**y**one fe**e**ling st**r**essed.
Re**s**ourceful an**g**lers us**u**ally fi**n**d ma**s**terful
le**a**pers fu**n** an**d** ad**m**it sw**o**rdfish ra**n**k
ov**e**rwhelming an**y**day.

By taking the third letter in each word, the following message emerges:

Send Lawyers, Guns, and Money.

The following message was actually sent by a German Spy in WWII (World War II)

A**p**parently n**e**utral's p**r**otest i**s** t**h**oroughly d**i**scounted
a**n**d ignored. I**s**man hard h**i**t. B**l**ockade issue affects
pretext for embargo on byproducts, ejecting suets and
vegetable oils.

Taking the second letter in each word the following message emerges:

Pershing sails from NY June 1.

Various other techniques have been used historically; some examples are the following

- **Character marking:** Selected letters of printed or typewritten text are overwritten in pencil. The marks are ordinarily not visible unless the paper is held at an angle to bright light.
- **Invisible ink:** A number of substances can be used for writing but leave no visible trace until heat or some chemical is applied to the paper.
- **Pin punctures:** Small pin punctures on selected letters are ordinarily not visible unless the paper is held up in front of a light.

- **Typewriter correction ribbon:** Used between lines typed with a black ribbon, the results of typing with the correction tape are visible only under a strong light.

Steganography has a number of drawbacks when compared to encryption. It requires a lot of overhead to hide a relatively few bits of information, although using some scheme like that proposed in the preceding paragraph may make it more effective. Also, once the system is discovered, it becomes virtually worthless. This problem, too, can be overcome if the insertion method depends on some sort of key. Alternatively, a message can be first encrypted and then hidden using steganography.

The advantage of steganography is that it can be employed by parties who have something to lose should the fact of their secret communication (not necessarily the content) be discovered. Encryption flags traffic as important or secret or may identify the sender or receiver as someone with something to hide.

REFERENCES

1. William Stallings, Cryptography and Network Security, PHI Publishers
2. www.wikipedia.org

Review Questions:

1. Give an account of Cryptanalysis
2. Explain Substitution and Transposition techniques with examples
3. Explain the conventional encryption model.
4. Write a short note on steganography.

3. BLOCK CIPHERS AND DES

A block cipher is a symmetric key cipher which operates on fixed-length groups of bits, termed *blocks*, with an unvarying transformation. When encrypting, a block cipher might take (for example) a 128-bit block of plaintext as input, and output a corresponding 128-bit block of ciphertext. The exact transformation is controlled using a second input — the secret key. Decryption is similar: the decryption algorithm takes, in this example, a 128-bit block of ciphertext together with the secret key, and yields the original 128-bit block of plaintext.

A block cipher consists of two paired algorithms, one for encryption, E, and another for decryption, E^{-1}. Both algorithms accept two inputs: an input block of size n bits and a key of size k bits, yielding an n-bit output block. For any one fixed key, decryption is the inverse function of encryption, so that

$$E_K^{-1}(E_K(M)) = M$$

for any block M and key K.

For each key K, E_K is a permutation (a bijective mapping) over the set of input blocks. Each key selects one permutation from the possible set of $2^n!$.

The block size, n, is typically 64 or 128 bits, although some ciphers have a variable block size. 64 bits was the most common length until the mid-1990s, when new designs began to switch to the longer 128-bit length. One of several modes of operation is generally used along with a padding scheme to allow plaintexts of arbitrary lengths to be encrypted. Each mode has different characteristics in regard to error propagation, ease of random access and vulnerability to certain types of attack. Typical key sizes (k) include 40, 56, 64, 80, 128, 192 and 256 bits. As of 2006, 80 bits is normally taken as the minimum key length needed to prevent brute force attacks.

ITERATED BLOCK CIPHERS

Most block ciphers are constructed by repeatedly applying a simpler function. This approach is known as *iterated block cipher* (see also product cipher). Each iteration is termed a *round*, and the repeated function is termed the *round function*; anywhere between 4 to 32 rounds are typical.

Many block ciphers can be categorised as Feistel networks, or, as more general substitution-permutation networks. Arithmetic operations, logical operations (especially XOR), S-boxes and various permutations are all frequently used as components.

FIESTEL NETWORK

A **Feistel cipher** (F**ig 1.3.1**) is a symmetric structure used in the construction of block ciphers, named after the German IBM cryptographer Horst Feistel; it is also commonly known as a **Feistel network**. A large proportion of block ciphers use the scheme, including the Data Encryption Standard (DES). The Feistel structure has the advantage that encryption and decryption operations are very similar, even identical in some cases, requiring only a reversal of the key schedule. Therefore the size of the code or circuitry required to implement such a cipher is nearly halved.

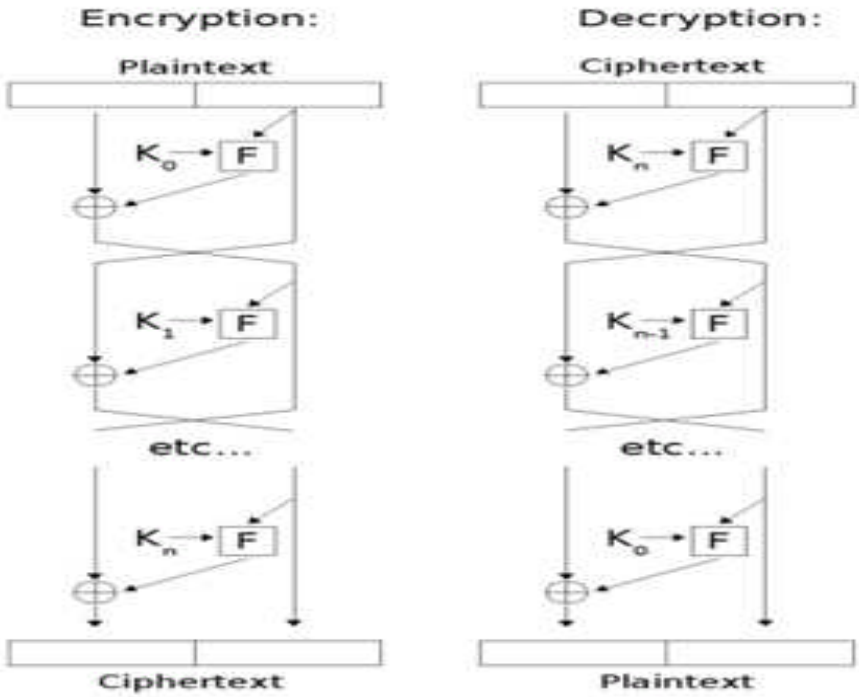


Fig 1.3.1 Feistel Cipher

Feistel networks and similar constructions are product ciphers, and so combine multiple rounds of repeated operations, such as:

- Bit-shuffling (often called permutation boxes or P-boxes)
- Simple non-linear functions (often called substitution boxes or S-boxes)
- Linear mixing (in the sense of modular algebra) using XOR to produce a function with large amounts of what Claude Shannon described as "confusion and diffusion".

Bit shuffling creates the diffusion effect, while substitution is used for confusion.

Construction Details

Let F be the round function and let K_0, K_1, .. K_n be the sub-keys for the rounds 0, 1, 2, ..n respectively. Then the basic operation is as follows.

Split the plaintext block into two equal pieces, (L_0, R_0). For each round I = 0, 1, 2, … N , compute

$$L_{i+1} = R_i$$
$$R_{i+1} = L_i \oplus F(R_i, K_i)$$

Then the ciphertext is (R_n,L_n). (Commonly the two pieces R_n and L_n are not switched after the last round.)

Decryption of a ciphertext (R_n,L_n) is accomplished by computing for I = n, n-1, n-2, ..0.

$$R_i = L_{i+1}$$
$$L_i = R_{i+1} \oplus F(L_{i+1}, K_i)$$

Then (L_0,R_0) is the plaintext again.

One advantage of this model is that the round function F does not have to be invertible, and can be very complex.

The diagram illustrates both encryption and decryption. Note the reversal of the subkey order for decryption. This is the only difference between encryption and decryption:

Unbalanced Feistel ciphers use a modified structure where L_0 and R_0 are not of equal lengths. The Skipjack encryption algorithm is an example of such a cipher. The Texas Instruments Digital Signature Transponder uses a proprietary unbalanced Feistel cipher to perform challenge-response authentication.

Diffusion and Confusion

In cryptography, **confusion** and **diffusion** are two properties of the operation of a secure cipher which were identified by Shannon in his paper, "Communication Theory of Secrecy Systems" published in 1949.

In Shannon's original definitions, *confusion* refers to making the relationship between the key and the ciphertext as complex and as involved as possible; *diffusion* refers to the property that redundancy in the statistics of the plaintext is "dissipated" in the statistics of the ciphertext.

Diffusion is associated with dependency of bits of the output on bits of the input. In a cipher with good diffusion, flipping an input bit should change each output bit with a probability of one half (this is termed the *Strict Avalanche Criterion*).

Substitution (a plaintext symbol is replaced by another) has been identified as a mechanism for primarily confusion. Conversely transposition (rearranging the order of symbols) is a technique for diffusion, although other mechanisms are also used in modern practice, such as linear transformation. Product ciphers use alternating substitution and transposition phases to achieve both confusion and diffusion respectively.

DATA ENCRYPTION STANDARD

The **Data Encryption Standard** (**DES**) is a cipher (a method for encrypting information) selected as an official Federal Information Processing Standard (FIPS) for the United States in 1976 and which has subsequently enjoyed widespread use internationally. DES consequently came under intense academic scrutiny which motivated the modern understanding of block ciphers and their cryptanalysis.

DES is the archetypal block cipher, an algorithm that takes a fixed-length string of plaintext bits and transforms it through a series of complicated operations into another ciphertext bit string of the same length. In the case of DES, the block size is 64 bits. DES also uses a key to customize the transformation, so that decryption can supposedly only be performed by those who know the particular key used to encrypt. The key ostensibly consists of 64 bits; however, only 56 of these are actually used by the algorithm. Eight bits are used solely for checking parity, and are thereafter discarded. Hence the effective key length is 56 bits.

Like other block ciphers, DES by itself is not a secure means of encryption but must instead be used in a mode of operation.

Overall structure

The overall scheme for DES encryption is illustrated in Fig 1.3.2. As with any encryption scheme, there are two inputs to the encryption function: the plaintext to be encrypted and the key. In this case, the plaintext must be 64 bits in length and the key is 56 bits in length.

Looking at the left-hand side of the figure, we can see that the processing of the plaintext proceeds in three phases. First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the permuted input. This is followed by a phase consisting of 16 rounds of the same function, which involves both permutation and substitution functions. The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the preoutput. Finally, the preoutput is passed through a permutation (IP^{-1}) that is the inverse of the initial permutation function, to produce the 64-bit ciphertext. With the

exception of the initial and final permutations, DES has the exact structure of a Feistel cipher

The right-hand portion of Fig 1.3.2 shows the way in which the 56-bit key is used. Initially, the key is passed through a permutation function. Then, for each of the 16 rounds, a subkey (K_i) is produced by the combination of a left circular shift and a permutation. The permutation function is the same for each round, but a different subkey is produced because of the repeated shifts of the key bits.

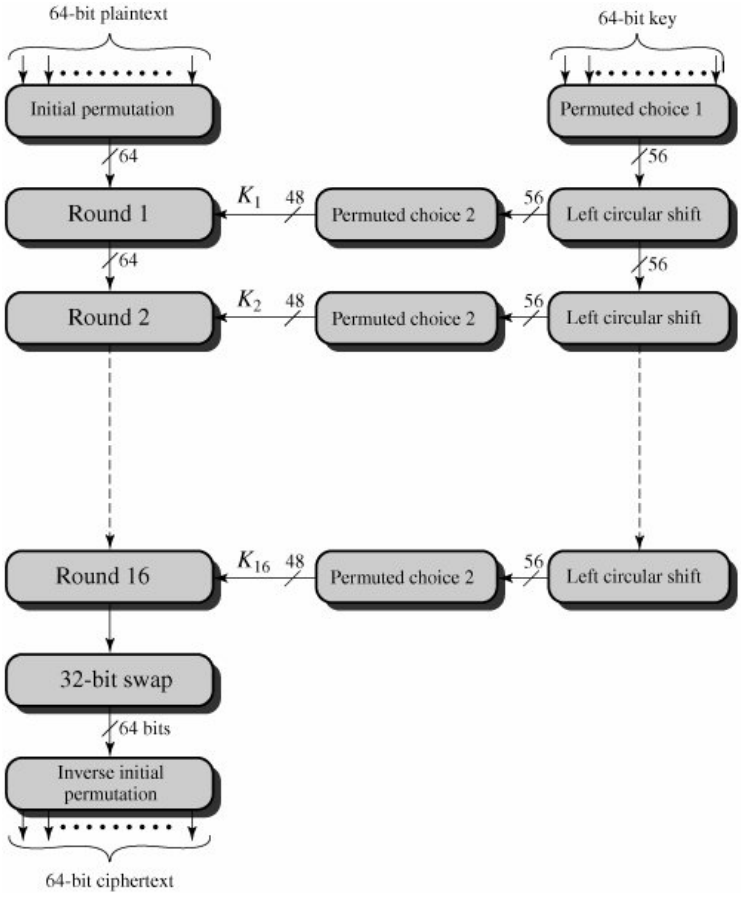


Fig 1.3.2 General Depiction of DES Encryption Algorithm

Fig 1.3.3 shows the internal structure of a single round. Again, begin by focusing on the left-hand side of the diagram. The left and right halves of each 64-bit intermediate value are treated as separate 32-bit quantities, labeled L (left) and R (right). As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulas:

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$

where $\oplus$ denotes the bitwise XOR function.

The round key K_i is 48 bits. The R input is 32 bits. This R input is first expanded to 48 bits by using a table that defines a permutation plus an expansion that involves duplication of 16 of the R bits. The resulting 48 bits are XORed with K_i. This 48-bit result passes through a substitution function that produces a 32-bit output.

The role of the S-boxes in the function F is illustrated in Fig 1.3.4. The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output. These transformations are defined in Table 1.3.1, which is interpreted as follows:

- The first and last bits of the input to box S_i form a 2-bit binary number to select one of four substitutions defined by the four rows in the table for S_i. The middle four bits select one of the sixteen columns. The decimal value in the cell selected by the row and column is then converted to its 4-bit representation to produce the output. For example, in S_1 for input 011001, the row is 01 (row 1) and the column is 1100 (column 12). The value in row 1, column 12 is 9, so the output is 1001.
- Each row of an S-box defines a general reversible substitution.

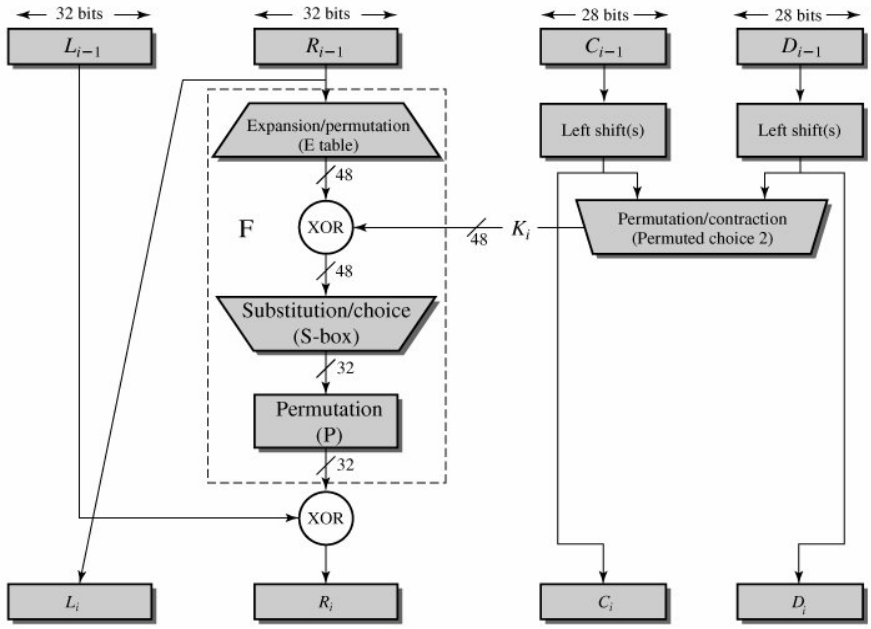


Fig 1.3.3 Single Round of DES Algorithm

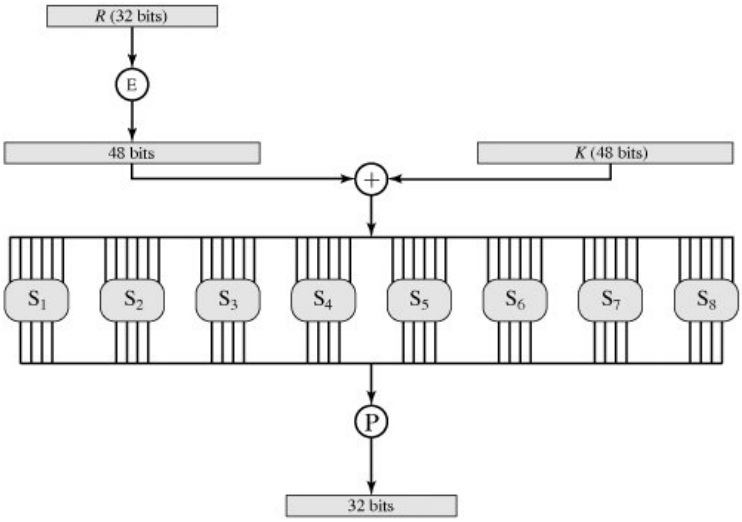


Fig 1.3.4 Calculation of F (R, K)

S_1	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Table 1.3.1 Definition of DES S-Boxes

A 64-bit key is used as input to the algorithm. The bits of the key are numbered from 1 through 64. Every eighth bit is ignored. The key is first subjected to a permutation governed by a Permuted Choice. The resulting 56-bit key is then treated as two 28-bit quantities, labeled C_0 and D_0. At each round, C_{i-1} and D_{i-1} are separately subjected to a circular left shift, or rotation, of 1 or 2 bits. These shifted values serve as input to the next round. They also serve as input to Permuted Choice to the next round which produces a 48-bit output that serves as input to the function $F(R_{i-1}, K_i)$.

DES DECRYPTION

As with any Feistel cipher, decryption uses the same algorithm as encryption, except that the application of the subkeys is reversed.

AVALANCHE EFFECT

A desirable property of any encryption algorithm is that a small change in either the plaintext or the key should produce a significant change in the ciphertext. In particular, a change in one bit of the plaintext or one bit of the key should produce a change in many bits of the ciphertext.

(a) Change in Plaintext		(b) Change in Key	
Round	**Number of bits that differ**	**Round**	**Number of bits that differ**
0	1	0	0
1	6	1	2
2	21	2	14
3	35	3	28
4	39	4	32
5	34	5	30
6	32	6	32
7	31	7	35
8	29	8	34
9	42	9	40
10	44	10	38

11	32	11	31
12	30	12	33
13	30	13	28
14	26	14	26
15	29	15	34
16	34	16	35

Table1.3.2 The Avalanche Effect

DES exhibits a strong avalanche effect. In Table 1.3.2a, two plaintexts that differ by one bit were used. The Table 1.3.2a shows that after just three rounds, 21 bits differ between the two blocks. On completion, the two ciphertexts differ in 34 bit positions.

Table 1.3.2b shows a similar test in which a single plaintext is input with two keys that differ in only one bit position. Again, the results show that about half of the bits in the ciphertext differ and that the avalanche effect is pronounced after just a few rounds.

STRENGTH OF DES

Although more information has been published on the cryptanalysis of DES than any other block cipher, the most practical attack to date is still a brute force approach. Various minor cryptanalytic properties are known, and three theoretical attacks are possible which, while having a theoretical complexity less than a brute force attack, require an unrealistic amount of known or chosen plaintext to carry out, and are not concerns in practice. Since its adoption as a federal standard, there have been lingering concerns about the level of security provided by DES. These concerns, by and large, fall into two areas: key size and the nature of the algorithm.

The Use of 56-Bit Keys

With a key length of 56 bits, there are 2^{56} possible keys, which is approximately 7.2×10^{16}. Thus, on the face of it, a brute-force attack appears impractical. Assuming that, on average, half the key space has to be searched, a single machine performing one DES encryption per microsecond would take more than a thousand years to break the cipher.

However, the assumption of one encryption per microsecond is overly conservative. As far back as 1977, Diffie and Hellman postulated that the technology existed to build a parallel machine with 1 million encryption

devices, each of which could perform one encryption per microsecond. This would bring the average search time down to about 10 hours.

DES finally and definitively proved insecure in July 1998, when the Electronic Frontier Foundation (EFF) announced that it had broken a DES encryption using a special-purpose "DES cracker" machine that was built for less than $250,000. The attack took less than three days. The EFF has published a detailed description of the machine, enabling others to build their own cracker. And hardware prices will continue to drop as speeds increase, making DES virtually worthless.

It is important to note that there is more to a key-search attack than simply running through all possible keys. Unless known plaintext is provided, the analyst must be able to recognize plaintext as plaintext. If the message is just plain text in English, then the result pops out easily, although the task of recognizing English would have to be automated. If the text message has been compressed before encryption, then recognition is more difficult. And if the message is some more general type of data, such as a numerical file, and this has been compressed, the problem becomes even more difficult to automate. Thus, to supplement the brute-force approach, some degree of knowledge about the expected plaintext is needed, and some means of automatically distinguishing plaintext from garble is also needed. The EFF approach addresses this issue as well and introduces some automated techniques that would be effective in many contexts.

The Nature of the DES Algorithm

Another concern is the possibility that cryptanalysis is possible by exploiting the characteristics of the DES algorithm. The focus of concern has been on the eight substitution tables, or S-boxes, that are used in each iteration. Because the design criteria for these boxes were not made public, there is a suspicion that the boxes were constructed in such a way that cryptanalysis is possible for an opponent who knows the weaknesses in the S-boxes. This assertion is tantalizing, and over the years a number of regularities and unexpected behaviors of the S-boxes have been discovered. Despite this, no one has so far succeeded in discovering the supposed fatal weaknesses in the S-boxes.

Timing Attacks

A timing attack is an example of an attack that exploits the implementation of an algorithm rather than the algorithm itself. The same algorithm can always be re-implemented in a way that leaks little or no information to a timing attack: consider an implementation in which every call to a subroutine always returns in exactly x seconds, where x is the

maximum time it ever takes to execute the routine. In such an implementation, timing gives an attacker no useful information; it also has the adverse effect of slower response times on average.

The practicality of the attack implies several things:

- It is algorithm-independent. Notice that the theoretical security of such algorithms remains — it is mainly the need for a high-speed implementation of a particular algorithm that introduces such vulnerabilities.
- Finding timing information is more routine. While finding cryptographic errors in a crypto-primitive such as the DES may require deeper knowledge of mathematics, timing is relatively easy. And measuring response time for a specific query might give away relatively large amounts of information.

A timing attack is one in which information about the key or the plaintext is obtained by observing how long it takes a given implementation to perform decryptions on various ciphertexts. A timing attack exploits the fact that an encryption or decryption algorithm often takes slightly different amounts of time on different inputs. DES appears to be fairly resistant to a successful timing attack but suggest there are some avenues to explore. Although this is an interesting line of attack, it so far appears unlikely that this technique will ever be successful against DES or more powerful symmetric ciphers such as triple DES and AES.

DES Design Criteria

The criteria used in the design of DES focused on the design of the S-boxes and on the P function that takes the output of the S boxes (Figure 1.3.4). The criteria for the S-boxes are as follows:

1. No output bit of any S-box should be too close a linear function of the input bits. Specifically, if we select any output bit and any subset of the six input bits, the fraction of inputs for which this output bit equals the XOR of these input bits should not be close to 0 or 1, but rather should be near 1/2.
2. Each row of an S-box (determined by a fixed value of the leftmost and rightmost input bits) should include all 16 possible output bit combinations.
3. If two inputs to an S-box differ in exactly one bit, the outputs must differ in at least two bits.
4. If two inputs to an S-box differ in the two middle bits exactly, the outputs must differ in at least two bits.

5. If two inputs to an S-box differ in their first two bits and are identical in their last two bits, the two outputs must not be the same.
6. For any nonzero 6-bit difference between inputs, no more than 8 of the 32 pairs of inputs exhibiting that difference may result in the same output difference.
7. This is a criterion similar to the previous one, but for the case of three S-boxes.

The first criterion in the preceding list was needed because the S-boxes are the only nonlinear part of DES. If the S-boxes were linear (i.e., each output bit is a linear combination of the input bits), the entire algorithm would be linear and easily broken. We have seen this phenomenon with the Hill cipher, which is linear. The remaining criteria were primarily aimed at thwarting differential cryptanalysis and at providing good confusion properties.

The criteria for the permutation P are as follows:

1. The four output bits from each S-box at round i are distributed so that two of them affect (provide input for) "middle bits" of round (i + 1) and the other two affect end bits. The two middle bits of input to an S-box are not shared with adjacent S-boxes. The end bits are the two left-hand bits and the two right-hand bits, which are shared with adjacent S-boxes.
2. The four output bits from each S-box affect six different S-boxes on the next round, and no two affect the same S-box.
3. For two S-boxes j, k, if an output bit from S_j affects a middle bit of S_k on the next round, then an output bit from S_k cannot affect a middle bit of S_j. This implies that for j = k, an output bit from S_j must not affect a middle bit of S_j.

   These criteria are intended to increase the diffusion of the algorithm.

Number of Rounds

The cryptographic strength of a Feistel cipher derives from three aspects of the design

- the number of rounds
- the function F
- the key schedule algorithm

The greater the number of rounds, the more difficult it is to perform cryptanalysis, even for a relatively weak F. In general, the criterion should be that the number of rounds is chosen so that known cryptanalytic

efforts require greater effort than a simple brute-force key search attack. This criterion was certainly used in the design of DES.

Design of Function F

The heart of a Feistel block cipher is the function F. In DES, this function relies on the use of S-boxes. This is also the case for most other symmetric block ciphers. However, we can make some general comments about the criteria for designing F. After that, we look specifically at S-box design.

Design Criteria for F

The function F provides the element of confusion in a Feistel cipher. Thus, it must be difficult to "unscramble" the substitution performed by F. One obvious criterion is that F be nonlinear. The more nonlinear F, the more difficult any type of cryptanalysis will be. In rough terms, the more difficult it is to approximate F by a set of linear equations, the more nonlinear F is. Another criterion proposed is the bit independence criterion (BIC), which states that output bits j and k should change independently when any single input bit i is inverted, for all i, j, and k.

S-Box Design

One of the most intense areas of research in the field of symmetric block ciphers is that of S-box design. In essence, we would like any change to the input vector to an S-box to result in random-looking changes to the output. The relationship should be nonlinear and difficult to approximate with linear functions.

One obvious characteristic of the S-box is its size. An n x m S-box has n input bits and m output bits. DES has 6 x 4 S-boxes. Larger S-boxes are more resistant to all types of cryptanalysis. On the other hand, the larger the dimension n, the (exponentially) larger the lookup table. Thus, for practical reasons, a limit of n equal to about 8 to 10 is usually imposed. Another practical consideration is that the larger the S-box, the more difficult it is to design it properly.

S-boxes are typically organized in a different manner than used in DES. An n x m S-box typically consists of 2^n rows of m bits each. The n bits of input select one of the rows of the S-box, and the m bits in that row are the output. For example, in an 8 x 32 S-box, if the input is 00001001, the output consists of the 32 bits in row 9 (the first row is labeled row 0).

Nyberg, who has written a lot about the theory and practice of S-box design, suggests the following approaches

- Random: Use some pseudorandom number generation or some table of random digits to generate the entries in the S-boxes. This may lead to boxes with undesirable characteristics for small sizes

(e.g., 6 x 4) but should be acceptable for large S-boxes (e.g., 8 x 32).

- Random with testing: Choose S-box entries randomly, then test the results against various criteria, and throw away those that do not pass.
- Human-made: This is a more or less manual approach with only simple mathematics to support it. It is apparently the technique used in the DES design. This approach is difficult to carry through for large S-boxes.
- Math-made: Generate S-boxes according to mathematical principles. By using mathematical construction, S-boxes can be constructed that offer proven security against linear and differential cryptanalysis, together with good diffusion.

A variation on the first technique is to use S-boxes that are both random and key dependent. A tremendous advantage of key-dependent S-boxes is that, because they are not fixed, it is impossible to analyze the S-boxes ahead of time to look for weaknesses.

Key Schedule Algorithm

A final area of block cipher design, and one that has received less attention than S-box design, is the key schedule algorithm. With any Feistel block cipher, the key is used to generate one subkey for each round. In general, we would like to select subkeys to maximize the difficulty of deducing individual subkeys and the difficulty of working back to the main key. At minimum, the key schedule should guarantee key/ciphertext Strict Avalanche Criterion and Bit Independence Criterion.

Block Cipher Modes of Operation

In cryptography, a block cipher operates on blocks of fixed length, often 64 or 128 bits. Because messages may be of any length, and because encrypting the same plaintext under the same key always produces the same output (as described in the ECB section below), several **modes of operation** have been invented which allow block ciphers to provide confidentiality for messages of arbitrary length. All these modes (except ECB) require an initialization vector, or *IV* -- a sort of 'dummy block' to kick off the process for the first real block, and also to provide some randomization for the process. There is no need for the IV to be secret, in most cases, but it is important that it is never reused with the same key. For CBC and CFB, reusing an IV leaks some information about the first block of plaintext, and about any common prefix shared by the two messages. For OFB and CTR, reusing an IV completely destroys

security. In CBC mode, the IV must, in addition, be randomly generated at encryption time.

Electronic Codebook Mode (ECB)

The simplest mode is the electronic codebook (ECB) mode, in which plaintext is handled one block at a time and each block of plaintext is encrypted using the same key (Figure 1.3.5). The term codebook is used because, for a given key, there is a unique ciphertext for every b-bit block of plaintext. Therefore, we can imagine a gigantic codebook in which there is an entry for every possible b-bit plaintext pattern showing its corresponding ciphertext.

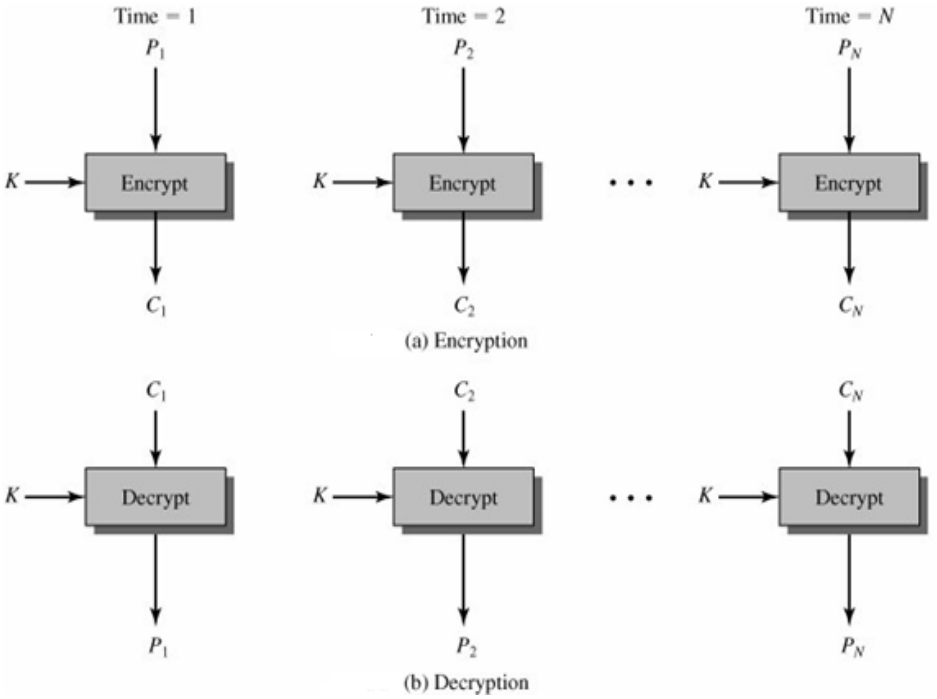


Fig 1.3.5 Electronic Code Book Mode

For a message longer than b bits, the procedure is simply to break the message into b-bit blocks, padding the last block if necessary. Decryption is performed one block at a time, always using the same key. In Fig 1.3.5, the plaintext (padded as necessary) consists of a sequence of b-bit blocks, P_1, P_2,..., P_N; the corresponding sequence of ciphertext blocks is C_1, C_2,..., C_N.

The ECB method is ideal for a short amount of data, such as an encryption key. Thus, if you want to transmit a DES key securely, ECB is the appropriate mode to use.

The disadvantage of this method is that identical plaintext blocks are encrypted into identical ciphertext blocks; thus, it does not hide data patterns well. In some senses, it doesn't provide serious message confidentiality, and it is not recommended for use in cryptographic protocols at all.

Cipher Block Chaining Mode (CBC)

To overcome the security deficiencies of ECB, we would like a technique in which the same plaintext block, if repeated, produces different ciphertext blocks. A simple way to satisfy this requirement is the cipher block chaining (CBC) mode (Fig 1.3.6). In this scheme, the input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block; the same key is used for each block. In effect, we have chained together the processing of the sequence of plaintext blocks. The input to the encryption function for each plaintext block bears no fixed relationship to the plaintext block. Therefore, repeating patterns of b bits are not exposed.

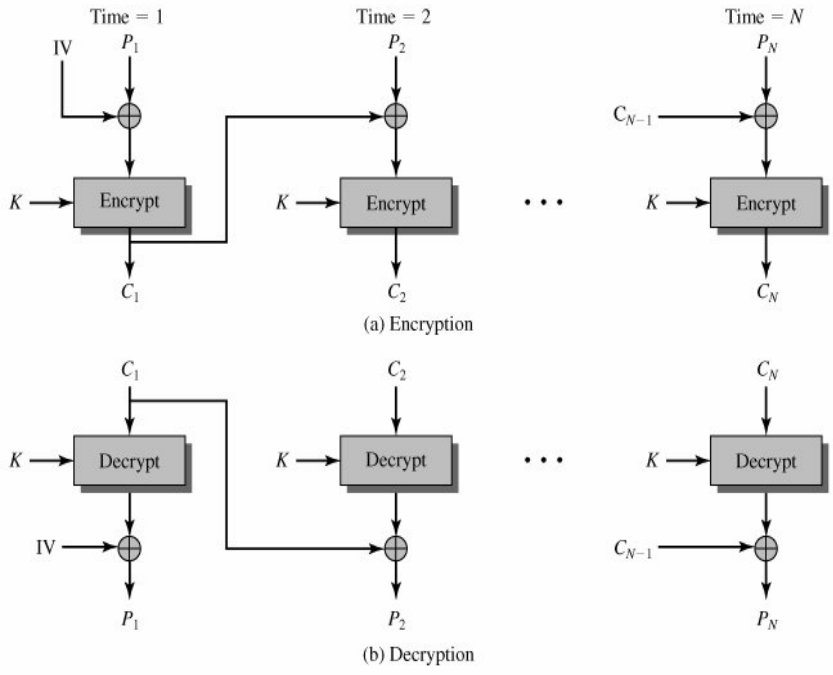


Fig 1.3.6 Cipher block Chaining Mode

For decryption, each cipher block is passed through the decryption algorithm. The result is XORed with the preceding ciphertext block to produce the plaintext block. To see that this works, we can write

$$C_j = E(K, [C_{j-1} \oplus P_j])$$

Then

$$D(K, C_j) = D(K, E(K, [C_{j-1} \oplus P_j]))$$

$$D(K, C_j) = C_{j-1} \oplus P_j$$

$$C_{j-1} \oplus D(K, C_j) = C_{j-1} \oplus C_{j-1} \oplus P_j = P_j$$

To produce the first block of ciphertext, an initialization vector (IV) is XORed with the first block of plaintext. On decryption, the IV is XORed with the output of the decryption algorithm to recover the first block of plaintext. The IV is a data block that is that same size as the cipher block.

The IV must be known to both the sender and receiver but be unpredictable by a third party. For maximum security, the IV should be protected against unauthorized changes. This could be done by sending the IV using ECB encryption. One reason for protecting the IV is as follows: If an opponent is able to fool the receiver into using a different value for IV, then the opponent is able to invert selected bits in the first block of plaintext. To see this, consider the following:

$$C_1 = E(K, [IV \oplus P_1])$$

$$P_1 = IV \oplus D(K, C_1)$$

Now use the notation that X[i] denotes the i^{th} bit of the b-bit quantity X. Then

$$P_1[i] = IV[i] \oplus D(K, C_1)[i]$$

Then, using the properties of XOR, we can state

$$P_1[i]' = IV[i]' \oplus D(K, C_1)[i]$$

where the prime notation denotes bit complementation.

This means that if an opponent can predictably change bits in IV, the corresponding bits of the received value of P_1 can be changed. In conclusion, because of the chaining mechanism of CBC, it is an appropriate mode for encrypting messages of length greater than b bits.

Cipher Feedback Mode (CFB)

The DES scheme is essentially a block cipher technique that uses b-bit blocks. However, it is possible to convert DES into a stream cipher, using either the cipher feedback (CFB) or the output feedback mode. A stream cipher eliminates the need to pad a message to be an integral number of blocks. It also can operate in real time. Thus, if a character stream is being transmitted, each character can be encrypted and transmitted immediately using a character-oriented stream cipher.

One desirable property of a stream cipher is that the ciphertext be of the same length as the plaintext. Thus, if 8-bit characters are being transmitted, each character should be encrypted to produce a cipher text

output of 8 bits. If more than 8 bits are produced, transmission capacity is wasted.

Figure 1.3.7 depicts the CFB scheme. In the figure, it is assumed that the unit of transmission is s bits; a common value is s = 8. As with CBC, the units of plaintext are chained together, so that the ciphertext of any plaintext unit is a function of all the preceding plaintext. In this case, rather than units of b bits, the plaintext is divided into segments of s bits.

Consider encryption. The input to the encryption function is a b-bit shift register that is initially set to some initialization vector (IV). The leftmost (most significant) s bits of the output of the encryption function are XORed with the first segment of plaintext P_1 to produce the first unit of ciphertext C_1, which is then transmitted. In addition, the contents of the shift register are shifted left by s bits and C_1 is placed in the rightmost (least significant) s bits of the shift register. This process continues until all plaintext units have been encrypted.

For decryption, the same scheme is used, except that the received ciphertext unit is XORed with the output of the encryption function to produce the plaintext unit. Note that it is the *encryption* function that is used, not the decryption function. This is easily explained. Let $S_s(X)$ be defined as the most significant s bits of X. Then

$$\mathbf{C_1 = P_1 \oplus S_s[\, E(K,\ IV)\ ]}$$

Therefore,

$$\mathbf{P_1 = C_1 \oplus S_s[E(K, IV)]}$$

The same reasoning holds for subsequent steps in the process.

Output Feedback Mode (OFB)

The output feedback (OFB) mode is similar in structure to that of CFB, as illustrated in **Fig 1.3.8**. It is the output of the encryption function that is fed back to the shift register in OFB, whereas in CFB the ciphertext unit is fed back to the shift register.

One advantage of the OFB method is that bit errors in transmission do not propagate. For example, if a bit error occurs in C_1 only the recovered value of is P_1 affected; subsequent plaintext units are not corrupted. With CFB, C_1 also serves as input to the shift register and therefore causes additional corruption downstream.

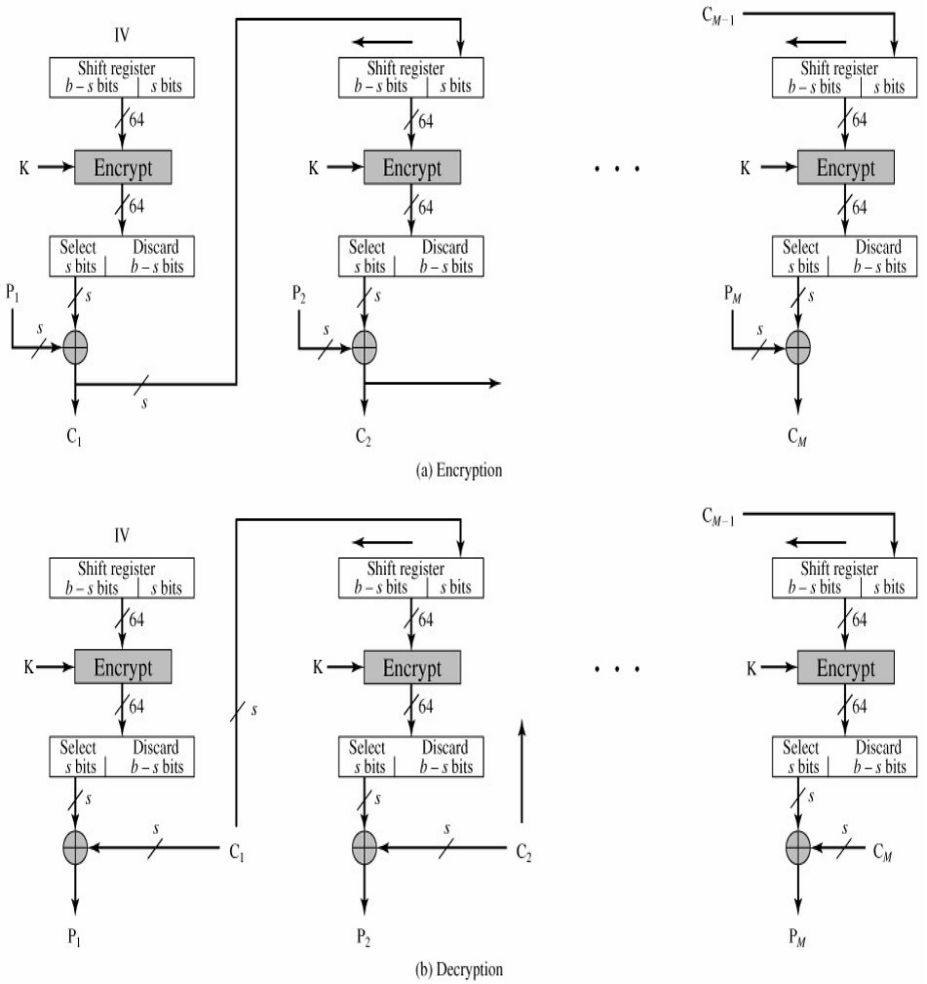


Fig 1.3.7 Cipher Feedback Mode

The disadvantage of OFB is that it is more vulnerable to a message stream modification attack than is CFB. Consider that complementing a bit in the ciphertext complements the corresponding bit in the recovered plaintext. Thus, controlled changes to the recovered plaintext can be made. This may make it possible for an opponent, by making the necessary changes to the checksum portion of the message as well as to the data portion, to alter the ciphertext in such a way that it is not detected by an error-correcting code.

Counter Mode (CTR)

Fig 1.3.8 depicts the CTR mode. Like OFB, counter mode turns a block cipher into a stream cipher. It generates the next keystream block by encrypting successive values of a "counter". The counter can be any

simple function which produces a sequence which is guaranteed not to repeat for a long time, although an actual counter is the simplest and most popular. CTR mode has similar characteristics to OFB, but also allows a random access property during decryption. CTR mode is well suited to operation on a multi-processor machine where blocks can be encrypted in parallel. A counter, equal to the plaintext block size is used. The only requirement

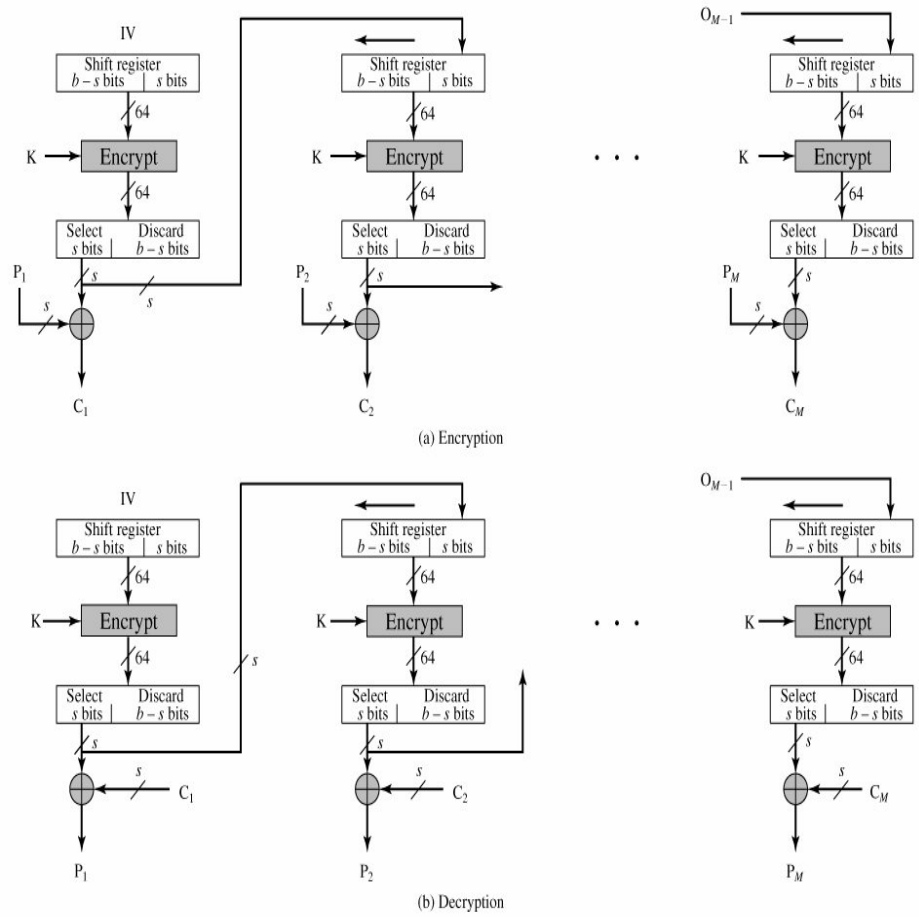


Fig 1.3.8 Output Feedback Mode

is that the counter value must be different for each plaintext block that is encrypted. Typically, the counter is initialized to some value and then incremented by 1 for each subsequent block (modulo 2^b where b is the block size). For encryption, the counter is encrypted and then XORed with the plaintext block to produce the ciphertext block; there is no chaining. For decryption, the same sequence of counter values is used, with each encrypted counter XORed with a ciphertext block to recover the corresponding plaintext block.

Advantages of CTR mode:

- **Hardware efficiency:** Unlike the three chaining modes, encryption (or decryption) in CTR mode can be done in parallel on multiple blocks of plaintext or ciphertext. For the chaining modes, the algorithm must complete the computation on one block before beginning on the next block. This limits the maximum throughput of the algorithm to the reciprocal of the time for one execution of block encryption or decryption. In CTR mode, the throughput is only limited by the amount of parallelism that is achieved.
- **Software efficiency:** Similarly, because of the opportunities for parallel execution in CTR mode, processors that support parallel features, such as aggressive pipelining, multiple instruction dispatch per clock cycle, a large number of registers, and SIMD instructions, can be effectively utilized.
- **Preprocessing:** The execution of the underlying encryption algorithm does not depend on input of the plaintext or ciphertext. Therefore, if sufficient memory is available and security is maintained, preprocessing can be used to prepare the output of the encryption boxes that feed into the XOR functions in Figure 1.3.9. When the plaintext or ciphertext input is presented, then the only computation is a series of XORs. Such a strategy greatly enhances throughput.

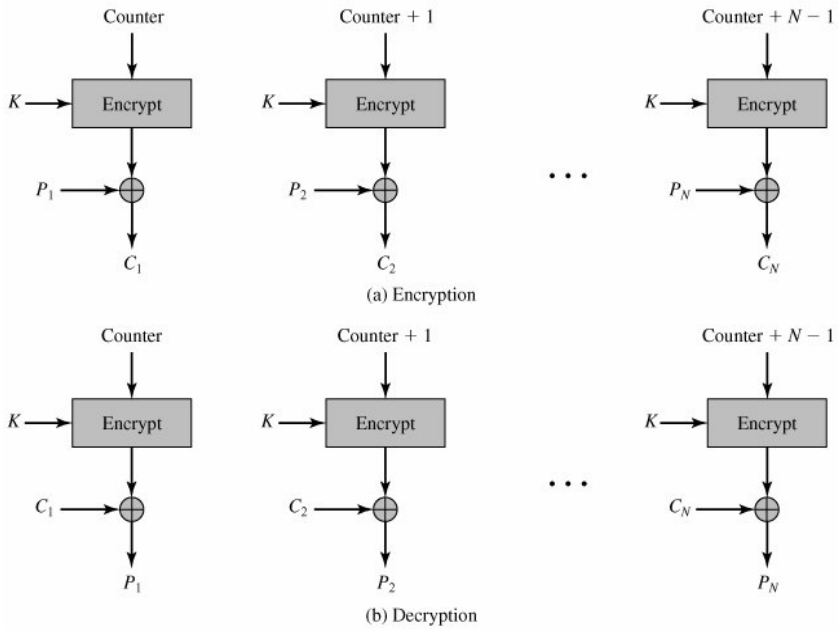


Fig 1.3.9 Counter Mode

- **Random access:** The i^{th} block of plaintext or ciphertext can be processed in random-access fashion. With the chaining modes, block C_i cannot be computed until the i - 1 prior block are computed. There may be applications in which a ciphertext is stored and it is desired to decrypt just one block; for such applications, the random access feature is attractive.
- **Provable security:** It can be shown that CTR is at least as secure as the other modes discussed in this section.
- **Simplicity:** Unlike ECB and CBC modes, CTR mode requires only the implementation of the encryption algorithm and not the decryption algorithm. This matters most when the decryption algorithm differs substantially from the encryption algorithm, as it does for AES. In addition, the decryption key scheduling need not be implemented.

Multiple Encryption and Triple DES

Given the potential vulnerability of DES to a brute-force attack, there has been considerable interest in finding an alternative. AES is a prime example. Another alternative, which would preserve the existing investment in software and equipment, is to use multiple encryption with DES and multiple keys. We begin by examining the simplest example of this second alternative.

Double DES

The simplest form of multiple encryption has two encryption stages and two keys (Fig 1.3.10). Given a plaintext P and two encryption keys K_1 and K_2, ciphertext C is generated as

$$C = E(K_2, E(K_1, P))$$

Decryption requires that the keys be applied in reverse order:

$$\mathbf{P = D\ (K_1,\ D\ (K_2,\ C))}$$

For DES, this scheme apparently involves a key length of 56 x 2 = 112 bits, of resulting in a dramatic increase in cryptographic strength. But we need to examine the algorithm more closely.

Reduction to a Single Stage

Suppose it were true for DES, for all 56-bit key values, that given any two keys K_1 and K_2, it would be possible to find a key K_3 such that

$$\mathbf{E(K_2,\ E(K_1,\ P)) = E\ (K_3, P)} \qquad \text{----- (a)}$$

If this were the case, then double encryption, and indeed any number of stages of multiple encryption with DES, would be useless because the result would be equivalent to a single encryption with a single 56-bit key.

On the face of it, it does not appear that Equation (a) is likely to hold. Consider that encryption with DES is a mapping of 64-bit blocks to 64-bit

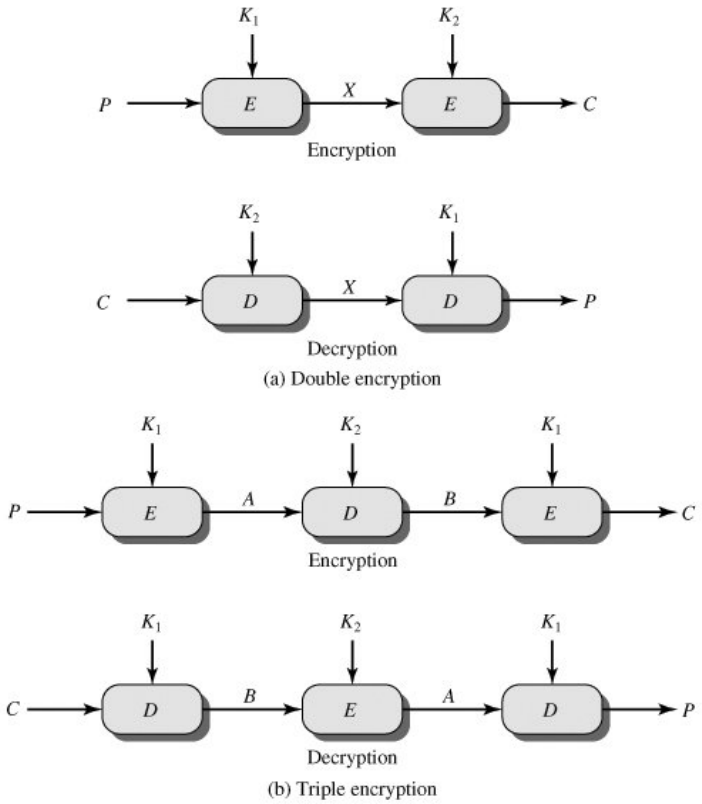


Fig 1.3.10 Multiple Encryption

blocks. In fact, the mapping can be viewed as a permutation. That is, if we consider all 2^{64} possible input blocks, DES encryption with a specific key will map each block into a unique 64-bit block. Otherwise, if, say, two given input blocks mapped to the same output block, then decryption to recover the original plaintext would be impossible. With 2^{64} possible inputs, how many different mappings are there those generate a permutation of the input blocks? The value is easily seen to be

$$(2^{64})! = 10^{34738000000000000000} > (10^{10^{20}})$$

On the other hand, DES defines one mapping for each different key, for a total number of mappings:

$$2^{56} > 10^{17}$$

Therefore, it is reasonable to assume that if DES is used twice with different keys, it will produce one of the many mappings that are not defined by a single application of DES.

Meet-in-the-Middle Attack

The use of double DES results in a mapping that is not equivalent to a single DES encryption. But there is a way to attack this scheme, one that does not depend on any particular property of DES but that will work against any block encryption cipher.

The meet-in-the-middle attack algorithm is based on the observation that, if we have

$$\mathbf{C = E(K_2, E(K_1, P))}$$

Then (see Figure 1.3.10a)

$$\mathbf{X = E(K_1, P) = D(K_2, P)}$$

Given a known pair, (P, C), the attack proceeds as follows. First, encrypt P for all 2^{56} possible values of K_1 Store these results in a table and then sort the table by the values of X. Next, decrypt C using all 2^{56} possible values of K_2. As each decryption is produced, check the result against the table for a match. If a match occurs, then test the two resulting keys against a new known plaintext-ciphertext pair. If the two keys produce the correct ciphertext, accept them as the correct keys.

For any given plaintext P, there are 2^{64} possible ciphertext values that could be produced by double DES. Double DES uses, in effect, a 112-bit key, so that there are 2^{112} possible keys. Therefore, on average, for a given plaintext P, the number of different 112-bit keys that will produce a given ciphertext C is $2^{112}/2^{64} = 2^{48}$. Thus, the foregoing procedure will produce about 2^{48} false alarms on the first (P, C) pair. A similar argument indicates that with an additional 64 bits of known plaintext and ciphertext, the false alarm rate is reduced to $2^{48-64} = 2^{-16}$ Put another way, if the meet-in-the-middle attack is performed on two blocks of known plaintext-ciphertext, the probability that the correct keys are determined is 1 2^{-16}. The result is that a known plaintext attack will succeed against double DES, which has a key size of 112 bits, with an effort on the order of 2^{56}, not much more than the 2^{55} required for single DES.

Triple DES with Two Keys

An obvious counter to the meet-in-the-middle attack is to use three stages of encryption with three different keys. This raises the cost of the known-plaintext attack to 2^{112}, which is beyond what is practical now and far into the future. However, it has the drawback of requiring a key length of 56 x 3 = 168 bits.

As an alternative, Tuchman proposed a triple encryption method that uses only two keys. The function follows an encrypt-decrypt-encrypt (EDE) sequence (Figure 1.3.10b):

$$C = E(K_1, D(K_2, E(K_1, P)))$$

There is no cryptographic significance to the use of decryption for the second stage. Its only advantage is that it allows users of 3DES to decrypt data encrypted by users of the older single DES:

$$C = E(K_1, D(K_1, E(K_1, P))) = E(K_1, P)$$

- 3DES with two keys is a relatively popular alternative to DES.

Triple DES with Three Keys

Although the attacks just described appear impractical, anyone using two-key 3DES may feel some concern. Thus, many researchers now feel that three-key 3DES is the preferred alternative. Three-key 3DES has an effective key length of 168 bits and is defined as follows:

$$C = E(K_3, D(K_2, E(K_1, P)))$$

Backward compatibility with DES is provided by putting $K_3 = K_2$ or $K_1 = K_2$.

Mode	Description	Typical Application
Electronic Codebook (ECB)	Each block of 64 plaintext bits is encoded independently using the same key.	• Secure transmission of single values (e.g., an encryption key)
Cipher Block Chaining (CBC)	The input to the encryption algorithm is the XOR of the next 64 bits of plaintext and the preceding 64 bits of ciphertext.	• General-purpose block-oriented transmission • Authentication
Cipher Feedback (CFB)	Input is processed j bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is XORed with plaintext to produce next unit of ciphertext.	• General-purpose stream-oriented transmission • Authentication
Output Feedback (OFB)	Similar to CFB, except that the input to the encryption algorithm is the preceding DES output.	• Stream-oriented transmission over noisy channel (e.g., satellite communication)
Counter (CTR)	Each block of plaintext is XORed with an encrypted counter. The counter is incremented for each subsequent block.	• General-purpose block-oriented transmission • Useful for high-speed requirements

Table 1.3.3 Block Cipher Modes of Operation

Cipher	Key Length	Speed (Mbps)
DES	56	9
3DES	168	3
RC2	variable	0.9
RC4	variable	45

Table 1.3.4 Speed Comparisons of Symmetric Ciphers on a Pentium II

REFERENCES

1. William Stallings, Cryptography and Network Security, PHI Publishers
2. www.wikipedia.org

Review Questions:

1. Write a short notes on Fiestel Network
2. Explain the Des Algorithm in detail.
3. Give an account of DES design criteria
4. Explain the block ciphers modes of operations

4. ADVANCED ENCRYPTION STANDARD

OBJECTIVE

The goal of this lesson is to introduce the Advanced Encryption Standard. This conventional encryption algorithm is designed to replace DES and triple DES. In this lesson, we cover the evaluation criteria and the algorithm issues in detail

ADVANCED ENCRYPTION STANDARD

AES is a symmetric encryption algorithm processing data in block of 128 bits. A bit can take the values zero and one, in effect a binary digit with two possible values as opposed to decimal digits, which can take one of 10 values. Under the influence of a key, a 128-bit block is encrypted by transforming it in a unique way into a new block of the same size. AES is symmetric since the same key is used for encryption and the reverse transformation, decryption. The only secret necessary to keep for security is the key. AES may configured to use different key-lengths, the standard defines 3 lengths and the resulting algorithms are named AES-128, AES-192 and AES-256 respectively to indicate the length in bits of the key. Each additional bit in the key effectively doubles the strength of the algorithm, when defined as the time necessary for an attacker to stage a brute force attack, i.e. an exhaustive search of all possible key combinations in order to find the right one. AES is founded on solid and well-published mathematical ground, and appears to resist all known attacks well.

A strong encryption algorithm need only meet only single main criteria:

- There must be no way to find the unencrypted clear text if the key is unknown, except brute force, i.e. to try all possible keys until the right one is found.

A secondary criterion must also be met:

- The number of possible keys must be so large that it is computationally infeasible to actually stage a successful brute force attack in short enough a time.

The older standard, DES or Data Encryption Standard, meets the first criterion, but no longer the secondary one – computer speeds have

caught up with it, or soon will. AES meets both criteria in all of its variants: AES-128, AES-192 and AES-256.

AES CIPHER

The Rijndael proposal for AES defined a cipher in which the block length and the key length can be independently specified to be 128, 192, or 256 bits. The AES specification uses the same three key size alternatives but limits the block length to 128 bits. A number of AES parameters depend on the key length (Table 1.8). In the description of this section, we assume a key length of 128 bits, which is likely to be the one most commonly implemented.

Key size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext block size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of rounds	10	12	14
Round key size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded key size (words/bytes)	44/176	52/208	60/240

Table 1.4.1 AES Parameters

Rijndael was designed to have the following characteristics:

- Resistance against all known attacks
- Speed and code compactness on a wide range of platforms
- Design simplicity

Fig 1.4.1 shows the overall structure of AES. The input to the encryption and decryption algorithms is a single 128-bit block. In FIPS PUB 197, this block is depicted as a square matrix of bytes. This block is copied into the State array, which is modified at each stage of encryption or decryption. After the final stage, State is copied to an output matrix. These operations are depicted in 1.4.1a. Similarly, the 128-bit key is depicted as a square matrix of bytes. This key is then expanded into an array of key schedule words; each word is four bytes and the total key schedule is 44 words for the 128-bit key (Fig 1.4.1b). Note that the ordering of bytes within a matrix is by column. So, for example, the first four bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the in matrix, the second four bytes occupy the second column, and so on. Similarly, the first four bytes of the expanded key, which form a word, occupy the first column of the **w** matrix.

Before delving into details, we can make several comments about the overall AES structure:

1. One noteworthy feature of this structure is that it is not a Feistel structure. Recall that in the classic Feistel structure, half of the data block is used to modify the other half of the data block, and then the halves are swapped. Two of the AES finalists, including Rijndael, do not use a Feistel structure but process the entire data block in parallel during each round using substitutions and permutation.
2. The key that is provided as input is expanded into an array of forty-four 32-bit words, w[i]. Four distinct words (128 bits) serve as a round key for each round.
3. Four different stages are used, one of permutation and three of substitution:
    - **Substitute bytes:** Uses an S-box to perform a byte-by-byte substitution of the block
    - **Shift Rows :** A simple permutation
    - **Mix Columns :** A substitution that makes use of arithmetic over GF(2^8)
    - **AddRoundKey:** A simple bitwise XOR of the current block with a portion of the expanded key
4. The structure is quite simple. For both encryption and decryption, the cipher begins with an AddRoundKey stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages. Fig 1.4.3 depicts the structure of a full encryption round.
5. Only the AddRoundKey stage makes use of the key. For this reason, the cipher begins and ends with an AddRoundKey stage. Any other stage, applied at the beginning or end, is reversible without knowledge of the key and so would add no security.
6. The AddRoundKey stage is, in effect, a form of Vernam cipher and by itself would not be formidable. The other three stages together provide confusion, diffusion, and nonlinearity, but by themselves would provide no security because they do not use the key. We can view the cipher as alternating operations of XOR encryption (AddRoundKey) of a block, followed by scrambling of the block (the other three stages), followed by XOR encryption, and so on. This scheme is both efficient and highly secure.
7. Each stage is easily reversible. For the Substitute Byte, ShiftRows, and MixColumns stages, an inverse function is used in

the decryption algorithm. For the AddRoundKey stage, the inverse is achieved by XORing the same round key to the block, using the result that A ⊕ A ⊕ B = B.

8. As with most block ciphers, the decryption algorithm makes use of the expanded key in reverse order. However, the decryption algorithm is not identical to the encryption algorithm. This is a consequence of the particular structure of AES.
9. Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext. Fig 1.4.1 lays out encryption and decryption going in opposite vertical directions. At each horizontal point (e.g., the dashed line in the figure), State is the same for both encryption and decryption.
10. The final round of both encryption and decryption consists of only three stages. Again, this is a consequence of the particular structure of AES and is required to make the cipher reversible.

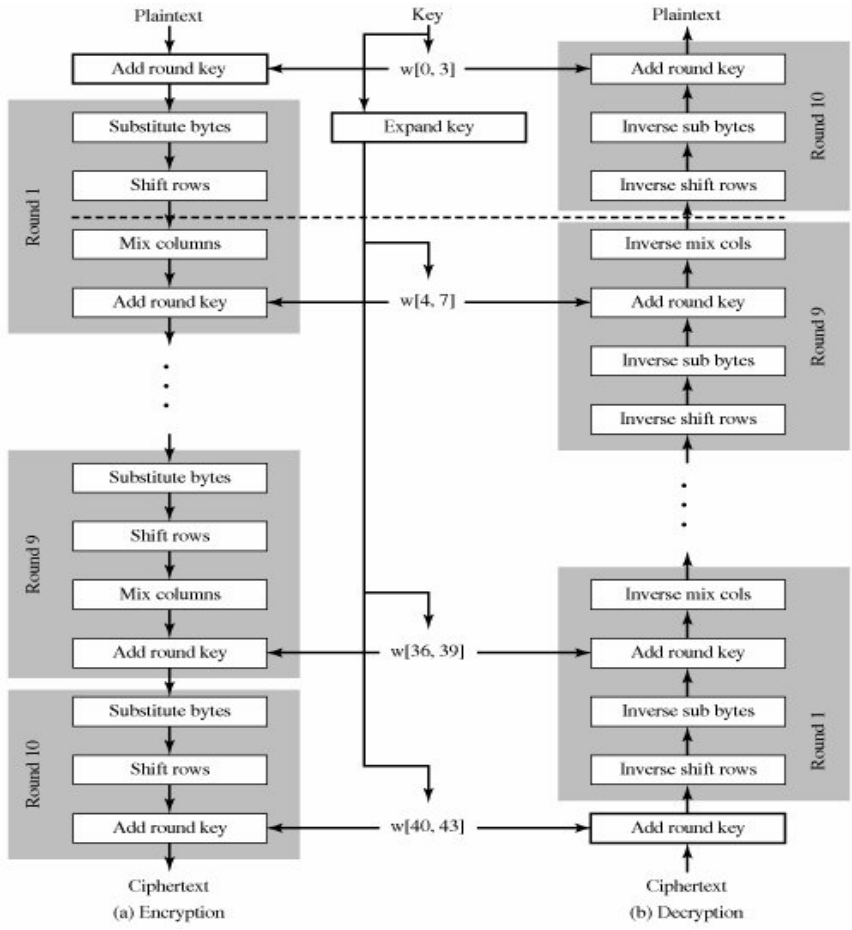


Fig 1.4.1 AES Encryption and Decryption

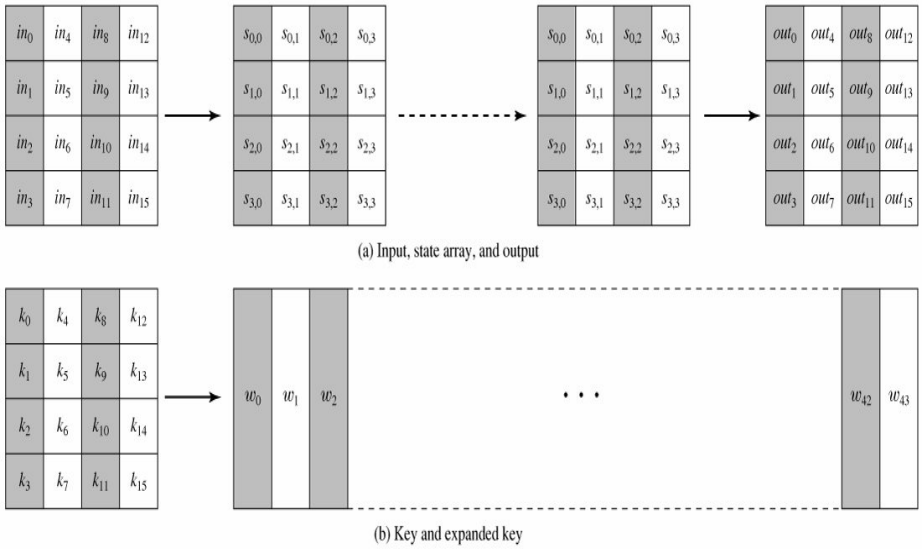


Fig 1.4.2 AES Data Structure

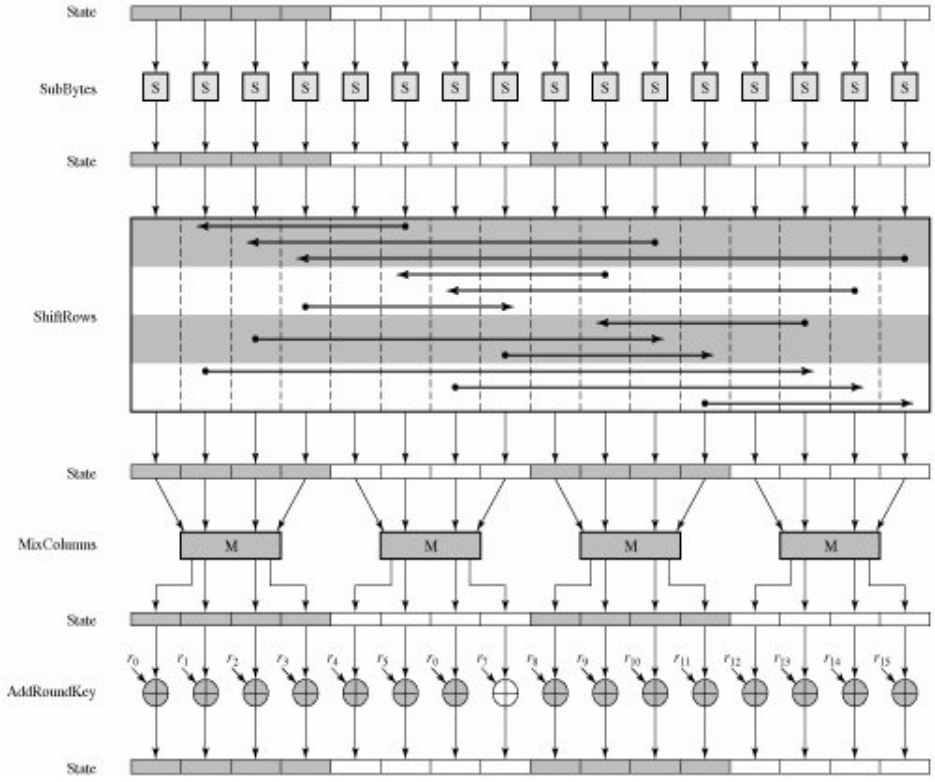


Fig 1.4.3 AES Encryption Round

Let us discuss each of the four stages used in AES. For each stage, we describe the forward (encryption) algorithm, the inverse (decryption) algorithm, and the rationale for the stage. This is followed by a discussion of key expansion.

AES uses arithmetic in the finite field GF(2^8), with the irreducible polynomial

$$m(x) = x^8 + x^4 + x^3 + x + 1.$$

Substitute Bytes Transformation

Forward and Inverse Transformations

The forward substitute byte transformation, called SubBytes, is a simple table lookup (Fig 1.4.4a). AES defines a 16 x 16 matrix of byte values, called an S-box (Table 1.4.2a), that contains a permutation of all possible 256 8-bit values. Each individual byte of State is mapped into a new byte in the following way: The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value. These row and column values serve as indexes into the S-box to select a unique 8-bit output value. For example, the hexadecimal value {95} references row 9, column 5 of the S-box, which contains the value {2A}. Accordingly, the value {95} is mapped into the value {2A}.

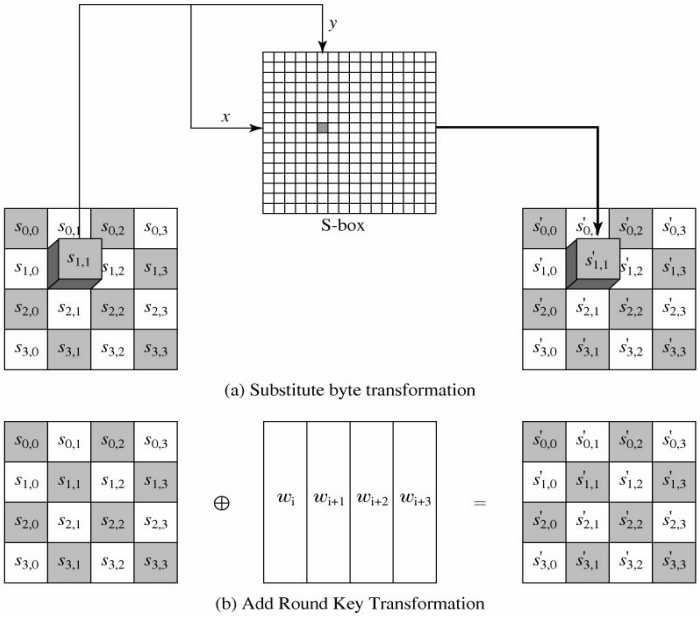


Fig 1.4.4 AES Byte Level Operation

The S-box is constructed in the following fashion:

1. Initialize the S-box with the byte values in ascending sequence row by row. The first row contains {00}, {01}, {02},.... {0F}; the second row contains {10}, {11}, etc.; and so on. Thus, the value of the byte at row x, column y is {xy}.
2. Map each byte in the S-box to its multiplicative inverse in the finite field GF(2^8); the value {00} is mapped to itself.
3. Consider that each byte in the S-box consists of 8 bits labeled (b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0). Apply the following transformation to each bit of each byte in the S-box:

$$b'_i = b_i \oplus b_{(i+1)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus C_i \quad (1.4.1)$$

where c_i is the ith bit of byte c with the value {63}; i.e., (C_7 C_6 C_5 C_4 C_3 C_2 C_1 C_0) = (01100011).

(a) S-box

x \ y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

(b) Inverse S-box

x \ y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Table 1.4.2 AES S-Boxes

Here is an example of the SubBytes transformation:

EA	04	65	85
83	45	5D	96
5C	33	98	B0
F0	2D	AD	C5

→

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

The prime (') indicates that the variable is to be updated by the value on the right. The AES standard depicts this transformation in matrix form as follows:

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad \textbf{(1.4.2)}$$

Equation (1.4.2) has to be interpreted carefully. In ordinary matrix multiplication, each element in the product matrix is the sum of products of the elements or one row and one column. In this case, each element in the product matrix is the bitwise XOR of products of elements of one row and one column. Further, the final the final addition shown in Eqn 1.4.2 is a bitwise XOR.

As an example, consider the input value {95}. The multiplicative inverse in GF(2^8) is {95} = {8A}, which is 10001010 in binary. Using Equation (2),

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

The result is {2A}, which should appear in row {09} column {05} of the S-box. This is verified by checking Table 1.4.2a.

The inverse substitute byte transformation, called InvSubBytes, makes use of the inverse S-box shown in Table 1.4.2b. Note, for example, that the input {2A} produces the output {95} and the input {95} to the S-box produces {2A}. The inverse S-box is constructed by applying the inverse

of the transformation in Equation (1.4.1) followed by taking the multiplicative inverse in GF(2^8). The inverse transformation is:

$$b_i' = b_{(i+2) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus d_i$$

where byte d = {05}, or 00000101. We can depict this transformation as follows:

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

To see that InvSubBytes is the inverse of SubBytes, label the matrices in SubBytes and InvSubBytes as X and Y, respectively, and the vector versions of constants c and d as C and D, respectively. For some 8-bit vector B, Equation (2) becomes B' = XB $\oplus$ C.

We need to show that Y (XB $\oplus$ C) $\oplus$ D = B. Multiply out, we must show YXB $\oplus$ YC $\oplus$ D = B. This becomes

We have demonstrated that YX equals the identity matrix, and the YC = D, so that YC $\oplus$ D equals the null vector.

Rationale

The S-box is designed to be resistant to known cryptanalytic attacks. In addition, the constant in Equation (1) was chosen so that the S-box has no fixed points [S-box(a) = a] and no "opposite fixed points" [S-box(a) =$\bar{a}$], where $\bar{a}$ is the bitwise complement of a.

Of course, the S-box must be invertible, that is, IS-box[S-box(a)] = a. However, the S-box is not self-inverse in the sense that it is not true that S-box(a) = IS - box(a). For example, [S-box({95}) = {2A}, but IS-box({95}) = {AD}.

ShiftRows Transformation

Forward and Inverse Transformations

The forward shift row transformation, called ShiftRows, is depicted in Figure 1.4.5a. The first row of State is not altered. For the second row, a 1-byte circular left shift is performed. For the third row, a 2-byte circular left shift is performed. For the fourth row, a 3-byte circular left shift is performed.

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus$$

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix}$$

The following is an example of ShiftRows:

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

The inverse shift row transformation, called InvShiftRows, performs the circular shifts in the opposite direction for each of the last three rows, with a one-byte circular right shift for the second row, and so on.

Rationale

The shift row transformation is more substantial than it may first appear. This is because the State, as well as the cipher input and output, is treated as an array of four 4-byte columns. Thus, on encryption, the first 4 bytes of the plaintext are copied to the first column of State, and so on. Further, the round key is applied to State column by column. Thus, a row shift moves an individual byte from one column to another, which is a linear distance of a multiple of 4 bytes.

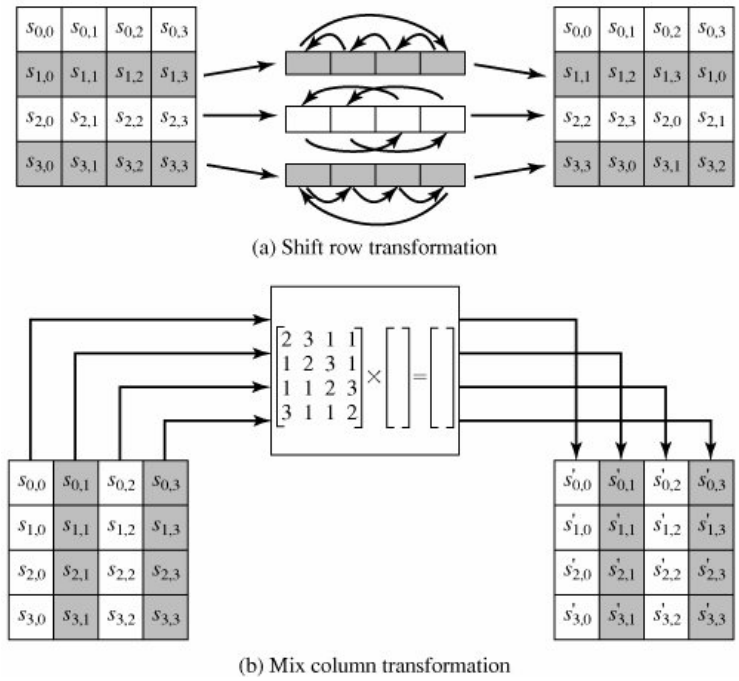


Fig 1.4.5 AES Row and Column Operations

Also note that the transformation ensures that the 4 bytes of one column are spread out to four different columns. Fig 1.4.3 illustrates the effect.

MixColumns Transformation

Forward and Inverse Transformations

The forward mix column transformation, called MixColumns, operates on each column individually. Each byte of a column is mapped into a new value that is a function of all four bytes in that column. The transformation can be defined by the following matrix multiplication on State (Fig 1.4.5b):

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix} \quad \textbf{(1.4.3)}$$

Each element in the product matrix is the sum of products of elements of one row and one column. In this case, the individual additions and multiplications are performed in GF(2^8). The MixColumns transformation on a single column j($0 \leq j \leq 3$) of State can be expressed as

$$
\begin{aligned}
s'_{0,j} &= (2 \bullet s_{0,j}) \oplus (3 \bullet s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \\
s'_{1,j} &= s_{0,j} \oplus (2 \bullet s_{1,j}) \oplus (3 \bullet s_{2,j}) \oplus s_{3,j} \\
s'_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus (2 \bullet s_{2,j}) \oplus (3 \bullet s_{3,j}) \\
s'_{3,j} &= (3 \bullet s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \bullet s_{3,j})
\end{aligned}
\qquad \textbf{(1.4.4)}
$$

The following is an example of MixColumns:

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

→

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

Let us verify the first column of this example. In GF(2^8), addition is the bitwise XOR operation and multiplication can be performed according to the rule

$$
x \times f(x) = \begin{cases} (b_6 b_5 b_4 b_3 b_2 b_1 b_0 0) & \text{if } b_7 = 0 \\ (b_6 b_5 b_4 b_3 b_2 b_1 b_0 0) \oplus (00011011) & \text{if } b_7 = 1 \end{cases}
$$

In particular, multiplication of a value by x (i.e., by {02}) can be implemented as a 1-bit left shift followed by a conditional bitwise XOR with (0001 1011) if the leftmost bit of the original value (prior to the shift) is 1. Thus, to verify the MixColumns transformation on the first column, we need to show that

({02} · {87})	⊕ ({03} · {6E})	⊕ {46}	⊕ {A6}	= {47}
{87}	⊕ ({02} · {6E})	⊕ ({03} · {46})	⊕ {A6}	= {37}
{87}	⊕ {6E}	⊕ ({02} · {46})	⊕ ({03} · {A6})	= {94}
({03} · {87})	⊕ {6E}	⊕ {46}	⊕ ({02} · {A6})	= {ED}

For the first equation, we have {02} · {87} = (0000 1110) ⊕ (0001 1011) = (0001 0101); and {03} · {6E} = {6E} ⊕ ({02} · {6E}) = (0110 1110) ⊕ (1101 1100) = (1011 0010). Then

{02} · {87} = 0001 0101

{03} · {6E} = 1011 0010

{46} = 0100 0110

$$\{02\} \cdot \{87\} = 0001\ 0101$$

$$\{A6\} = 1010\ 0110$$

$$0100\ 0111 = \{47\}$$

The other equations can be similarly verified.

The inverse mix column transformation, called InvMixColumns, is defined by the following matrix multiplication:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix} \quad \textbf{(5)}$$

It is not immediately clear that Equation (1.4.5) is the inverse of Equation (1.4.3). We need to show that:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}$$

which is equivalent to showing that:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(1.4.6)}$$

That is, the inverse transformation matrix times the forward transformation matrix equals the identity matrix. To verify the first column of Equation (1.4.6), we need to show that:

$$(\{0E\} \cdot \{02\}) \oplus \{0B\} \oplus \{0D\} \oplus (\{09\} \cdot \{03\}) = \{01\}$$

$$(\{09\} \cdot \{02\}) \oplus \{0E\} \oplus \{0B\} \oplus (\{0D\} \cdot \{03\}) = \{00\}$$

$$(\{0D\} \cdot \{02\}) \oplus \{09\} \oplus \{0E\} \oplus (\{0B\} \cdot \{03\}) = \{00\}$$

$$(\{0B\} \cdot \{02\}) \oplus \{0D\} \oplus \{09\} \oplus (\{0E\} \cdot \{03\}) = \{00\}$$

For the first equation, we have $\{0E\} \cdot \{02\}) \oplus 00011100$; and $\{09\} \cdot \{03\} = \{09\} \oplus (\{09\} \cdot \{02\}) = 00001001 \oplus 00010010 = 00011011$. Then

$$\{0E\} \cdot \{02\} = 00011100$$

$$\{0B\} = 00001011$$

$$\{0D\} = 00001101$$

$$\{0E\} \cdot \{02\} = 00011100$$

$$\{09\} \cdot \{03\} = 00011011$$

$$00000001$$

The other equations can be similarly verified.

The AES document describes another way of characterizing the MixColumns transformation, which is in terms of polynomial arithmetic. In the standard, MixColumns is defined by considering each column of State to be a four-term polynomial with coefficients in GF(2^8). Each column is multiplied modulo ($x^4 + 1$) by the fixed polynomial a(x), given by

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \qquad \textbf{(1.4.7)}$$

It can readily be shown that b(x) = a^1 (x) mod ($x^4 + 1$).

Rationale

The coefficients of the matrix in Equation (1.4.3) are based on a linear code with maximal distance between code words, which ensures a good mixing among the bytes of each column. The mix column transformation combined with the shift row transformation ensures that after a few rounds, all output bits depend on all input bits.

In addition, the choice of coefficients in MixColumns, which are all {01}, {02}, or {03}, was influenced by implementation considerations. As was discussed, multiplication by these coefficients involves at most a shift and an XOR. The coefficients in InvMixColumns are more formidable to implement. However, encryption was deemed more important than decryption for two reasons:

1. For the CFB and OFB cipher modes, only encryption is used.
2. As with any block cipher, AES can be used to construct a message authentication code (Part Two), and for this only encryption is used.

AddRoundKey Transformation

Forward and Inverse Transformations

In the forward add round key transformation, called AddRoundKey, the 128 bits of State are bitwise XORed with the 128 bits of the round key. As shown in Fig 1.4.4b, the operation is viewed as a columnwise operation between the 4 bytes of a State column and one word of the round key; it can also be viewed as a byte-level operation. The following is an example of AddRoundKey:

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

⊕

AC	19	28	57
77	FA	D1	5C
66	DC	29	00
F3	21	41	6A

=

EB	59	8B	1B
40	2E	A1	C3
F2	38	13	42
1E	84	E7	D2

The first matrix is State, and the second matrix is the round key.

The inverse add round key transformation is identical to the forward add round key transformation, because the XOR operation is its own inverse.

The add round key transformation is as simple as possible and affects every bit of State. The complexity of the round key expansion, plus the complexity of the other stages of AES, ensure security.

AES Key Expansion

Key Expansion Algorithm

The AES key expansion algorithm takes as input a 4-word (16-byte) key and produces a linear array of 44 words (176 bytes). This is sufficient to provide a 4-word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher. The following pseudocode describes the expansion:

```
KeyExpansion (byte key[16], word w[44])
{
   word temp
   for (i = 0; i < 4; i++)
    w[i] = (key[4*i],
    key[4*i+1],
   key[4*i+2],
   key[4*i+3]);
   for (i = 4; i < 44; i++)
   {
    temp = w[i  1];
    if (i mod 4 = 0) temp = SubWord (RotWord (temp))
                      ⊕Rcon[i/4];
    w[i] = w[i4]  temp
   }
}
```

The key is copied into the first four words of the expanded key. The remainder of the expanded key is filled in four words at a time. Each

added word w[i] depends on the immediately preceding word, w[i 1], and the word four positions back, w[i 4]. In three out of four cases, a simple XOR is used. For a word whose position in the w array is a multiple of 4, a more complex function is used. Fig 1.4.6 illustrates the generation of the first eight words of the expanded key, using the symbol g to represent that complex function. The function g consists of the following subfunctions:

1. RotWord performs a one-byte circular left shift on a word. This means that an input word [b0, b1, b2, b3] is transformed into [b1, b2, b3, b0].
2. SubWord performs a byte substitution on each byte of its input word, using the S-box(Table 1.9a).
3. The result of steps 1 and 2 is XORed with a round constant, Rcon[j].

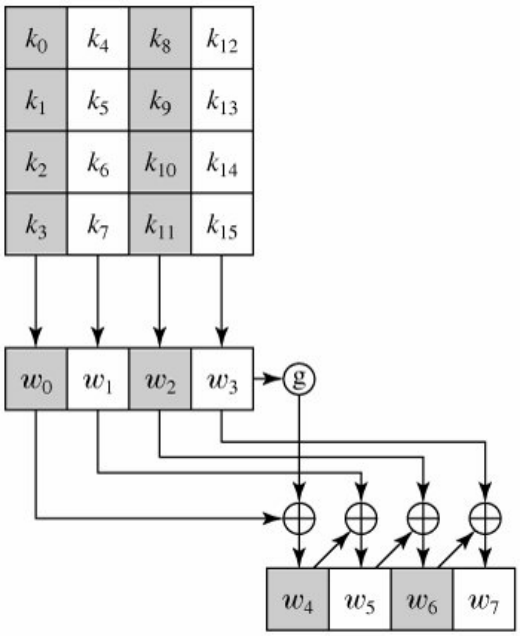


Fig 1.4.6 AES Key Expansion

The round constant is a word in which the three rightmost bytes are always 0. Thus the effect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byte of the word. The round constant is different for each round and is defined as Rcon[j] = (RC[j], 0, 0, 0), with RC[1] = 1, RC[j] = 2 · RC[j - 1] and with multiplication defined over the field GF(2^8). The values of RC[j] in hexadecimal are

j	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1B	36

For example, suppose that the round key for round 8 is

EA D2 73 21 B5 8D BA D2 31 2B F5 60 7F 8D 29 2F

Then the first 4 bytes (first column) of the round key for round 9 are calculated as follows:

i (decimal)	temp	After RotWord	After SubWord	Rcon (9)	After XOR with Rcon	w[i 4]	w[i] = temp ⊕ w[i 4]
36	7F8D 292F	8D292F7F	5DA515D2	1B000 000	46A515 D2	EAD2 7321	AC776 6F3

Rationale

The Rijndael developers designed the expansion key algorithm to be resistant to known cryptanalytic attacks. The inclusion of a round-dependent round constant eliminates the symmetry, or similarity, between the ways in which round keys are generated in different rounds.

- Knowledge of a part of the cipher key or round key does not enable calculation of many other round key bits
- An invertible transformation [i.e., knowledge of any Nk consecutive words of the Expanded Key enables regeneration the entire expanded key (Nk = key size in words)]
- Speed on a wide range of processors
- Usage of round constants to eliminate symmetries
- Diffusion of cipher key differences into the round keys; that is, each key bit affects many round key bits
- Enough nonlinearity to prohibit the full determination of round key differences from cipher key differences only
- Simplicity of description

The authors do not quantify the first point on the preceding list, but the idea is that if you know less than Nk consecutive words of either the cipher key or one of the round keys, then it is difficult to reconstruct the remaining unknown bits. The fewer bits one knows, the more difficult it is to do the reconstruction or to determine other bits in the key expansion.

Equivalent Inverse Cipher

The AES decryption cipher is not identical to the encryption cipher (Fig1.4.1). That is, the sequence of transformations for decryption differs

from that for encryption, although the form of the key schedules for encryption and decryption is the same. This has the disadvantage that two separate software or firmware modules are needed for applications that require both encryption and decryption. There is an equivalent version of the decryption algorithm that has the same structure as the encryption algorithm. The equivalent version has the same sequence of transformations as the encryption algorithm (with transformations replaced by their inverses). To achieve this equivalence, a change in key schedule is needed.

Two separate changes are needed to bring the decryption structure in line with the encryption structure. An encryption round has the structure SubBytes, ShiftRows, MixColumns, AddRoundKey. The standard decryption round has the structure InvShiftRows, InvSubBytes, AddRoundKey, InvMixColumns. Thus, the first two stages of the decryption round need to be interchanged, and the second two stages of the decryption round need to be interchanged.

Interchanging InvShiftRows and InvSubBytes

InvShiftRows affects the sequence of bytes in State but does not alter byte contents and does not depend on byte contents to perform its transformation. InvSubBytes affects the contents of bytes in State but does not alter byte sequence and does not depend on byte sequence to perform its transformation. Thus, these two operations commute and can be interchanged. For a given State S_i,

InvShiftRows [InvSubBytes (S_i)] = InvSubBytes [InvShiftRows (S_i)]

Interchanging AddRoundKey and InvMixColumns

The transformations AddRoundKey and InvMixColumns do not alter the sequence of bytes in State. If we view the key as a sequence of words, then both AddRoundKey and InvMixColumns operate on State one column at a time. These two operations are linear with respect to the column input. That is, for a given State S_i and a given round key w_j:

InvMixColumns ($S_i \oplus w_j$) = [InvMixColumns (S_i)] $\oplus$ [InvMixColumns (w_j)]

To see this, suppose that the first column of State S_i is the sequence (y_0, y_1, y_2, y_3) and the first column of the round key w_j is (k_0, k_1, k_2, k_3). Then we need to show that

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} y_0 \oplus k_0 \\ y_1 \oplus k_1 \\ y_2 \oplus k_2 \\ y_3 \oplus k_3 \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \oplus \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} k_0 \\ k_1 \\ k_2 \\ k_3 \end{bmatrix}$$

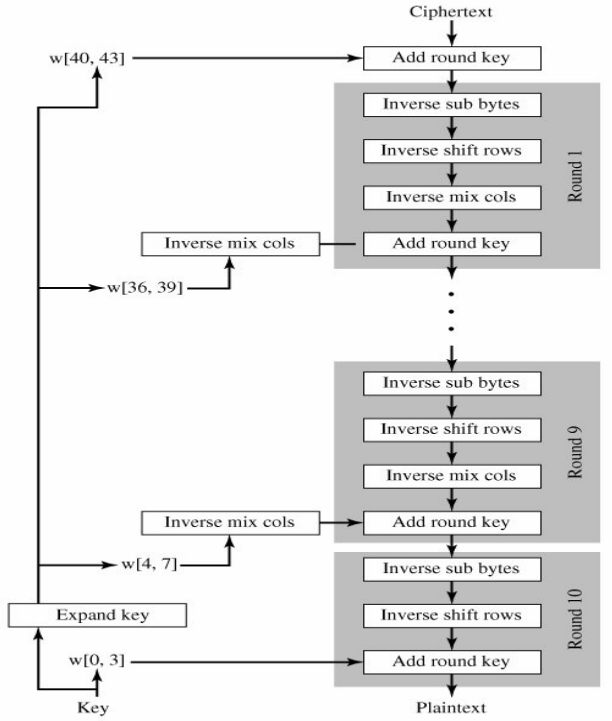


Fig 1.4.7 Equivalent Inverse Cipher

Let us demonstrate that for the first column entry. We need to show that:

$$[\{0E\} \cdot (y_0 \oplus k_0)] \oplus [\{0B\} \cdot (y_1 \oplus k_1)] \oplus [\{0D\} \cdot (y_2 \oplus k_2)] \oplus [\{09\} \cdot (y_3 \oplus k_3)]$$

$$= [\{0E\} \cdot y_0] \oplus [\{0B\} \cdot y_1] \oplus [\{0D\} \cdot y_2] \oplus [\{09\} \cdot y_3] \oplus [[\{0E\} \cdot k_0] \oplus] [\{0B\} \cdot k_1] \oplus [\{0D\} \cdot k_2] \oplus [\{09\} \cdot k_3]$$

This equation is valid by inspection. Thus, we can interchange AddRoundKey and InvMixColumns, provided that we first apply InvMixColumns to the round key. Note that we do not need to apply InvMixColumns to the round key for the input to the first AddRoundKey transformation (preceding the first round) nor to the last AddRoundKey transformation (in round 10). This is because these two AddRoundKey transformations are not interchanged with InvMixColumns to produce the equivalent decryption algorithm. Figure 1.4.7 illustrates the equivalent decryption algorithm.

REFERENCES

1. William Stallings, Cryptography and Network Security, PHI Publishers
2. www.wikipedia.org

Review Questions:

1. Explain the AES algorithm in detail?
2. Explain the issues related to the strength of the AES algorithm?

UNIT II

1. NUMBER THEORY

2. DISCRETE LOGARITHMS AND ALGORITHMS AND ALGORITHM COMPLEXITY

3. PUBLIC KEY CRYPTOSYSTEMS

4. RSA AND DEFFIE HELLMAN KEY EXCHANGE ALGORITHMS

1.NUMBER THEORY

OBJECTIVE

A number of concepts from number theory are essential in the design of public-key cryptographic algorithms. This lesson provides an overview of the concepts referred to in other lessons.

Prime Numbers

An integer p > 1 is a prime number if and only if its only divisors are ± 1 and ±p. Prime numbers play a critical role in number theory. Table 2.1.1 shows the primes less than 2000.

2	101	211	307	401	503	601	701	809	0	1009	1103	1201	1301	1409	1511	1601	1709	1801	1901
3	103	223	311	409	509	607	709	811	911	1013	1109	1213	1303	1423	1523	1607	1721	1811	1907
5	107	227	313	419	521	613	719	821	919	1019	1117	1217	1307	1427	1531	1609	1723	1823	1913
7	109	229	317	421	523	617	727	823	929	1021	1123	1223	1319	1429	1543	1613	1733	1831	1931
11	113	233	331	431	541	619	733	827	937	1031	1129	1229	1321	1433	1549	1619	1741	1847	1933
13	127	239	337	433	547	631	739	829	941	1033	1151	1231	1327	1439	1553	1621	1747	1861	1949
17	131	241	347	439	557	641	743	839	947	1039	1153	1237	1361	1447	1559	1627	1753	1867	1951
19	137	251	349	443	563	643	751	853	953	1049	1163	1249	1367	1451	1567	1637	1759	1871	1973
23	139	257	353	449	569	647	757	857	967	1051	1171	1259	1373	1453	1571	1657	1777	1873	1979
29	149	263	359	457	571	653	761	859	971	1061	1181	1277	1381	1459	1579	1663	1783	1877	1987
31	151	269	367	461	577	659	769	863	977	1063	1187	1279	1399	1471	1583	1667	1787	1879	1999
37	157	271	373	463	587	661	773	877	983	1069	1193	1283		1481	1597	1669	1789	1889	1997
41	163	277	379	467	593	673	787	881	991	1087		1289		1483		1693			1999
43	167	281	383	479	599	677	797	883	997	1091		1291		1487		1697			
47	173	283	389	487		683		887		1093		1297		1489		1699			
53	179	293	397	491		691				1097				1493					
59	181			499										1499					
61	191																		
67	193																		
71	197																		
73	199																		
79																			
83																			
89																			
97																			

Table 2.1.1 Primes under 2000

Any integer a > 1 can be factored in a unique way as

$$a = p_1^{a_1} p_2^{a_2} p_3^{a_3} \ldots\ldots p_t^{a_t} \quad \text{----- (2.1.1)}$$

where p1 < p_2 < ... < p_t are prime numbers and where each is a positive integer. This is known as the fundamental theorem of arithmetic.

91	= 7 x 13
3600	= 2^4 x 3^2 x 5^2
11011	= 7 x 11^2 x 13

If P is the set of all prime numbers, then any positive integer a can be written uniquely in the following form:

$$a = \prod_{p \in P} p^{a_p}$$

where each $a_p \geq 0$.

The right-hand side is the product over all possible prime numbers p. For any particular value of a, most of the exponents a_p will be 0. The value of any given positive integer can be specified by simply listing all the nonzero exponents in the foregoing formulation.

The integer 12 is represented by $\{a_2 = 2, a_3 = 1\}$.
The integer 18 is represented by $\{a_2 = 1, a_3 = 2\}$.
The integer 91 is represented by $\{a_7 = 2, a_{13} = 1\}$.

Multiplication of two numbers is equivalent to adding the corresponding exponents. Given

$$a = \prod_{p \in P} p^{a_p}, \qquad b = \prod_{p \in P} p^{b_p}$$

Define k = ab We know that the integer k can be expressed as the product of powers of primes:

$$k = \prod_{p \in P} p^{k_p}$$

It follows that $k_p = a_p + b_p$ for all p Є P.

$k = 12 \times 18 = (2^2 \times 3) \times (2 \times 3^2) = 216$
$k_2 = 2 + 1 = 3; k_3 = 1 + 2 = 3$
$216 = 2^3 \times 3^3 = 8 \times 27$

Any integer of the form can be divided only by an integer that is of a lesser or equal power of the same prime number, p^j with $j \leq n$. Thus, we can say the following:
Given,

$$a = \prod_{p \in P} p^{a_p}, \qquad b = \prod_{p \in P} p^{b_p}$$

If a|b, then $a_p \leq b_p$ then for all p.

a	= 12;b = 36; 12|36
12	$= 2^2 \times 3; 36 = 2^2 \times 3^2$
a_2	$= 2 = b_2$
a_3	$= 1 \leq 2 = b_3$
Thus, the inequality $a_p \leq b_p$ is satisfied for all prime numbers.	

It is easy to determine the greatest common divisor of two positive integers if we express each integer as the product of primes. The greatest common divisor of integers a and b, expressed gcd(a, b), is an

integer c that divides both a and b without remainder and that any divisor of a and b is a divisor of c.

300	$= 2^2 \times 3^1 \times 5^2$
18	$= 2^1 \times 3^2$
gcd(18,300)	$= 2^1 \times 3^1 \times 5^0 = 6$

The following relationship always holds:

If k = gcd(a,b) then $k_p = \min(a_p, b_p)$ for all p

Determining the prime factors of a large number is no easy task, so the preceding relationship does not directly lead to a practical method of calculating the greatest common divisor.

Fermat's and Euler's Theorems

Two theorems that play important roles in public-key cryptography are Fermat's theorem and Euler's theorem.

Fermat's Theorem

This is sometimes referred to as Fermat's little theorem. Fermat's theorem states the following: **If p is prime and a is a positive integer not divisible by p, then**

$$a^{p-1} \equiv 1 \pmod{p} \quad \text{----- (2.1.2)}$$

Proof: Consider the set of positive integers less than p:{1,2,..., p 1} and multiply each element by a, modulo p, to get the set X = {a mod p, 2a mod p, . . . (p 1)a mod p}. None of the elements of X is equal to zero because p does not divide a. Furthermore no two of the integers in X are equal. To see this, assume that ja ≡ ka(mod p) where 1 ≡ j < k ≡ p 1. Because a is relatively prime to p, we can eliminate a from both sides of the equation resulting in: j ≡ k(mode p). This last equality is impossible because j and k are both positive integers less than p. Therefore, we know that the (p 1) elements of X are all positive integers, with no two elements equal. We can conclude the X consists of the set of integers {1,2,..., p 1} in some order. Multiplying the numbers in both sets and taking the result mod p yields

Recall that two numbers are relatively prime if they have no prime factors in common; that is, their only common divisor is 1. This is equivalent to saying that two numbers are relatively prime if their greatest common divisor is 1.

$$a \times 2a \times \ldots \times (p\ 1) \equiv [(1 \times 2 \times \ldots \times (p\ 1)](\text{mode } p)$$

$$a^{p1(p\ 1)!} \equiv (p\ 1)!(\text{mod } p)$$

We can cancel the (p 1)! term because it is relatively prime to p. This yields Equation (2.1.2).

An alternative form of Fermat's theorem is also useful: If p is prime and a is a positive integer, then

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

a = 7, p = 19
$7^2 = 49 \equiv 11 (\text{mod } 19)$
$7^4 \equiv 121 \equiv 7 (\text{mod } 19)$
$7^8 \equiv 49 \equiv 7 (\text{mod } 19)$
$7^{16} \equiv 121 \equiv 7 (\text{mod } 19)$
$a^{p1} = 7^{18} = 7^{16} \times 7^2 \equiv 7 \times 11 \equiv 1 (\text{mod } 19)$

Note that the first form of the theorem [Equation (2.1.2)] requires that a be relatively prime to p, but this form does not.

p = 5,a = 3	$a^p = 3^5 = 243 \equiv 3 (\text{mod } 5) = a (\text{mod } p)$
p = 5, a = 10	$a^p = 10^5 = 100000 \equiv 10 (\text{mod } 5) = 0 (\text{mod } 5) = a (\text{mod } p)$

Euler's Totient Function

Before presenting Euler's theorem, we need to introduce an important quantity in number theory, referred to as Euler's totient function and written Φ(n), defined as the number of positive integers less than n and relatively prime to n. By convention, Φ (1) = 1.

Determine Φ(37) and Φ(35).
Because 37 is prime, all of the positive integers from 1 through 36 are relatively prime to 37. Thus Φ(37) = 36.
To determine Φ(35), we list all of the positive integers less than 35 that are relatively prime to it:
1, 2, 3, 4, 6, 8, 9, 11, 12, 13, 16, 17, 18,
19, 22, 23, 24, 26, 27, 29, 31, 32, 33, 34.
There are 24 numbers on the list, so Φ(35) = 24.

Table 2.1.2 lists the first 30 values of Φ (n). The value Φ (1) is without meaning but is defined to have the value 1.

n	**Φ(n)**
1	1
2	1
3	2
4	2
5	4
6	2
7	6
8	4

n	Φ(n)
9	6
10	4
11	10
12	4
13	12
14	6
15	8
16	8
17	16
18	6
19	18
20	8
21	12
22	10
23	22
24	8
25	20
26	12
27	18
28	12
29	28
30	8

Table 2.1.2 Some Values of Euler's Totient Function f(n)

It should be clear that for a prime number p,

$$\Phi (p) = p\ 1$$

Now suppose that we have two prime numbers p and q, with $p \neq q$. Then we can show that for n = pq,

$\Phi(n) = \Phi(pq) = \Phi(p) \times \Phi(q) = (p\ 1) \times (q \times 1)$

To see that $\Phi(n) = \Phi(p) \times \Phi(q)$, consider that the set of positive integers less that n is the set {1,..., (pq 1)}. The integers in this set that are not relatively prime to n are the set {p,2 p,..., (q 1)p} and the set {q,2q,..., (p 1)q} Accordingly,

$$\begin{aligned}\Phi (n) &= (pq\ 1)\ [(q\ 1) + (p\ 1)] \\ &= pq\ (p + q) + 1 \\ &= (p\ 1) \times (q\ 1) \\ &= \Phi (p) \times \Phi (q)\end{aligned}$$

Φ (21) = Φ (3) x Φ (7) = (3 1) x (7 1) = 2 x 6 = 12
where the 12 integers are {1,2,4,5,8,10,11,13,16,17,19,20}

Euler's Theorem

Euler's theorem states that

For every a and n that are relatively prime:

$$a^{\Phi(n)} \equiv 1 \ (mod\ n) \quad \text{----- (2.1.3)}$$

a = 3; n = 10; Φ(10) = 4	$a^{\Phi(n)} = 3^4 = 81 \equiv 1 (\text{mod } 10) = 1 \ (\text{mod } n)$
a = 2; n = 11; Φ(11) = 10	$a^{\Phi(n)} = 2^{10} = 1024 \equiv 1 (\text{mod } 11) = 1 \ (\text{mod } n)$

Proof: Equation (2.1.3) is true if n is prime, because in that case Φ(n) = (n 1) and Fermat's theorem holds. However, it also holds for any integer n. Recall that Φ(n) is the number of positive integers less than n that are relatively prime to n. Consider the set of such integers, labeled as follows:

$$R \{x_1, x_2, \ldots, x_{\Phi(n)}\}$$

That is, each element x_i of R is a unique positive integer less than n with $\gcd(x_i, n) = 1$. Now multiply each element by a, modulo n:

$$S = \{(ax_1 \bmod n), (ax_2 \bmod n), \ldots, (ax_{\Phi(n)} \bmod n)\}$$

The set S is a permutation of R, by the following line of reasoning:

1. Because a is relatively prime to n and x_i is relatively prime to n, ax_i must also be relatively prime to n. Thus, all the members of S are integers that are less than n and that are relatively prime to n.
2. There are no duplicates in S. we know that $ax_i \bmod n = ax_j \bmod n$ then $x_i = x_j$.

Therefore,

$$\prod_{i=1}^{\Phi(n)} (ax_i \bmod n) = \prod_{i=1}^{\Phi(n)} x_i$$

$$\prod_{i=1}^{\Phi(n)} ax_i = \prod_{i=1}^{\Phi(n)} x_i \ (mod\ n)$$

$$a^{\Phi(n)} \times \left[\prod_{i=1}^{\Phi(n)} x_i\right] \equiv \prod_{i=1}^{\Phi(n)} x_i \ (mod\ n)$$

$$a^{\Phi(n)} \equiv 1 (mod\ n) \quad \text{----- (2.1.4)}$$

This is the same line of reasoning applied to the proof of Fermat's theorem. As is the case for Fermat's theorem, an alternative form of the theorem is also useful:

$$a^{\Phi(n)+1} \equiv a \ (mod\ n)$$

Again, similar to the case with Fermat's theorem, the first form of Euler's theorem [Equation (3)] requires that a be relatively prime to n, but this form does not.

Testing for Primality

For many cryptographic algorithms, it is necessary to select one or more very large prime numbers at random. Thus we are faced with the task of determining whether a given large number is prime. There is no simple yet efficient means of accomplishing this task.

In this section, we present one attractive and popular algorithm. This algorithm yields a number that is not necessarily a prime. However, the algorithm can yield a number that is almost certainly a prime.

Miller-Rabin Algorithm

Also referred to in the literature as the Rabin-Miller algorithm, or the Rabin-Miller test, or the Miller-Rabin test, the algorithm due to Miller and Rabin is typically used to test a large number for primality. Before explaining the algorithm, we need some background. First, any positive odd integer n ≥ 3 can be expressed as follows:

$$n\ 1 = 2^k q \text{ with } k > 0, q \text{ odd}$$

To see this, note that (n 1) is an even integer. Then, divide (n 1) by 2 until the result is an odd number q, for a total of k divisions. If n is expressed as a binary number, then the result is achieved by shifting the number to the right until the rightmost digit is a 1, for a total of k shifts. We now develop two properties of prime numbers that we will need.

Two Properties of Prime Numbers

The first property is stated as follows: If p is prime and a is a positive integer less than p, then a^2 mod p = 1 if and only if either a mod p = 1 or a mod p= 1 mode p = p 1. By the rules of modular arithmetic (a mode p) (a mode p) = a^2 mod p. Thus if either a mode p = 1 or a mod p = 1, then a^2 mod p = 1. Conversely, if a^2 mod p = 1, then (a mod p)^2 = 1, which is true only for a mod p = 1 or a mod p = 1.

The second property is stated as follows: Let p be a prime number greater than 2. We can then write p 1 = 2^kq, with k > 0 q odd. Let a be any integer in the range 1 < a < p 1. Then one of the two following conditions is true:

1. a^q is congruent to 1 modulo p. That is, a^q mod p = 1, or equivalently, $a^q \equiv 1 \pmod p$.
2. One of the numbers a^q, a^{2q}, a^{4q},..., a^{2k-1q} is congruent to 1 modulo p. That is, there is some number j in the range ($1 \le j \le k$) such that a^{2j-1q} mod p = 1 mod p = p 1, or equivalently, $a^{2j-1q} \equiv 1 \pmod p$.

Proof: Fermat's theorem [Equation (2.1.1)] states that $a^{n1} \equiv 1 \pmod n$ if n is prime. We have p 1 = 2^kq. Thus, we know that a^{p1} mod p = a^{2kq} mod p = 1. Thus, if we look at the sequence of numbers

$$a^q \bmod p, a^{2q} \bmod p, a^{4c} \bmod p, \ldots, a^{2k-1q} \bmod p, a^{2kc} \bmod p \quad \text{----- (2.1.5)}$$

we know that the last number in the list has value 1. Further, each number in the list is the square of the previous number. Therefore, one of the following possibilities must be true:

1. The first number on the list, and therefore all subsequent numbers on the list, equals 1.

2. Some number on the list does not equal to 1, but its square mod p does equal 1. By virtue of the first property of prime numbers defined above, we know that the only number that satisfies this condition p 1 is So, in this case, the list contains an element equal to p 1.

   This completes the proof.

A Deterministic Primality Algorithm

Prior to 2002, there was no known method of efficiently proving the primality of very large numbers. All of the algorithms in use, including the most popular (Miller-Rabin), produced a probabilistic result. In 2002, Agrawal, Kayal, and Saxena developed a relatively simple deterministic algorithm that efficiently determines whether a given large number is a prime. The algorithm, known as the AKS algorithm, does not appear to be as efficient as the Miller-Rabin algorithm. Thus far, it has not supplanted this older, probabilistic technique.

Distribution of Primes

It is worth noting how many numbers are likely to be rejected before a prime number is found using the Miller-Rabin test, or any other test for primality. A result from number theory, known as the prime number theorem, states that the primes near n are spaced on the average one every (ln n) integers. Thus, on average, one would have to test on the order of ln(n) integers before a prime is found. Because all even integers can be immediately rejected, the correct figure is 0.5 ln(n). For example, if a prime on the order of magnitude of 2^{200} were sought, then about 0.5 $\ln(2^{200}) = 69$ trials would be needed to find a prime. However, this figure is just an average. In some places along the number line, primes are closely packed, and in other places there are large gaps.

> The two consecutive odd integers 1,000,000,000,061 and 1,000,000,000,063 are both prime. On the other hand, 1001! + 2,1001! + 3,..., 1001! + 1000, 1001! + 1001 is a sequence of 1000 consecutive composite integers.

The Chinese Remainder Theorem

One of the most useful results of number theory is the Chinese remainder theorem (CRT). In essence, the CRT says it is possible to reconstruct integers in a certain range from their residues modulo a set of pairwise relatively prime moduli. The CRT is so called because it is believed to have been discovered by the Chinese mathematician Sun-Tsu in around 100 A.D.

> The 10 integers in Z_{10}, that is the integers 0 through 9, can be reconstructed from their two residues modulo 2 and 5 (the relatively prime factors of 10). Say the known residues of a decimal digit x are $r_2 = 0$ and $r_5 = 3$; that is, x mod 2 =0 and x mod 5 = 3. Therefore, x is an even integer in Z_{10} whose remainder, on division by 5, is 3. The unique solution is x = 8.

The CRT can be stated in several ways. We present here a formulation that is most useful from the point of view of this text. An alternative formulation is explored in Problem 8.17. Let

$$M = \prod_{i=1}^{k} m_i$$

where the m_i are pairwise relatively prime; that is, gcd(m_i, m_j) = 1 for 1 ≤ i, j ≤ k, and i ≠ j. We can represent any integer A in Z^M by a k-tuple whose elements are in Z_{mi} using the following correspondence:

$$A \leftrightarrow (a_1, a_2, \ldots\ldots, a_k) \quad \textbf{----- (2.1.6)}$$

where A ЄZ_M, a_i Є Z_{mi} and a_i = A mod m_i for 1 ≤ i ≤ k. The CRT makes two assertions.

1. The mapping of Equation (6) is a one-to-one correspondence (called a **bijection**) between Z_M and the Cartesian product Z_{m1} x Z_{m2} x ... x Z_{mk}. That is, for every integer A such that 0 ≤ A < M there is a unique k-tuple (a_1, a_2,..., a_k) with 0 ≤ a_i < m_i that represents it, and for every such k-tuple (a_1, a_2,..., a_k) there is a unique integer A in Z_M.
2. Operations performed on the elements of Z_M can be equivalently performed on the corresponding k-tuples by performing the operation independently in each coordinate position in the appropriate system.

Let us demonstrate the first assertion. The transformation from A to (a_1, a_1,..., a_k) is obviously unique; that is, each a_i is uniquely calculated as a_i = A mod m_i. Computing A from (a_1, a_1,..., a_k) can be done as follows. Let M_i = M/m_i for 1 ≤ i ≤ k. Note that M_i = m_1 x m_2 x ... x m_{i-1} x m_{i+1} x ... x m_k so that M_i ≡ 0(mod m_j) for all j ≠ i. Then let

$$c_i = M_i \, X \, (M_i^{-1} \bmod m_i) \quad \textbf{for 1 ≤ I ≤ k} \quad \textbf{----- (2.1.7)}$$

By the definition of M_i it is relatively prime to m_i and therefore has a unique multiplicative inverse mod m_i So Equation (2.1.7) is well defined and produces a unique value c_i. We can now compute:

$$A \equiv \left(\sum_{i=1}^{k} a_i \, c_i \right) (\text{mod } M) \quad \text{----- (2.1.8)}$$

To show that the value of A produced by Equation (8) is correct, we must show that a_i = A mod m_i for 1≤ i ≤ k. Note that c_j ≡ M_j ≡ 0(mod m_i) if j ≠ i and that c_i ≡ 1(mod m_i). It follows that a_i = A mod m_i.

The second assertion of the CRT, concerning arithmetic operations, follows from the rules for modular arithmetic. That is, the second assertion can be stated as follows: If

$$A \leftrightarrow (a_1, a_2, \ldots\ldots, a_k)$$
$$B \leftrightarrow (b_1, b_2, \ldots\ldots, b_k)$$

then

$$(A + B) \bmod M \leftrightarrow ((a_1 + b_1) \bmod m_1, \ldots\ldots\ldots, (a_k + b_k) \bmod m_k)$$
$$(A - B) \bmod M \leftrightarrow ((a_1 - b_1) \bmod m_1, \ldots\ldots\ldots, (a_k - b_k) \bmod m_k)$$
$$(A \, X \, B) \bmod M \leftrightarrow ((a_1 \, X \, b_1) \bmod m_1, \ldots\ldots\ldots, (a_k \, X \, b_k) \bmod m_k)$$

One of the useful features of the Chinese remainder theorem is that it provides a way to manipulate (potentially very large) numbers mod M in terms of tuples of smaller numbers. This can be useful when M is 150

digits or more. However, note that it is necessary to know beforehand the factorization of M.

Worked Example:

To represent 973 mod 1813 as a pair of numbers mod 37 and 49, define

$m_1 = 37$

$m_2 = 49$

M = 1813

A = 973

We also have $M_1 = 49$ and $M_2 = 37$. Using the extended Euclidean algorithm, we compute

$M_1^{-1} = 34 \bmod m_1$ and $M_2^{-1} = 4 \bmod m_2$. (Note that we only need to compute each M_i and each M_i^{-1} = once.)

Taking residues modulo 37 and 49, our representation of 973 is (11, 42), because 973 mod 37 = 11 and 973 mod 49 = 42.

Now suppose we want to add 678 to 973. What do we do to (11, 42)? First we compute

(678) ↔ (678 mod 37, 678 mod 49) = (12, 41). Then we add the tuples element-wise and reduce (11 + 12 mod 37, 42 + 41 mod 49) = (23, 34).

To verify that this has the correct effect, we compute (23,34) ↔ $a_1M_1 M_1^{-1} + a_2M_2 M_2^{-1} \bmod M$

= [(23)(49)(34) + (34)(37)(4)] mod 1813

= 43350 mod 1813

= 1651

Check that it is equal to (973 + 678) mod 1813 = 1651.

Remember that in the above derivation, M_1^{-1} is the multiplicative inverse of M_1 modulo m_1, and M_2^{-1} is the multiplicative inverse of M_2 modulo m_2.

Suppose we want to multiply 1651 (mod 1813) by 73. We multiply (23, 34) by 73 and reduce to get (23 x 73 mod 37, 34 x 73 mod 49) = (14,32). It is easily verified that

(32,14) ↔ [(14)(49)(34) + (32)(37)(4)] mod 1813

= 865

= 1651 x 73 mod 1813

2. DISCRETE LOGARITHMS AND ALGORITHM COMPLEXITY

OBJECTIVE

Discrete logarithms are fundamental to a number of public-key algorithms, including Diffie-Hellman key exchange and the digital signature algorithm (DSA). This lesson provides a brief overview of discrete logarithms.

The Powers of an Integer, Modulo n

Recall from Euler's theorem we have, for every a and n that are relatively prime:

$a\Phi(n) \equiv 1 \pmod{n}$

where Φ(n), Euler's totient function, is the number of positive integers less than n and relatively prime to n. Now consider the more general expression:

$$a^m \equiv 1 \pmod{n} \quad \text{------ (2.2.1)}$$

If a and n are relatively prime, then there is at least one integer m that satisfies Equation (2.2.1), namely, m = Φ (n). The least positive exponent m for which Equation (2.2.1) holds is referred to in several ways:

- the order of a (mod n)
- the exponent to which a belongs (mod n)
- the length of the period generated by a

To see this last point, consider the powers of 7, modulo 19:

$7^1 \equiv 7 \pmod{19}$

$7^2 = 49 = 2 \times 19 + 11 \equiv 11 \pmod{19}$

$7^3 = 343 = 18 \times 19 + 1 \equiv 1 \pmod{19}$

$7^4 = 2401 = 126 \times 19 + 7 \equiv 7 \pmod{19}$

$7^5 = 16807 = 884 \times 19 + 11 \equiv 11 \pmod{19}$

There is no point in continuing because the sequence is repeating. This can be proven by noting that $7^3 \equiv 1 \pmod{19}$ and therefore $7^{3+j} \equiv 7^3 7^j \equiv 7^j \pmod{19}$, and hence any two powers of 7 whose exponents differ by 3 (or a multiple of 3) are congruent to each other (mod 19). In other words, the sequence is periodic, and the length of the period is the smallest positive exponent m such that $7^m \equiv 1 \pmod{19}$.

Table 2.2.1 shows all the powers of a, modulo 19 for all positive a < 19. The length of the sequence for each base value is indicated by shading. Note the following:

1. All sequences end in 1. This is consistent with the reasoning of the preceding few paragraphs.
2. The length of a sequence divides Φ(19) = 18. That is, an integral number of sequences occur in each row of the table.

3. Some of the sequences are of length 18. In this case, it is said that the base integer a generates (via powers) the set of nonzero integers modulo 19. Each such integer is called a primitive root of the modulus 19.

a	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}	a^{13}	a^{14}	a^{15}	a^{16}	a^{17}	a^{18}
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	1
3	9	8	5	15	7	2	6	18	16	10	11	14	4	12	17	13	1
4	16	7	9	17	11	6	5	1	4	16	7	9	17	11	6	5	1
5	6	11	17	9	7	16	4	1	5	6	11	17	9	7	16	4	1
6	17	7	4	5	11	9	16	1	6	17	7	4	5	11	9	16	1
7	11	1	7	11	1	7	11	1	7	11	1	7	11	1	7	11	1
8	7	18	11	12	1	8	7	18	11	12	1	8	7	18	11	12	1
9	5	7	6	16	11	4	17	1	9	5	7	6	16	11	4	17	1
10	5	12	6	3	11	15	17	18	9	14	7	13	16	8	4	2	1
11	7	1	11	7	1	11	7	1	11	7	1	11	7	1	11	7	1
12	11	18	7	8	1	12	11	18	7	8	1	12	11	18	7	8	1
13	17	12	4	14	11	10	16	18	6	2	7	15	5	8	9	3	1
14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	1
15	16	12	9	2	11	13	5	18	4	3	7	10	17	8	6	14	1
16	9	11	5	4	7	17	6	1	16	9	11	5	4	7	17	6	1
17	4	11	16	6	7	5	9	1	17	4	11	16	6	7	5	9	1
18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1

Table 2.2.1 Powers of Integers, Modulo 19

The highest possible exponent to which a number can belong (mod n) is Φ(n). If a number is of this order, it is referred to as a primitive root of n. The importance of this notion is that if a is a primitive root of n, then its powers

$$a, a^2, ..., a^{\Phi(n)}$$

are distinct (mod n) and are all relatively prime to n. In particular, for a prime number p, if a is a primitive root of p, then

$$a, a^2, ..., a^{p1}$$

are distinct (mod p). For the prime number 19, its primitive roots are 2, 3, 10, 13, 14, and 15.

Not all integers have primitive roots. In fact, the only integers with primitive roots are those of the form 2, 4, p^{Φ}, and $2p^{\Phi}$, where p is any odd prime and Φ is a positive integer.

Logarithms for Modular Arithmetic

With ordinary positive real numbers, the logarithm function is the inverse of exponentiation. An analogous function exists for modular arithmetic.

Let us briefly review the properties of ordinary logarithms. The logarithm of a number is defined to be the power to which some positive base (except 1) must be raised in order to equal the number. That is, for base x and for a value y:

$$\mathbf{y = x^{log_x(y)}}$$

The properties of logarithms include the following:

$\log_x(1) = 0$

$\log_x(x) = 1$

$$\mathbf{\log_x(yz) = \log_x(y) + \log_x(y) \text{ ----- (2.2.2)}}$$

$$\log_x(y^r) = r \times \log_x(y) \quad \text{----- (2.2.3)}$$

Consider a primitive root a for some prime number p (the argument can be developed for nonprimes as well). Then we know that the powers of a from 1 through (p 1) produce each integer from 1 through (p 1) exactly once. We also know that any integer b satisfies

b ≡ r(mod p) for some r, where 0 ≤ r ≤ (p 1)

by the definition of modular arithmetic. It follows that for any integer b and a primitive root a of prime number p, we can find a unique exponent i such that

$$b \equiv a^i \pmod p \text{ where } 0 \le i \le (p-1)$$

This exponent i is referred to as the **discrete logarithm** of the number b for the base a (mod p). We denote this value as

$$\text{dlog}_{a,p}(b)$$

Note the following:

$$\text{dlog}_{a,p}(1) = 0, \text{ because } a^0 \bmod p = 1 \bmod p = 1 \quad \text{----- (2.2.3)}$$

$$\text{dlog}_{a,p}(a) = 1, \text{ because } a^1 \bmod p = a \quad \text{----- (2.2.4)}$$

Here is an example using a nonprime modulus, n = 9. Here Φ(n) = 6 and a = 2 is a primitive root. We compute the various powers of a and find

$2^0 = 1 \quad 2^4 \equiv 7 \pmod 9$

$2^1 = 2 \quad 2^5 \equiv 5 \pmod 9$

$2^2 = 4 \quad 2^6 \equiv 1 \pmod 9$

$2^3 = 8$

This gives us the following table of the numbers with given discrete logarithms (mod 9) for the root a = 2:

Logarithm	0	1	2	3	4	5
Number	1	2	4	8	7	5

To make it easy to obtain the discrete logarithms of a given number, we rearrange the table:

Number	1	2	4	5	7	8
Logarithm	0	1	2	5	4	3

Now consider

$$x = a^{\text{dloga,p(x)}} \bmod p \quad y = a^{\text{dloga,p(y)}} \bmod p$$

$$xy = a^{\text{dloga,p(xy)}} \bmod p$$

Using the rules of modular multiplication,

$$xy \bmod p = [(x \bmod p)(y \bmod p)] \bmod p$$

$$a^{dlog_{a,p}(xy)} \bmod p = \left[\left(a^{dlog_{a,p}(x)} \bmod p\right)\left(a^{dlog_{a,p}(y)} \bmod p\right)\right] \bmod p$$

$$= \left(a^{dlog_{a,p}(x) + dlog_{a,p}(y)}\right) \bmod p$$

But now consider Euler's theorem, which states that, for every a and n that are relatively prime:

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

Any positive integer z can be expressed in the form $z = q + k\Phi(n)$, with $0 \leq q < \Phi(n)$. Therefore, by Euler's theorem,

$$a^z \equiv a^q \pmod{n} \text{ if } z \equiv q \bmod \Phi(n)$$

Applying this to the foregoing equality, we have

$$\text{dlog}_{a,p}(xy) \equiv [\text{dlog}_{a,p}(x) + \text{dlog}_{a,p}(y)] \pmod{\Phi(p)}$$

and generalizing,

$$\text{dlog}_{a,p}(y^r) \equiv [r \times \text{dlog}_{a.p}(y)] \pmod{\Phi(n)}$$

This demonstrates the analogy between true logarithms and discrete logarithms. Keep in mind that unique discrete logarithms mod m to some base a exist only if a is a primitive root of m.

Table 2.2.1, which is directly derived from Table 2.2.1, shows the sets of discrete logarithms that can be defined for modulus 19.

(a) Discrete logarithms to the base 2, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\log_{2,19}(a)$	18	1	13	2	16	14	6	3	8	17	12	15	5	7	11	4	10	9

(b) Discrete logarithms to the base 3, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\log_{3,19}(a)$	18	7	1	14	4	8	6	3	2	11	12	15	17	13	5	10	16	9

(c) Discrete logarithms to the base 10, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\log_{10,19}(a)$	18	17	5	16	2	4	12	15	10	1	6	3	13	11	7	14	8	9

(d) Discrete logarithms to the base 13, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\log_{13,19}(a)$	18	11	17	4	14	10	12	15	16	7	6	3	1	5	13	8	2	9

(e) Discrete logarithms to the base 14, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\log_{14,19}(a)$	18	13	7	8	10	2	6	3	14	5	12	15	11	1	17	16	4	9

(f) Discrete logarithms to the base 15, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\log_{15,19}(a)$	18	5	11	10	8	16	12	15	4	13	6	3	7	17	1	2	14	9

Table 2.2.2 Tables of Discrete Logarithms, Modulo 19

Calculation of Discrete Logarithms

Consider the equation

$$y = g^x \bmod p$$

Given g, x, and p, it is a straightforward matter to calculate y. At the worst, we must perform x repeated multiplications, and algorithms exist for achieving greater efficiency.

However, given y, g, and p, it is, in general, very difficult to calculate x (take the discrete logarithm). The difficulty seems to be on the same order of magnitude as that of factoring primes required for RSA. At the

time of this writing, the asymptotically fastest known algorithm for taking discrete logarithms modulo a prime number is on the order of:

$$e^{((\ln p)^{1/3}(\ln(\ln p))^{2/3})}$$

which is not feasible for large primes.

The Complexity of Algorithms

The central issue in assessing the resistance of an encryption algorithm to cryptanalysis is the amount of time that a given type of attack will take. Typically, one cannot be sure that one has found the most efficient attack algorithm. The most that one can say is that for a particular algorithm, the level of effort for an attack is of a particular order of magnitude. One can then compare that order of magnitude to the speed of current or predicted processors to determine the level of security of a particular algorithm.

A common measure of the efficiency of an algorithm is its time complexity. We define the time complexity of an algorithm to be f(n) if, for all n and all inputs of length n, the execution of the algorithm takes at most f(n) steps. Thus, for a given size of input and a given processor speed, the time complexity is an upper bound on the execution time.

There are several ambiguities here. First, the definition of a step is not precise. A step could be a single operation of a Turing machine, a single processor machine instruction, a single high-level language machine instruction, and so on. However, these various definitions of step should all be related by simple multiplicative constants. For very large values of n, these constants are not important. What is important is how fast the relative execution time is growing. For example, if we are concerned about whether to use 50-digit ($n = 10^{50}$) or 100-digit ($n = 10^{100}$) keys for RSA, it is not necessary (or really possible) to know exactly how long it would take to break each size of key. Rather, we are interested in ballpark figures for level of effort and in knowing how much extra relative effort is required for the larger key size.

A second issue is that, generally speaking, we cannot pin down an exact formula for f(n). We can only approximate it. But again, we are primarily interested in the rate of change of f(n) as n becomes very large.

There is a standard mathematical notation, known as the "big-O" notation, for characterizing the time complexity of algorithms that is useful in this context. The definition is as follows: if and only if there exist two numbers a and M such that

$$|f(n)| \leq a \times |g(n)| \qquad n \geq M \quad \text{----- (2.2.5)}$$

Consider the following simple algorithm.

```
algorithm P1;
    n, i, j: integer; x, polyval: real;
    a, S: array [0..100] of real;
    begin
       read(x, n);
       for i := 0 upto n do
       begin
          S[i] := 1; read(a[i]);
          for j := 1 upto i do S[i] := x x S[i];
```

```
        S[i] := a[i] x S[i]
    end;
    polyval := 0;
    for i := 0 upto n do polyval := polyval + S[i];
    write ('value at', x, 'is', polyval)
end.
```

In this algorithm, each subexpression is evaluated separately. Each S[i] requires (i + 1) multiplications: i multiplications to compute S[i] and one to multiply by a[i]. Computing all n terms requires

$$\sum_{i=0}^{n}(i+1)=\frac{(n+2)(n+1)}{2}$$

multiplications. There are also (n + 1) additions, which we can ignore relative to the much larger number of multiplications. Thus, the time complexity of this algorithm is f(n) = (n + 2)(n + 1)/2. We now show that f(n) = O(n^2). From the definition of Equation (2.2.5), we want to show that for a = 1 and M = 4, the relationship holds for g(n) = n^2. We do this by induction on n. The relationship holds for n = 4 because (4 + 2) (4 +1)/2 = 15 < 4^2 = 16. Now assume that it holds for all values of n up to k [i.e.,(k + 2)(k + 1)/2 < k^2. Then, with n = k + 1.

$$\frac{(n+1)(n+2)}{2}=\frac{(k+3)(k+2)}{2}$$
$$=\frac{(k+2)(k+1)}{2}+k+2$$
$$\leq k^2+k+2$$
$$\leq k^2+2k+1=(k+1)^2=n^2$$

Therefore, the result is true for n = k + 1.

In general, the big-O notation makes use of the term that grows the fastest. For example,

1. O[$ax^7 + 3x^3$ + sin(x)] O(ax^7) = O(x^7)
2. O(e^n + an^{10}) = O(e^n)
3. O(n! + n^{50}) = O(n!)

An algorithm with an input of size n is said to be

- Linear: If the running time is O(n)
- Polynomial: If the running time is O(n^t) for some constant t
- Exponential: If the running time is O($t^{h(n)}$) for some constant t and polynomial h(n)

Generally, a problem that can be solved in polynomial time is considered feasible, whereas anything worse than polynomial time, especially exponential time, is considered infeasible. But you must be careful with these terms. First, if the size of the input is small enough, even very complex algorithms become feasible. Suppose, for example, that you have a system that can execute operations per unit time. Table 2.2.3 shows the size of input that can be handled in one time unit for algorithms of various complexities. For algorithms of exponential or factorial time, only very small inputs can be accommodated.

Complexity	Size	Operations
$\log_2 n$	$2^{1012} = 10^{3x1011}$	10^{12}
N	10^{12}	10^{12}
n^2	10^6	10^{12}
n^6	10^2	10^{12}
2^2	39	10^{12}
n!	15	10^{12}

Table 2.2.3 Level of Effort for Various Levels of Complexity

The second thing to be careful about is the way in which the input is characterized. For example, the complexity of cryptanalysis of an encryption algorithm can be characterized equally well in terms of the number of possible keys or the length of the key. For the Advanced Encryption Standard (AES), for example, the number of possible keys is 2^{128} and the length of the key is 128 bits. If we consider a single encryption to be a "step" and the number of possible keys to be $N = 2^n$, then the time complexity of the algorithm is linear in terms of the number of keys [O(N)] but exponential in terms of the length of the key [$O(2^n)$].

REFERENCES

1. William Stallings, Cryptography and Network Security, PHI Publishers
2. www.wikipedia.org

Review Questions:

1. Write short notes on algorithm complexities

3. PUBLIC KEY CRYPTOSYSTEMS

OBJECTIVE

This lesson provide an overview of public-key cryptography and concentrates on its use to provide confidentiality. First, we look at its conceptual framework. Interestingly, the concept for this technique was developed and published before it was shown to be practical to adopt it. All the key features of the Public Key cryptosystems were analysed.

INTRODUCTION

Public-key cryptography, also known as asymmetric cryptography, is a form of cryptography in which the key used to encrypt a message differs from the key used to decrypt it. In public key cryptography, a user has a pair of cryptographic keys—a **public key** and a **private key**. The private key is kept secret, while the public key may be widely distributed. Incoming messages would have been encrypted with the recipient's public key and can only be decrypted with his corresponding private key. The keys are related mathematically, but the private key cannot be practically derived from the public key.

Conversely, *secret key cryptography*, also known as symmetric cryptography, uses a single secret key for both encryption and decryption. To use symmetric cryptography for communication, both the sender & receiver would have to know the key beforehand, or it would have to be sent along with the message.

The two main branches of public key cryptography are:

- **Public key encryption** — a message encrypted with a recipient's public key cannot be decrypted by anyone except the recipient possessing the corresponding private key. This is used to ensure confidentiality.
- **Digital signatures** — a message signed with a sender's private key can be verified by anyone who has access to the sender's public key, thereby proving that the sender signed it and that the message has not been tampered with. This is used to ensure authenticity.

An analogy for public-key encryption is that of a locked mailbox with a mail slot. The mail slot is exposed and accessible to the public; its location (the street address) is in essence the public key. Anyone knowing the street address can go to the door and drop a written message through the slot; however, only the person who possesses the key can open the mailbox and read the message.

An analogy for digital signatures is the sealing of an envelope with a personal wax seal. The message can be opened by anyone, but the presence of the seal authenticates the sender.

A central problem for use of public-key cryptography is confidence (ideally proof) that a public key is correct, belongs to the person or entity claimed (ie, is 'authentic'), and has not been tampered with or replaced by a malicious third party. The usual approach to this problem is to use a

public-key infrastructure (PKI), in which one or more third parties, known as certificate authorities, certify ownership of key pairs. Another approach, used by PGP, is the "web of trust" method to ensure authenticity of key pairs.

So far, public key techniques have been much more computationally intensive than purely symmetric algorithms. The judicious use of these techniques enables a wide variety of applications without incurring a prohibitive computational penalty. In practice, public key cryptography is often used in combination with secret-key methods for efficiency reasons. Such a combination is called a hybrid cryptosystem. For encryption, the sender encrypts the message with a secret-key algorithm using a randomly generated key, and that random key is then encrypted with the recipient's public key. For digital signatures, the sender hashes the message (using a cryptographic hash function) and then signs the resulting "hash value". Before verifying the signature, the recipient also computes the hash of the message, and compares this hash value with the signed hash value to check that the message has not been tampered with.

Public-Key Cryptosystems

Asymmetric algorithms rely on one key for encryption and a different but related key for decryption. These algorithms have the following important characteristic:

- It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key.

In addition, some algorithms, such as RSA, also exhibit the following characteristic:

- Either of the two related keys can be used for encryption, with the other used for decryption.

A public-key encryption scheme has six ingredients (Figure 2.3.1a)

- **Plaintext:** This is the readable message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The encryption algorithm performs various transformations on the plaintext.
- **Public and private keys:** This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the algorithm depend on the public or private key that is provided as input.
- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.
- **Decryption algorithm:** This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

The essential steps are the following:

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.
2. Each user places one of the two keys in a public register or other

accessible file. This is the public key. The companion key is kept private. As Fig 2.3.1a suggests, each user maintains a collection of public keys obtained from others.

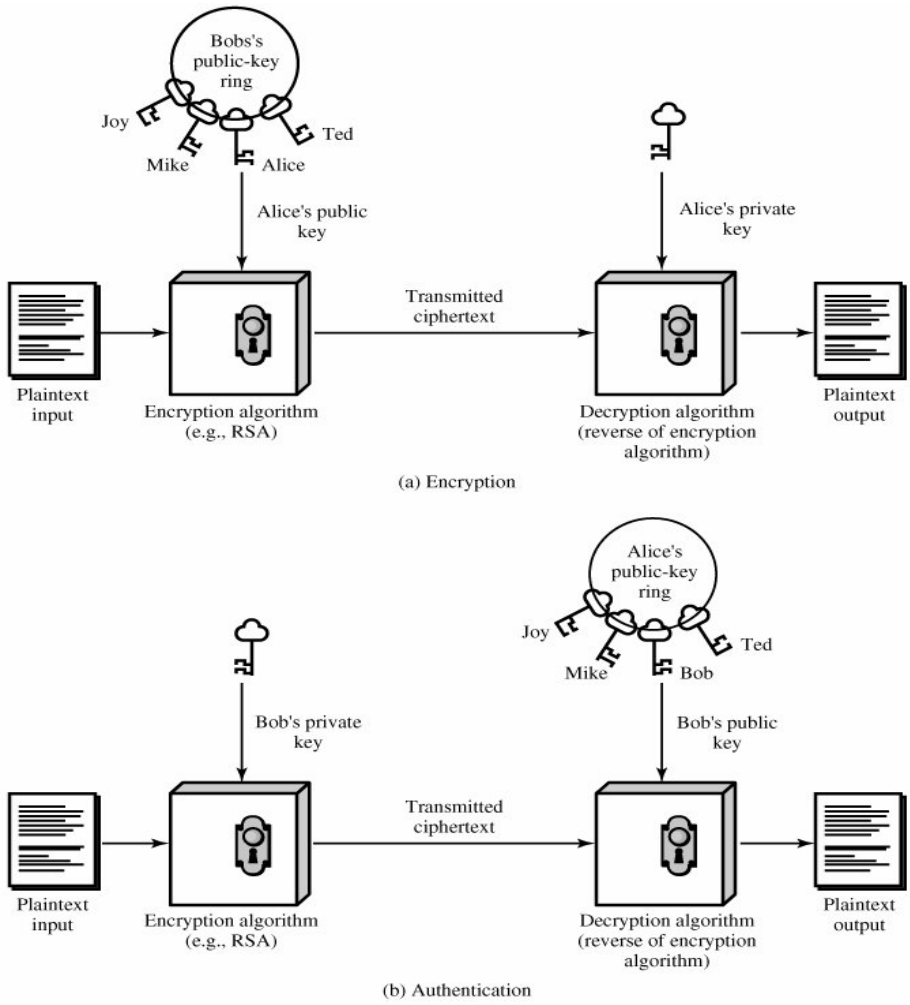


Fig 2.3.1 Public-Key Cryptography

3. If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key.

4. When Alice receives the message, she decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice's private key.

With this approach, all participants have access to public keys, and private keys are generated locally by each participant and therefore need never be distributed. As long as a user's private key remains protected and secret, incoming communication is secure. At any time, a system can change its private key and publish the companion public key to replace its old public key.

Table 2.3.1 summarizes some of the important aspects of symmetric and public-key encryption. To discriminate between the two, we refer to the key used in symmetric encryption as a secret key. The two keys used for asymmetric encryption are referred to as the **public key** and the **private key**. Invariably, the private key is kept secret, but it is referred to as a private key rather than a secret key to avoid confusion with symmetric encryption.

Conventional Encryption	**Public-Key Encryption**
Needed to Work:	**Needed to Work:**
1. The same algorithm with the same key is used for encryption and decryption. 2. The sender and receiver must share the algorithm and the key.	1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption. 2. The sender and receiver must each have one of the matched pair of keys (not the same one).
Needed for Security:	**Needed for Security:**
1. The key must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	1. One of the two keys must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

Table 2.3.1 Conventional and Public-Key Encryption

Let us take a closer look at the essential elements of a public-key encryption scheme, using Figure 2.3.2. There is some source A that produces a message in plaintext, $X = [X_1, X_2, ..., X_M,]$. The M elements of X are letters in some finite alphabet. The message is intended for destination B. B generates a related pair of keys: a public key, PU_b, and a private key, PU_b. PU_b is known only to B, whereas PU_b is publicly available and therefore accessible by A.

With the message X and the encryption key PU_b as input, A forms the ciphertext $Y = [Y_1, Y_2, ..., Y_N]$:

$$\mathbf{Y = E(PU_b, X)}$$

The intended receiver, in possession of the matching private key, is able to invert the transformation:

$$\mathbf{X = D(PR_b, Y)}$$

An adversary, observing Y and having access to PU_b but not having access to PR_b or X, must attempt to recover X and/or PR_b. It is assumed that the adversary does have knowledge of the encryption (E) and decryption (D) algorithms. If the adversary is interested only in this particular message, then the focus of effort is to recover X, by generating

a plaintext estimate $\hat{X}$ Often, however, the adversary is interested in being able to read future messages as well, in which case an attempt is made to recover PR_b by generating an estimate $\hat{PR}_b$.

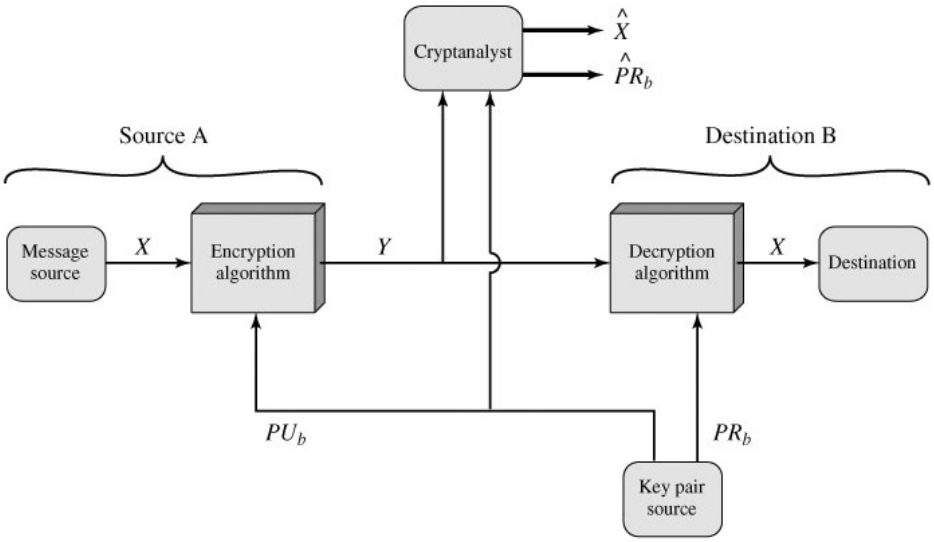


Fig 2.3.2 Public-Key Cryptosystem: Secrecy

We mentioned earlier that either of the two related keys can be used for encryption, with the other being used for decryption. This enables a rather different cryptographic scheme to be implemented. Whereas the scheme illustrated in Fig 2.3.2 provides confidentiality, Fig 2.3.1b and 2.3.3 show the use of public-key encryption to provide authentication:

$$\mathbf{Y = E(PR_a, X)}$$

$$\mathbf{Y = E(PU_a, Y)}$$

In this case, A prepares a message to B and encrypts it using A's private key before transmitting it. B can decrypt the message using A's public key. Because the message was encrypted using A's private key, only A could have prepared the message. Therefore, the entire encrypted message serves as a digital signature. In addition, it is impossible to alter the message without access to A's private key, so the message is authenticated both in terms of source and in terms of data integrity.

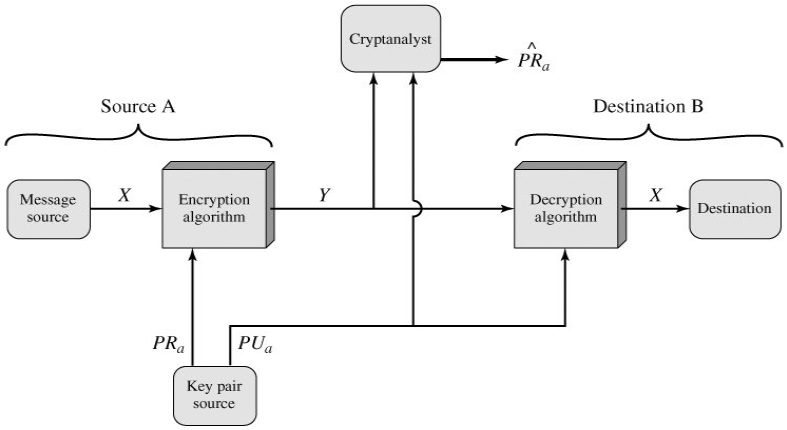


Fig 2.3.3 Public-Key Cryptosystem: Authentication

In the preceding scheme, the entire message is encrypted although validating both author and contents, requires a great deal of storage. Each document must be kept in plaintext to be used for practical purposes. A copy also must be stored in ciphertext so that the origin and contents can be verified in case of a dispute. A more efficient way of achieving the same results is to encrypt a small block of bits that is a function of the document. Such a block, called an authenticator, must have the property that it is infeasible to change the document without changing the authenticator. If the authenticator is encrypted with the sender's private key, it serves as a signature that verifies origin, content, and sequencing.

It is important to emphasize that the encryption process depicted in Fig 2.3.1b and 2.3.3 does not provide confidentiality. That is, the message being sent is safe from alteration but not from eavesdropping. This is obvious in the case of a signature based on a portion of the message, because the rest of the message is transmitted in the clear.

It is possible to provide both the authentication function and confidentiality by a double use of the public-key scheme (Fig 2.3.4):

$$Z = E(PU_b, E(PR_a, X))$$
$$X = D(PU_a, E(PR_b, Z))$$

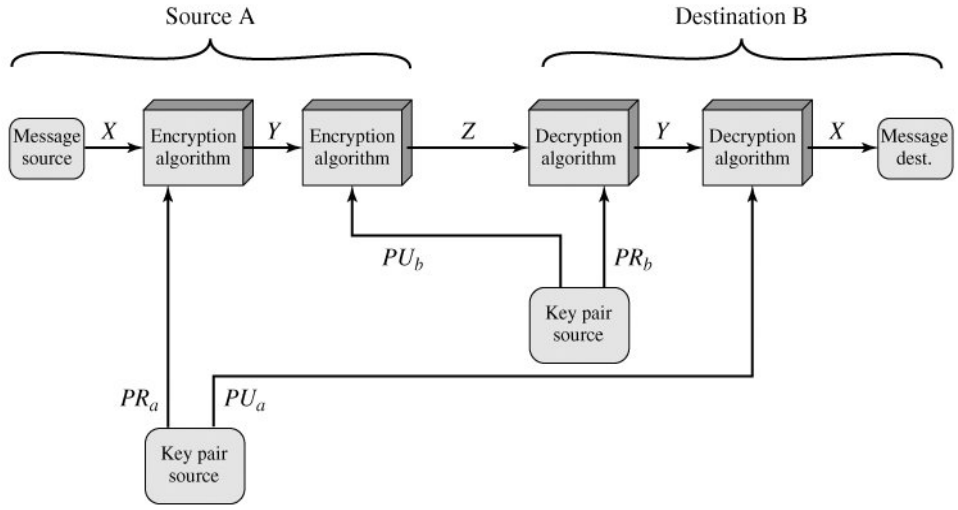


Fig 2.3.4 Public-Key Cryptosystem: Authentication and Secrecy

In this case, we begin as before by encrypting a message, using the sender's private key. This provides the digital signature. Next, we encrypt again, using the receiver's public key. The final ciphertext can be decrypted only by the intended receiver, who alone has the matching private key. Thus, confidentiality is provided. 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.

Applications for Public-Key Cryptosystems

Before proceeding, we need to clarify one aspect of public-key cryptosystems that is otherwise likely to lead to confusion. Public-key systems are characterized by the use of a cryptographic algorithm with two keys, one held private and one available publicly. Depending on the

application, the sender uses either the sender's private key or the receiver's public key, or both, to perform some type of cryptographic function. In broad terms, we can classify the use of public-key cryptosystems into three categories:

- Encryption/decryption: The sender encrypts a message with the recipient's public key.
- Digital signature: The sender "signs" a message with its private key. Signing is achieved by a cryptographic algorithm applied to the message or to a small block of data that is a function of the message.
- **Key exchange:** Two sides cooperate to exchange a session key. Several different approaches are possible, involving the private key(s) of one or both parties.

Some algorithms are suitable for all three applications, whereas others can be used only for one or two of these applications. Table 2.3.2 indicates the applications supported by the algorithms discussed in this book.

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

Table 2.3.2 Applications for Public-Key Cryptosystems

Requirements for Public-Key Cryptography

The cryptosystem illustrated in Figures 2.3.2 through 2.3.4 depends on a cryptographic algorithm based on two related keys. Diffie and Hellman postulated this system without demonstrating that such algorithms exist.

The conditions that algorithms must fulfill:

1. It is computationally easy for a party B to generate a pair (public key PU_b, private key PR_b).
2. It is computationally easy for a sender A, knowing the public key and the message to be encrypted, M, to generate the corresponding ciphertext:

$$\mathbf{C = E(PU_b, M)}$$

3. It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message:

$$\mathbf{M = D(PR_b, C) = D[PR_b, E(PU_b, M)]}$$

4. It is computationally infeasible for an adversary, knowing the public key, PU_b, to determine the private key, PR_b.
5. It is computationally infeasible for an adversary, knowing the public key, PU_b, and a ciphertext, C, to recover the original message, M.

We can add a sixth requirement that, although useful, is not necessary for all public-key applications:

6. The two keys can be applied in either order:

$$M = D[PU_b, E(PR_b, M)] = D[PR_b, E(PU_b, M)]$$

These are formidable requirements, as evidenced by the fact that only a few algorithms (RSA, elliptic curve cryptography, Diffie-Hellman, DSS) have received widespread acceptance in the several decades since the concept of public-key cryptography was proposed.

The requirements boil down to the need for a trap-door one-way function. A *one-way function* is one that maps a domain into a range such that every function value has a unique inverse, with the condition that the calculation of the function is easy whereas the calculation of the inverse is infeasible:

$$Y = f(X) \quad \text{easy}$$
$$X = f^{1}(X) \quad \text{infeasible}$$

Generally, easy is defined to mean a problem that can be solved in polynomial time as a function of input length. Thus, if the length of the input is n bits, then the time to compute the function is proportional to n^a where a is a fixed constant. Such algorithms are said to belong to the class P. The term infeasible is a much fuzzier concept. In general, we can say a problem is infeasible if the effort to solve it grows faster than polynomial time as a function of input size. For example, if the length of the input is n bits and the time to compute the function is proportional to 2^n, the problem is considered infeasible. Unfortunately, it is difficult to determine if a particular algorithm exhibits this complexity. Furthermore, traditional notions of computational complexity focus on the worst-case or average-case complexity of an algorithm. These measures are inadequate for cryptography, which requires that it be infeasible to invert a function for virtually all inputs, not for the worst case or even average case.

The trap-door one-way function is the one which is easy to calculate in one direction and infeasible to calculate in the other direction unless certain additional information is known. With the additional information the inverse can be calculated in polynomial time. We can summarize as follows: A trap-door one-way function is a family of invertible functions f_k, such that

$Y = f_k(X)$ easy, if k and X are known

$X = f_k^{-1}(Y)$ easy, if k and Y are known

$X = f_k^{-1}(Y)$ infeasible, if Y is known but k is not known

Thus, the development of a practical public-key scheme depends on discovery of a suitable trap-door one-way function.

Public-Key Cryptanalysis

As with symmetric encryption, a public-key encryption scheme is vulnerable to a brute-force attack. The countermeasure is the same: Use large keys. However, there is a tradeoff to be considered. Public-key systems depend on the use of some sort of invertible mathematical function. The complexity of calculating these functions may not scale linearly with the number of bits in the key but grow more rapidly than that.

Thus, the key size must be large enough to make brute-force attack impractical but small enough for practical encryption and decryption. In practice, the key sizes that have been proposed do make brute-force attack impractical but result in encryption/decryption speeds that are too slow for general-purpose use. Instead, as was mentioned earlier, public-key encryption is currently confined to key management and signature applications.

Another form of attack is to find some way to compute the private key given the public key. To date, it has not been mathematically proven that this form of attack is infeasible for a particular public-key algorithm. Thus, any given algorithm, including the widely used RSA algorithm, is suspect. The history of cryptanalysis shows that a problem that seems insoluble from one perspective can be found to have a solution if looked at in an entirely different way.

Finally, there is a form of attack that is peculiar to public-key systems. This is, in essence, a probable-message attack. Suppose, for example, that a message were to be sent that consisted solely of a 56-bit DES key. An adversary could encrypt all possible 56-bit DES keys using the public key and could discover the encrypted key by matching the transmitted ciphertext. Thus, no matter how large the key size of the public-key scheme, the attack is reduced to a brute-force attack on a 56-bit key. This attack can be thwarted by appending some random bits to such simple messages.

REFERENCES

1. William Stallings, Cryptography and Network Security, PHI Publishers
2. www.wikipedia.org

Review Questions:

1. Write short notes on public-key Cryptosystems
2. Explain the model for Network security?
   a. Secrecy
   b. Authentication
   c. Secrecy and Authentication
3. What is the need for Public Key cryptosystems?

4. RSA AND DIFFIE-HELLMAN ALGORITHMS

OBJECTIVE

The objective of this lesson is to introduce to the user one of the public-key cryptosystems namely RSA. Also, we examine the RSA algorithm, which is the most important encryption/decryption algorithm that has been shown to be feasible for public-key encryption.

The RSA Algorithm

This was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and first published in 1978. The Rivest-Shamir-Adleman (RSA) scheme has since that time reigned supreme as the most widely accepted and implemented general-purpose approach to public-key encryption.

The RSA scheme is a block cipher in which the plaintext and ciphertext are integers between 0 and n 1 for some n. A typical size for n is 1024 bits, or 309 decimal digits. That is, n is less than 2^{1024}. We examine RSA in this section in some detail, beginning with an explanation of the algorithm. Then we examine some of the computational and cryptanalytical implications of RSA.

Description of the Algorithm

The scheme developed by Rivest, Shamir, and Adleman makes use of an expression with exponentials. Plaintext is encrypted in blocks, with each block having a binary value less than some number n. That is, the block size must be less than or equal to $\log_2(n)$; in practice, the block size is i bits, where $2^i < n \leq 2^{i+1}$. Encryption and decryption are of the following form, for some plaintext block M and ciphertext block C:

$$\mathbf{C = M^e \bmod n}$$

$$\mathbf{M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n}$$

Both sender and receiver must know the value of n. The sender knows the value of e, and only the receiver knows the value of d. Thus, this is a public-key encryption algorithm with a public key of PU = {e, n} and a private key of PU = {d, n}. For this algorithm to be satisfactory for public-key encryption, the following requirements must be met:

1. It is possible to find values of e, d, n such that M^{ed} mod n = M for all M < n.
2. It is relatively easy to calculate mod M^e mod n and C^d for all values of M < n.
3. It is infeasible to determine d given e and n.

For now, we focus on the first requirement and consider the other questions later. We need to find a relationship of the form

$$\mathbf{M^{ed} \bmod n = M}$$

The preceding relationship holds if e and d are multiplicative inverses modulo Φ(n), where Φ(n) is the Euler totient function. We know that for p,

q prime, Φ (pq) = (p 1)(q 1) The relationship between e and d can be expressed as

ed mod Φ(n) = 1 ------ (2.4.1)

This is equivalent to saying

ed ≡ 1 mod Φ(n)

d ≡ e^1 mod Φ(n)

That is, e and d are multiplicative inverses mod Φ(n). Note that, according to the rules of modular arithmetic, this is true only if d (and therefore e) is relatively prime to Φ(n). Equivalently, gcd(Φ(n),d) = 1.

The ingredients OF RSA scheme are the following:

p,q, two prime numbers	**(private, chosen)**
n = pq	**(public, calculated)**
e, with gcd(Φ(n),e) = 1;1 < e < Φ(n)	**(public, chosen)**
d ≡ e^1(mod Φ(n))	**(private, calculated)**

The private key consists of {d, n} and the public key consists of {e, n}. Suppose that user A has published its public key and that user B wishes to send the message M to A. Then B calculates C = M^e mod n and transmits C. On receipt of this ciphertext, user A decrypts by calculating

M = C^d mod n.

Key Generation

Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

Encryption

Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption

Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

Fig 2.4.1 The RSA Algorithm

Fig 2.4.1 summarizes the RSA algorithm. An example is shown in Fig 2.4.2. For this example, the keys were generated as follows:

1. Select two prime numbers, p = 17 and q = 11.
2. Calculate n = pq = 17 x 11 = 187.
3. Calculate Φ(n) = (p 1)(q 1) = 16 x 10 = 160.
4. Select e such that e is relatively prime to Φ(n) = 160 and less than Φ(n) we choose e = 7.
5. Determine d such that de ≡ 1 (mod 160) and d < 160. The correct value is d = 23, because 23 x 7 = 161 = 10 x 160 + 1; d can be calculated using the extended Euclid's algorithm.

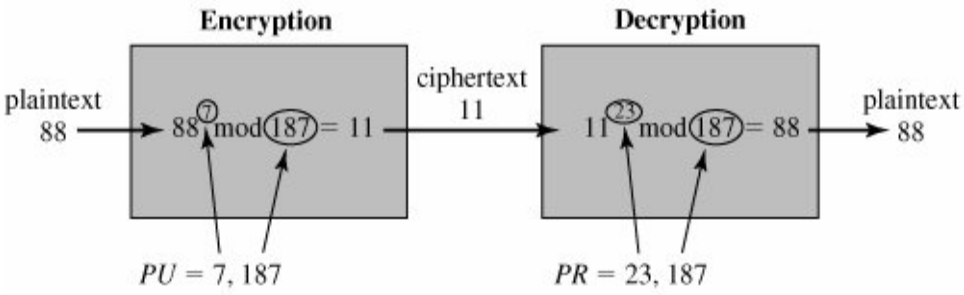


Fig 2.4.2 Example of RSA Algorithm

The resulting keys are public key PU = {7,187} and private key PR = {23,187}. The example shows the use of these keys for a plaintext input of M = 88. For encryption, we need to calculate

C = 88^7 mod 187. Exploiting the properties of modular arithmetic, we can do this as follows:

88^7 mod 187 = [(88^4 mod 187) x (88^2 mod 187) x (88^1 mod 187)] mod 187

88^1 mod 187 = 88

88^2 mod 187 = 7744 mod 187 = 77

88^4 mod 187 = 59,969,536 mod 187 = 132

88^7 mod 187 = (88 x 77 x 132) mod 187 = 894,432 mod 187 = 11

For decryption, we calculate M = 11^{23} mod 187:

11^{23} mod 187 = [(11^1 mod 187) x (11^2 mod 187) x (11^4 mod 187) x (11^8 mod 187) x

(11^8 mod 187)] mod 187

11^1 mod 187 = 11

11^2 mod 187 = 121

11^4 mod 187 = 14,641 mod 187 = 55

11^8 mod 187 = 214,358,881 mod 187 = 33

11^{23} mod 187 = (11 x 121 x 55 x 33 x 33) mod 187 = 79,720,245 mod 187 = 88

Computational Aspects

We now turn to the issue of the complexity of the computation required to use RSA. There are actually two issues to consider: encryption/decryption and key generation. Let us look first at the process of encryption and decryption and then consider key generation.

Exponentiation in Modular Arithmetic

Both encryption and decryption in RSA involve raising an integer to an integer power, mod n. If the exponentiation is done over the integers and then reduced modulo n, the intermediate values would be gargantuan. Fortunately, as the preceding example shows, we can make use of a property of modular arithmetic:

$$[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$$

Thus, we can reduce intermediate results modulo n. This makes the calculation practical.

Another consideration is the efficiency of exponentiation, because with RSA we are dealing with potentially large exponents. To see how efficiency might be increased, consider that we wish to compute x^{16}. A straightforward approach requires 15 multiplications:

$$x^{16} = x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x$$

However, we can achieve the same final result with only four multiplications if we repeatedly take the square of each partial result, successively forming x^2, x^4, x^8, x^{16}. As another example, suppose we wish to calculate x^{11} mod n for some integers x and n. Observe that $x^{11} = x^{1+2+8} = (x)(x^2)(x^8)$. In this case we compute x mod n, x^2 mod n, x^4 mod n, and x^8 mod n and then calculate [(x mod n) x (x^2 mod n) x (x^8 mod n) mod n.

More generally, suppose we wish to find the value a^b with a and b positive integers. If we express b as a binary number $b_k b_{k1} \ldots b_0$ then we have

$$b = \sum_{b_i \neq 0} 2^i$$

Therefore,

$$a^b = a^{\left(\sum_{b_i \neq 0} 2^i\right)}$$

$$= \prod_{b_i \neq 0} a^{(2^i)}$$

$$a^b \bmod n = \left[\prod_{b_i \neq 0} a^{(2^i)}\right] \bmod n$$

$$= \left(\left[\prod_{b_i \neq 0} a^{(2^i)} \bmod n\right]\right) \bmod n$$

We can therefore develop the algorithm for computing a^b mod n, shown in Fig 2.4.3. Table 2.4.1 shows an example of the execution of this algorithm. Note that the variable c is not needed; it is included for explanatory purposes. The final value of c is the value of the exponent.

```
c ← 0; f ← 1
for i ← k downto 0
    do  c ← 2 × c
        f ← (f × f) mod n
    if  b_i = 1
        then c ← c + 1
             f ← (f × a) mod n
return f
```

Fig 2.4.3 Algorithm for computing a^b mod n

Note: The integer b is expressed as a binary number $b_k b_k 1 \ldots b_0$

i	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
b_i	1	0	0	0	1	1	0	0	0	0
c	1	2	4	8	17	35	70	140	280	560
f	7	49	157	526	160	241	298	166	67	1

Table 2.4.1 Result of the Fast Modular Exponentiation Algorithm for a^b mod n, where a = 7, b = 560 = 1000110000, n = 561

Efficient Operation Using the Public Key

To speed up the operation of the RSA algorithm using the public key, a specific choice of e is usually made. The most common choice is 65537 (2^{16} 1); two other popular choices are 3 and 17. Each of these choices has only two 1 bits and so the number of multiplications required to perform exponentiation is minimized.

However, with a very small public key, such as e = 3, RSA becomes vulnerable to a simple attack. Suppose we have three different RSA users who all use the value e = 3 but have unique values of n, namely n_1, n_2, n_3. If user A sends the same encrypted message M to all three users, then the three ciphertexts are $C_1 = M^3 \bmod n_1$; $C_2 = M^3 \bmod n_2$; $C_3 = M^3 \bmod n_3$. It is likely that n_1, n_2, and n_3 are pairwise relatively prime. Therefore, one can use the Chinese remainder theorem (CRT) to compute $M^3 \bmod (n_1 n_2 n_3)$. By the rules of the RSA algorithm, M is less than each of the n_i therefore $M^3 < n_1 n_2 n_3$. Accordingly, the attacker need only compute the cube root of M^3. This attack can be countered by adding a unique pseudorandom bit string as padding to each instance of M to be encrypted. This approach is discussed subsequently.

The reader may have noted that the definition of the RSA algorithm (Fig 2.4.1) requires that during key generation the user selects a value of e that is relatively prime to Φ(n). Thus, for example, if a user has preselected e = 65537 and then generated primes p and q, it may turn

out that gcd(Φ(n),e) ≠ 1, Thus, the user must reject any value of p or q that is not congruent to 1 (mod 65537).

Efficient Operation Using the Private Key

We cannot similarly choose a small constant value of d for efficient operation. A small value of d is vulnerable to a brute-force attack and to other forms of cryptanalysis. However, there is a way to speed up computation using the CRT. We wish to compute the value $M = C^d \bmod n$. Let us define the following intermediate results:

$$V_p = C^d \bmod p \quad V_q = C^d \bmod q$$

Following the CRT (Chinese Remainder Theorem), define the quantities:

$$X_p = q \times (q^1 \bmod p) \quad X_q = p \times (p^1 \bmod q)$$

The CRT then shows that

$$M = (V_p X_p + V_q X_q) \bmod n$$

Further, we can simplify the calculation of V_p and V_q using Fermat's theorem, which states that $a^{p1} \equiv 1 \pmod p$ if p and a are relatively prime. Some thought should convince you that the following are valid:

$$V_p = C^d \bmod p = C^{d \bmod (p1)} \bmod p \quad V_q = C^d \bmod q = C^{d \bmod (q1)} \bmod q$$

The quantities d mod (P1) and d mod (q1) can be precalculated. The end result is that the calculation is approximately four times as fast as evaluating $M = C^d \bmod n$ directly.

Key Generation

Before the application of the public-key cryptosystem, each participant must generate a pair of keys. This involves the following tasks:

- Determining two prime numbers, p and q
- Selecting either e or d and calculating the other

First, consider the selection of p and q. Because the value of n = pq will be known to any potential adversary, to prevent the discovery of p and q by exhaustive methods, these primes must be chosen from a sufficiently large set (i.e., p and q must be large numbers). On the other hand, the method used for finding large primes must be reasonably efficient.

At present, there are no useful techniques that yield arbitrarily large primes, so some other means of tackling the problem is needed. The procedure that is generally used is to pick at random an odd number of the desired order of magnitude and test whether that number is prime. If not, pick successive random numbers until one is found that tests prime.

A variety of tests for primality have been developed. Almost invariably, the tests are probabilistic. That is, the test will merely determine that a given integer is probably prime. Despite this lack of certainty, these tests can be run in such a way as to make the probability as close to 1.0 as desired. As an example, one of the more efficient and popular algorithms, the Miller-Rabin algorithm. With this algorithm and most such algorithms, the procedure for testing whether a given integer n is prime is to perform some calculation that involves n and a randomly chosen integer a. If n "fails" the test, then n is not prime. If n "passes" the test, then n may be prime or nonprime. If n passes many such tests with many different

randomly chosen values for a, then we can have high confidence that n is, in fact, prime.

In summary, the procedure for picking a prime number is as follows.

1. Pick an odd integer n at random (e.g., using a pseudorandom number generator).
2. Pick an integer a < n at random.
3. Perform the probabilistic primality test, such as Miller-Rabin, with a as a parameter. If n fails the test, reject the value n and go to step 1.
4. If n has passed a sufficient number of tests, accept n; otherwise, go to step 2.

This is a somewhat tedious procedure. However, remember that this process is performed relatively infrequently: only when a new pair (PU, PR) is needed.

It is worth noting how many numbers are likely to be rejected before a prime number is found. A result from number theory, known as the prime number theorem, states that the primes near N are spaced on the average one every (ln N) integers. Thus, on average, one would have to test on the order of ln(N) integers before a prime is found. Actually, because all even integers can be immediately rejected, the correct figure is ln(N)/2. For example, if a prime on the order of magnitude of 2^{200} were sought, then about $\ln(2^{200})/2 = 70$ trials would be needed to find a prime.

Having determined prime numbers p and q, the process of key generation is completed by selecting a value of e and calculating d or, alternatively, selecting a value of d and calculating e. Assuming the former, then we need to select an e such that $\gcd(\Phi(n), e) = 1$ and then calculate $d \equiv e^{1} \pmod{\Phi(n)}$. Fortunately, there is a single algorithm that will, at the same time, calculate the greatest common divisor of two integers and, if the gcd is 1, determine the inverse of one of the integers modulo the other. Thus, the procedure is to generate a series of random numbers, testing each against Φ(n) until a number relatively prime to Φ(n) is found. It can be shown easily that the probability that two random numbers are relatively prime is about 0.6; thus, very few tests would be needed to find a suitable integer.

The Security of RSA

Four possible approaches to attacking the RSA algorithm are as follows:

- **Brute force:** This involves trying all possible private keys.
- **Mathematical attacks:** There are several approaches, all equivalent in effort to factoring the product of two primes.
- **Timing attacks**: These depend on the running time of the decryption algorithm.
- **Chosen ciphertext attacks:** This type of attack exploits properties of the RSA algorithm.

The defense against the brute-force approach is the same for RSA as for other cryptosystems, namely, use a large key space. Thus, the larger the number of bits in d, the better. However, because the calculations

involved, both in key generation and in encryption/decryption, are complex, the larger the size of the key, the slower the system will run.

In this subsection, we provide an overview of mathematical and timing attacks.

The Factoring Problem

We can identify three approaches to attacking RSA mathematically:

- Factor n into its two prime factors. This enables calculation of $\Phi(n) = (p\ 1) \times (q\ 1)$, which, in turn, enables determination of $d \equiv e^{1} \pmod{\Phi(n)}$.
- Determine $\Phi(n)$ directly, without first determining p and q. Again, this enables determination of $d \equiv e^{1} \pmod{\Phi(n)}$.
- Determine d directly, without first determining $\Phi(n)$.

For a large n with large prime factors, factoring is a hard problem.

Number of Decimal Digits	Approximate Number of Bits	Date Achieved	MIPS-years	Algorithm
100	332	April 1991	7	Quadratic sieve
110	365	April 1992	75	Quadratic sieve
120	398	June 1993	830	Quadratic sieve
129	428	April 1994	5000	Quadratic sieve
130	431	April 1996	1000	Generalized number field sieve
140	465	February 1999	2000	Generalized number field sieve
155	512	August 1999	8000	Generalized number field sieve
160	530	April 2003		Lattice sieve
174	576	December 2003		Lattice sieve
200	663	May 2005		Lattice sieve

Table 2.4.2 Progress in Factorization

A striking fact about Table 2.4.2 concerns the method used. Until the mid-1990s, factoring attacks were made using an approach known as the quadratic sieve. The attack on RSA-130 used a newer algorithm, the generalized number field sieve (GNFS), and was able to factor a larger number than RSA-129 at only 20% of the computing effort.

The threat to larger key sizes is twofold: the continuing increase in computing power, and the continuing refinement of factoring algorithms. We have seen that the move to a different algorithm resulted in a tremendous speedup. We can expect further refinements in the GNFS,

and the use of an even better algorithm is also a possibility. In fact, a related algorithm, the special number field sieve (SNFS), can factor numbers with a specialized form considerably faster than the generalized number field sieve. Fig 2.4.4 compares the performance of the two algorithms. It is reasonable to expect a breakthrough that would enable a general factoring performance in about the same time as SNFS, or even better. Thus, we need to be careful in choosing a key size for RSA. For the near future, a key size in the range of 1024 to 2048 bits seems reasonable.

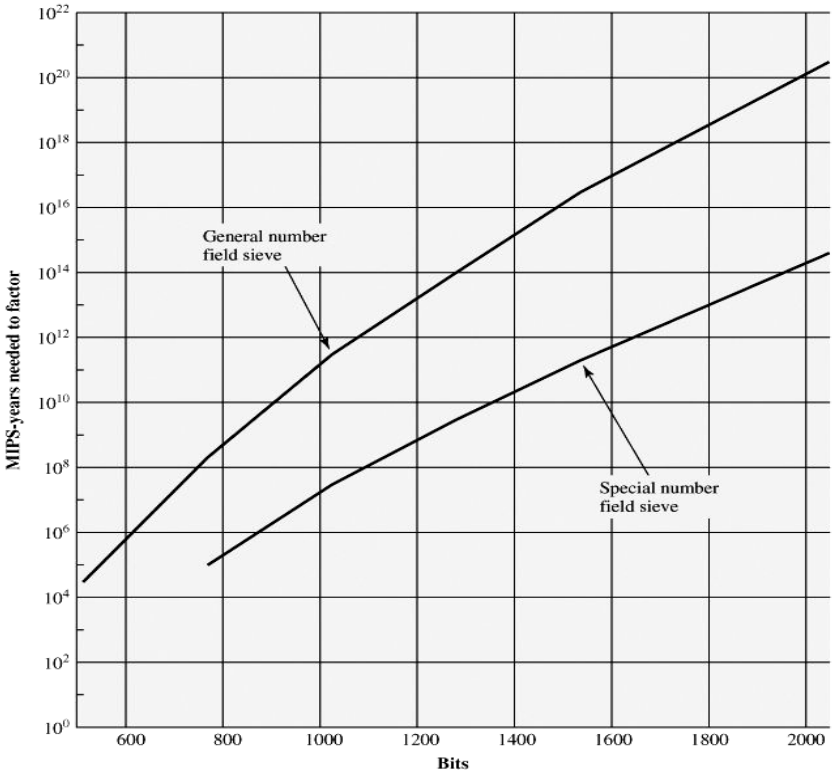


Fig 2.4.4 MIPS-years Needed to Factor

In addition to specifying the size of n, a number of other constraints have been suggested by researchers. To avoid values of n that may be factored more easily, the algorithm's inventors suggest the following constraints on p and q:

1. p and q should differ in length by only a few digits. Thus, for a 1024-bit key (309 decimal digits), both p and q should be on the order of magnitude of 10^{75} to 10^{100}.
2. Both (p 1) and (q 1) should contain a large prime factor.
3. gcd(p 1, q 1) should be small.

In addition, it has been demonstrated that if $e < n$ and $d < n^{1/4}$, then d can be easily determined.

Timing Attacks

If one needed yet another lesson about how difficult it is to assess the security of a cryptographic algorithm, the appearance of timing attacks provides a stunning one. Paul Kocher, a cryptographic consultant, demonstrated that a snooper can determine a private key by keeping track of how long a computer takes to decipher messages. Timing attacks

are applicable not just to RSA, but to other public-key cryptography systems. This attack is alarming for two reasons: It comes from a completely unexpected direction and it is a ciphertext-only attack.

A timing attack is somewhat analogous to a burglar guessing the combination of a safe by observing how long it takes for someone to turn the dial from number to number. We can explain the attack using the modular exponentiation algorithm of Fig 2.4.3, but the attack can be adapted to work with any implementation that does not run in fixed time. In this algorithm, modular exponentiation is accomplished bit by bit, with one modular multiplication performed at each iteration and an additional modular multiplication performed for each 1 bit.

The attack is simplest to understand in an extreme case. Suppose the target system uses a modular multiplication function that is very fast in almost all cases but in a few cases takes much more time than an entire average modular exponentiation. The attack proceeds bit-by-bit starting with the leftmost bit, b_k. Suppose that the first j bits are known (to obtain the entire exponent, start with j = 0 and repeat the attack until the entire exponent is known). For a given ciphertext, the attacker can complete the first j iterations of the for loop. The operation of the subsequent step depends on the unknown exponent bit. If the bit is set, d (d x a) mod n will be executed. For a few values of a and d, the modular multiplication will be extremely slow, and the attacker knows which these are. Therefore, if the observed time to execute the decryption algorithm is always slow when this particular iteration is slow with a 1 bit, then this bit is assumed to be 1. If a number of observed execution times for the entire algorithm are fast, then this bit is assumed to be 0.

In practice, modular exponentiation implementations do not have such extreme timing variations, in which the execution time of a single iteration can exceed the mean execution time of the entire algorithm. Nevertheless, there is enough variation to make this attack practical. Although the timing attack is a serious threat, there are simple countermeasures that can be used, including the following:

- Constant exponentiation time: Ensure that all exponentiations take the same amount of time before returning a result. This is a simple fix but does degrade performance.
- Random delay: Better performance could be achieved by adding a random delay to the exponentiation algorithm to confuse the timing attack. Kocher points out that if defenders don't add enough noise, attackers could still succeed by collecting additional measurements to compensate for the random delays.
- Blinding: Multiply the ciphertext by a random number before performing exponentiation. This process prevents the attacker from knowing what ciphertext bits are being processed inside the computer and therefore prevents the bit-by-bit analysis essential to the timing attack.

RSA Data Security incorporates a blinding feature into some of its products. The private-key operation $M = C^d \bmod n$ is implemented as follows:

1. Generate a secret random number r between 0 and n 1.
2. Compute $C' = C(r^e) \bmod n$, where e is the public exponent.

3. Compute M' = $(C')^d$ mod n with the ordinary RSA implementation.
4. Compute M = M'r^1 mod n. In this equation, r^1 is the multiplicative inverse of r mod n; see

It can be demonstrated that this is the correct result by observing that r^{ed} mod n =r mod n.

RSA Data Security reports a 2 to 10% performance penalty for blinding.

Chosen Ciphertext Attack and Optimal Asymmetric Encryption Padding

The basic RSA algorithm is vulnerable to a chosen ciphertext attack (CCA). CCA is defined as an attack in which adversary chooses a number of ciphertexts and is then given the corresponding plaintexts, decrypted with the target's private key. Thus, the adversary could select a plaintext, encrypt it with the target's public key and then be able to get the plaintext back by having it decrypted with the private key. Clearly, this provides the adversary with no new information. Instead, the adversary exploits properties of RSA and selects blocks of data that, when processed using the target's private key, yield information needed for cryptanalysis.

A simple example of a CCA against RSA takes advantage of the following property of RSA:

$$E(PU, M_1) \; X \; E(PU, M_2) = E(PU, [M_1 \; X \; M_2])$$

We can decrypt C = M^e using a CCA as follows.

1. Compute X = (C x 2^e) mod n.
2. Submit X as a chosen ciphertext and receive back Y = X^d mod n. But now note the following:

$$\begin{aligned} X &= (C \bmod n) \times (2^e \bmod n) \\ &= (M^e \bmod n) \times (2^e \bmod n) \\ &= (2M)^e \bmod n \end{aligned}$$

Therefore, Y = (2M) mod n From this, we can deduce M. To overcome this simple attack, practical RSA-based cryptosystems randomly pad the plaintext prior to encryption. This randomizes the ciphertext so that Equation (2.4.2) no longer holds. However, more sophisticated CCAs are possible and a simple padding with a random value has been shown to be insufficient to provide the desired security. To counter such attacks RSA Security Inc., a leading RSA vendor and former holder of the RSA patent, recommends modifying the plaintext using a procedure known as optimal asymmetric encryption padding (OAEP).

Figure 2.4.5 depicts OAEP encryption. As a first step the message M to be encrypted is padded. A set of optional parameters P is passed through a hash function H. The output is then padded with zeros to get the desired length in the overall data block (DB). Next, a random seed is generated and passed through another hash function, called the mask generating function (MGF). The resulting hash value is bit-by-bit XORed with DB to produce a maskedDB. The maskedDB is in turn passed through the MGF to form a hash that is XORed with the seed to produce

the masked seed. The concatenation of the maskedseed and the maskedDB forms the encoded message EM. Note that the EM includes

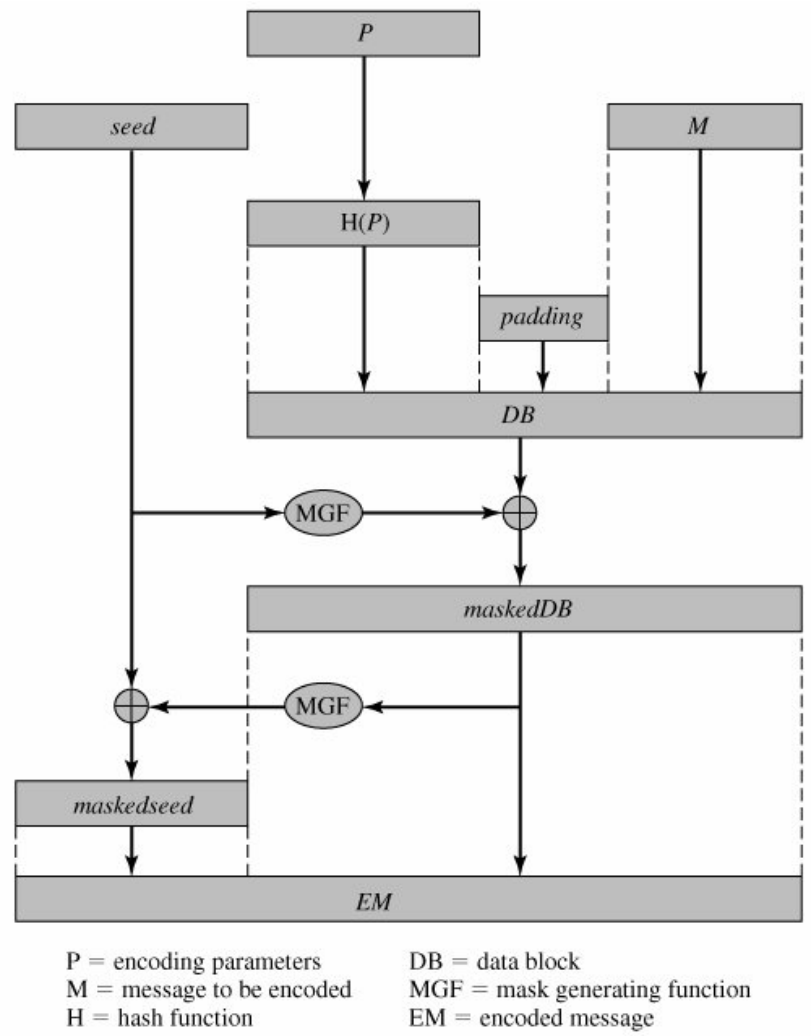


Fig 2.4.5 Encryption Using Optimal Assymetric Encryption Padding (OAEP)

the padded message, masked by the seed, and the seed, masked by the maskedDB. The EM is then encrypted using RSA.

DIFFIE-HELLMAN KEY EXCHANGE

Diffie-Hellman key exchange (**D-H**) is a cryptographic protocol that allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher.

Synonyms of Diffie-Hellman key exchange include:

- Diffie-Hellman key agreement
- Diffie-Hellman key establishment
- Diffie-Hellman key negotiation

- Exponential key exchange

The scheme was first published publicly by Whitfield Diffie and Martin Hellman in 1976.

The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent encryption of messages. The algorithm itself is limited to the exchange of secret values.

The Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms. Briefly, we can define the discrete logarithm in the following way. First, we define a primitive root of a prime number p as one whose powers modulo p generate all the integers from 1 to p 1. That is, if a is a primitive root of the prime number p, then the numbers

$$\mathbf{a \bmod p,\ a^2 \bmod p,\ldots,\ a^{p1} \bmod p}$$

are distinct and consist of the integers from 1 through p 1 in some permutation.

For any integer b and a primitive root a of prime number p, we can find a unique exponent i such that

$$\mathbf{b \equiv a^i \pmod p \text{ where } 0 \le i \le (p\ 1)}$$

The exponent i is referred to as the discrete logarithm of b for the base a, mod p. We express this value as $dlog_{a,p}$ (b).

The Algorithm

Fig 2.4.6 summarizes the Diffie-Hellman key exchange algorithm. For this scheme, there are two publicly known numbers: a prime number q and an integer that is a primitive root of q. Suppose the users A and B wish to exchange a key. User A selects a random integer $X_A < q$ and computes $Y_A = a^{XA} \bmod q$. Similarly, user B independently selects a random integer $X_A < q$ and computes $Y_B = a^{XB} \bmod q$. Each side keeps the X value private and makes the Y value available publicly to the other side. User A computes the key as

$$\mathbf{K = (Y_B)^{X}{}_{A} \bmod q}$$

and user B computes the key as

$$\mathbf{K = (Y_A)^{XB} \bmod q.}$$

These two calculations produce identical results:

$K = (Y_B)^{XA} \bmod q$

$= (a^{XB} \bmod q)^{XA} \bmod q$

$= (a^{XB})^{XA} \bmod q$ by the rules of modular arithmetic

$= (a^{XB\ XA} \bmod q$

$= (a^{XA})^{XB} \bmod q$

$= (a^{XA} \bmod q)$

$= (a^{XA} \bmod q)^{XB} \bmod q$

$= (Y_A)^{XB} \bmod q$

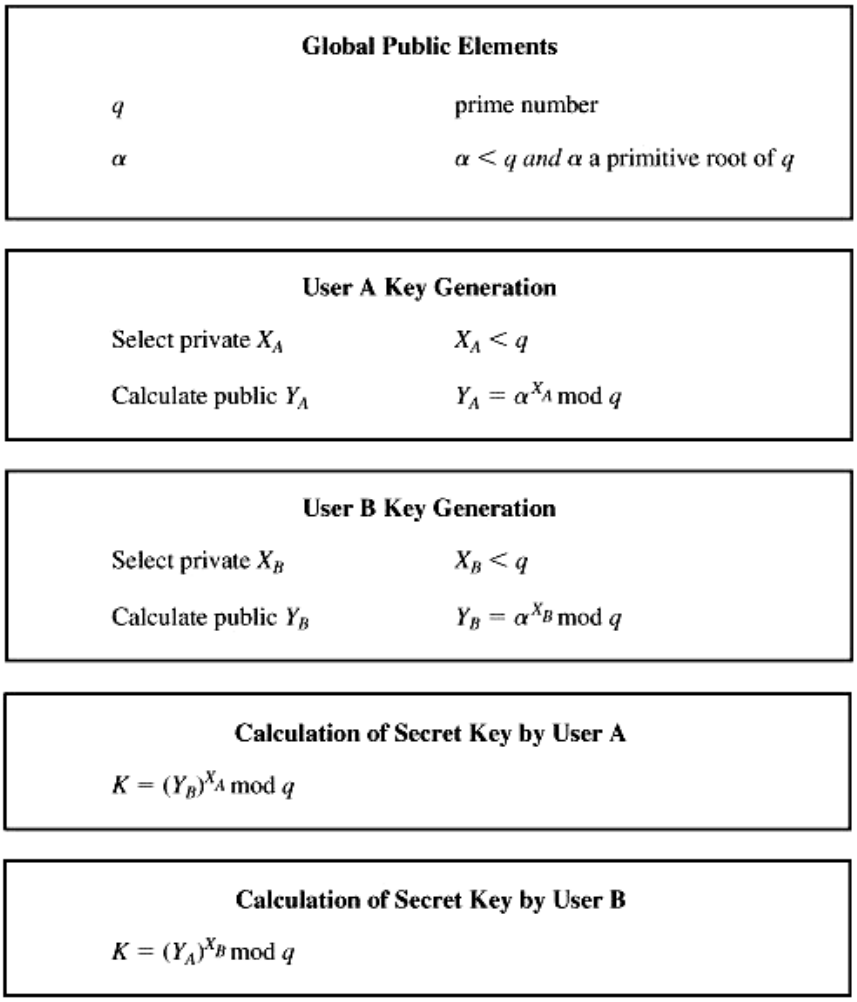


Fig 2.4.6 The Diffie-Hellman Key Exchange Algorithm

The result is that the two sides have exchanged a secret value. Furthermore, because X_A and X_B are private, an adversary only has the following ingredients to work with: q, a, Y_A, and Y_B. Thus, the adversary is forced to take a discrete logarithm to determine the key. For example, to determine the private key of user B, an adversary must compute

$$X_B = \text{dlog}_{a,q}(Y_B)$$

The adversary can then calculate the key K in the same manner as user B calculates it.

The security of the Diffie-Hellman key exchange lies in the fact that, while it is relatively easy to calculate exponentials modulo a prime, it is very difficult to calculate discrete logarithms. For large primes, the latter task is considered infeasible.

Here is an example. Key exchange is based on the use of the prime number q = 353 and a primitive root of 353, in this case a = 3. A and B select secret keys X_A = 97 and X_B = 233, respectively. Each computes its public key:

$$\text{A computes } Y_A = 3^{97} \bmod 353 = 40.$$

$$\textbf{A computes } Y_A = 3^{97} \bmod 353 = 40.$$

$$\textbf{B computes } Y_B = 3^{233} \bmod 353 = 248.$$

After they exchange public keys, each can compute the common secret key:

$$\textbf{A computes } K = (Y_B)^{X_A} \bmod 353 = 248^{97} \bmod 353 = 160.$$

$$\textbf{B computes } K = (Y_A)^{X_E} \bmod 353 = 40^{233} \bmod 353 = 160.$$

We assume an attacker would have available the following information:
$q = 353$; $a = 3$; $Y_A = 40$; $Y_B = 248$

In this simple example, it would be possible by brute force to determine the secret key 160. In particular, an attacker E can determine the common key by discovering a solution to the equation $3^a \bmod 353 = 40$ or the equation $3^b \bmod 353 = 248$. The brute-force approach is to calculate powers of 3 modulo 353, stopping when the result equals either 40 or 248. The desired answer is reached with the exponent value of 97, which provides $3^{97} \bmod 353 = 40$.

With larger numbers, the problem becomes impractical.

Key Exchange Protocols

Figure 2.4.7 shows a simple protocol that makes use of the Diffie-Hellman calculation. Suppose that user A wishes to set up a connection with user B and use a secret key to encrypt messages on that connection. User A can generate a one-time private key X_A, calculate Y_A, and send that to user B. User B responds by generating a private value X_B calculating Y_B, and sending Y_B to user A. Both users can now calculate the key. The necessary public values q and a would need to be known ahead of time. Alternatively, user A could pick values for q and a and include those in the first message.

As an example of another use of the Diffie-Hellman algorithm, suppose that a group of users (e.g., all users on a LAN) each generate a long-lasting private value X_i (for user i) and calculate a public value Y_i. These public values, together with global public values for q and a, are stored in some central directory. At any time, user j can access user i's public value, calculate a secret key, and use that to send an encrypted message to user A. If the central directory is trusted, then this form of communication provides both confidentiality and a degree of authentication. Because only i and j can determine the key, no other user can read the message (confidentiality). Recipient i knows that only user j could have created a message using this key (authentication). However, the technique does not protect against replay attacks.

Man-in-the-Middle Attack

The protocol depicted in Figure 2.4.7 is insecure against a man-in-the-middle attack. Suppose Alice and Bob wish to exchange keys, and Darth is the adversary. The attack proceeds as follows:

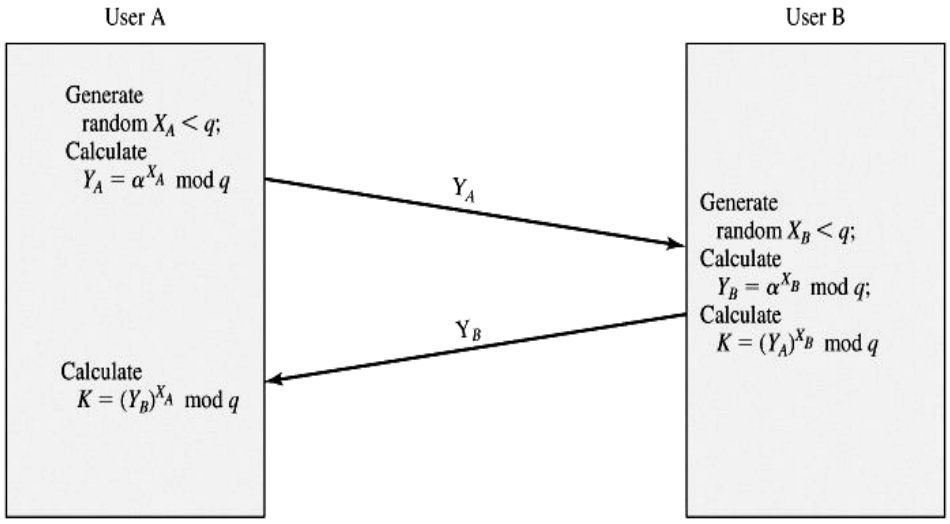


Figure 2.4.7 Diffie-Hellman Key Exchange

1. Darth prepares for the attack by generating two random private keys X_{D1} and X_{D2} and then computing the corresponding public keys Y_{D1} and Y_{D2}.
2. Alice transmits Y_A to Bob.
3. Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates K2 = $(Y_A)^{X}{}_{D2}$ mod q.
4. Bob receives Y_{D1} and calculates K1 = $(Y_{D1})^{X}{}_{E}$ mod q.
5. Bob transmits X_A to Alice.
6. Darth intercepts X_A and transmits Y_{D2} to Alice. Darth calculates K1 = $(Y_B)^{X}{}_{D1}$ mod q.
7. Alice receives Y_{D2} and calculates K2 = $(Y_{D2})^{X}{}_{A}$ mod q.

At this point, Bob and Alice think that they share a secret key, but instead Bob and Darth share secret key K1 and Alice and Darth share secret key K2. All future communication between Bob and Alice is compromised in the following way:

1. Alice sends an encrypted message M: E(K2, M).
2. Darth intercepts the encrypted message and decrypts it, to recover M.
3. Darth sends Bob E(K1, M) or E(K1, M'), where M' is any message. In the first case, Darth simply wants to eavesdrop on the communication without altering it. In the second case, Darth wants to modify the message going to Bob.

The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants. This vulnerability can be overcome with the use of digital signatures and public-key certificates.

REFERENCES

1. William Stallings, Cryptography and Network Security, PHI Publishers
2. www.wikipedia.org

Review Questions:

1. Write short notes on Diffie-Hellman Key Exchange algorithm
2. Write a note on computational aspects with security of RSA?
3. Explain the model for Network security?
4. What are the different types of security services?
5. Explain the Diffie-Hellman Key exchange algorithm in detail.
6. Explain RSA public key Encryption algorithm in detail.

UNIT III

1. **ELLIPTICAL CURVE CRYPTOSYSTEMS**
2. **MESSAGE AUTHENTICATION AND HASH FUNCTIONS**
3. **DIGITAL SIGNATURES**
4. **KEY MANAGEMENT SCHEMES**

1. ELLIPTICAL CURVE CRYPTOGRAPHY

OBJECTIVE

The objective of this lesson is to introduce the reads yet another variety of public-key cryptosystem namely the elliptical curve encryption. The elliptical curve encryption is a more recent public-key approach based on elliptic curves.

ELLIPTICAL CURVES

An elliptic curve is given by a cubic with a point. The "standard elliptic curve" has the form

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

for some fixed a and b. This is called a Weierstrass equation.

Why elliptic curves?

Elliptic curves are the simplest possible curves after lines and conics. Lines and conics (which are usually given by linear and quadratic equations respectively) are fairly easy to understand. Curves more complicated than elliptic curves (e.g., defined by polynomials of higher degree than 3) are generally difficult to understand. Elliptic curves are "just right". The fact that they can be dealt with is due to their very rich structure; in particular, they can be viewed as a group. Most of this course is about investigating this group structure.

In mathematics, an **elliptic curve** is a smooth, projective algebraic curve of genus one, on which there is a specified point *O*. An elliptic curve is in fact an abelian variety — that is, it has a multiplication defined algebraically with respect to which it is an abelian group — and *O* serves as the identity element. Often the curve itself, without *O* specified, is called an elliptic curve.

If $y^2 = P(x)$, where *P* is any polynomial of degree three in *x* with no repeated roots, then we obtain a nonsingular plane curve of genus one, which is thus also an elliptic curve. If *P* has degree four and is squarefree this equation again describes a plane curve of genus one; however, it has no natural choice of identity element. More generally, any algebraic curve of genus one, for example from the intersection of two three-dimensional quadric surfaces, is called an elliptic curve, provided that it has at least one rational point.

Using the theory of elliptic functions, it can be shown that elliptic curves defined over the complex numbers correspond to embeddings of the torus into the complex projective plane. The torus is also an abelian group, and in fact this correspondence is also a group isomorphism.

Elliptic curves are especially important in number theory, and constitute a major area of current research; for example, they were used in the proof, by Andrew Wiles (assisted by Richard Taylor), of Fermat's Last Theorem.

They also find applications in cryptography (see the article elliptic curve cryptography) and integer factorization.

Abelian Groups

An abelian group G, sometimes denoted by {G, •}, is a set of elements with a binary operation, denoted by •, that associates to each ordered pair (a, b) of elements in G an element (a • b) in G, such that the following axioms are obeyed:

(The operator • is generic and can refer to addition, multiplication, or some other mathematical operation).

(A1) Closure:	If a and b belong to G, then a • b is also in G.
(A2) Associative:	a • (b • c) = (a • b) • c for all a, b, c in G.
(A3) Identity element:	There is an element e in G such that a • e = e • a = a for all a in G.
(A4) Inverse element:	For each a in G there is an element a' in G such that a • a' = a' • a = e.
(A5) Commutative:	a • b = b • a for all a, b in G.

A number of public-key ciphers are based on the use of an abelian group. For example, Diffie-Hellman key exchange involves multiplying pairs of nonzero integers modulo a prime number q. Keys are generated by exponentiation over the group, with exponentiation defined as repeated multiplication. For example, a^k mod q = $\underbrace{a X a X \ldots . X a}_{K times}$ mod q. To attack Diffie-Hellman, the attacker must determine k given a and a^k; this is the discrete log problem.

For elliptic curve cryptography, an operation over elliptic curves, called addition, is used. Multiplication is defined by repeated addition. For example, $a X k = \underbrace{a X a X \ldots . X a}_{K times}$ where the addition is performed over an elliptic curve. Cryptanalysis involves determining k given a and (a x k).

An elliptic curve is defined by an equation in two variables, with coefficients. For cryptography, the variables and coefficients are restricted to elements in a finite field, which results in the definition of a finite abelian group. Before looking at this, we first look at elliptic curves in which the variables and coefficients are real numbers. This case is perhaps easier to visualize.

Elliptic Curves over Real Numbers

Elliptic curves are not ellipses. They are so named because they are described by cubic equations, similar to those used for calculating the circumference of an ellipse. In general, cubic equations for elliptic curves take the form

$$y^2 + axy + by = x^3 + cx^2 + dx + e$$

where a, b, c, d, and e are real numbers and x and y take on values in the real numbers.For our purpose, it is sufficient to limit ourselves to equations of the form

$$y^2 = x^3 + ax + b \qquad \text{- (Eqn 3.1.1)}$$

For given values of a and b, the plot consists of positive and negative values of y for each value of x. Thus each curve is symmetric about y = 0. Fig 3.1.1 shows two examples of elliptic curves. As you can see, the formula sometimes produces weird-looking curves.

Elliptic Curves over Z_p

Elliptic curve cryptography makes use of elliptic curves in which the variables and coefficients are all restricted to elements of a finite field. Two families of elliptic curves are used in cryptographic applications: prime curves over Z_p and binary curves over GF(2^m). For a prime curve over Z_p, we use a cubic equation in which the variables and coefficients all take on values in the set of integers from 0 through p 1 and in which calculations are performed modulo p. For a binary curve defined over GF(2^m), the variables and coefficients all take on values in GF(2^n) and in calculations are performed over GF(2^n).

There is no obvious geometric interpretation of elliptic curve arithmetic over finite fields. The algebraic interpretation used for elliptic curve arithmetic over real numbers does readily carry over, and this is the approach we take.

For elliptic curves over Z_p, as with real numbers, we limit ourselves to equations of the form of Equation (3.1.1), but in this case with coefficients and variables limited to Z_p:

$$y^2 \bmod p = (x^3 + ax + b) \bmod p \qquad \textbf{- (Eqn 3.1.2)}$$

For example, Equation (3.1.2) is satisfied for a = 1, b = 1, x = 9, y = 9, y = 7, p = 23:

$$7^2 \bmod 23 = (9^3 + 9 + 1) \bmod 23$$

$$49 \bmod 23 = 739 \bmod 23$$

$$3 = 3$$

Now consider the set E_p (a, b) consisting of all pairs of integers (x, y) that satisfy Equation (3.1.2), together with a point at infinity O. The coefficients a and b and the variables x and y are all elements of Z_p.

For example, let p = 23 and consider the elliptic curve $y^2 = x^3 + x + 1$. In this case, a = b = 1. Note that this equation is the same as that of Figure 3.1.1b. The figure shows a continuous curve with all of the real points that satisfy the equation. For the set $E_{23}(1, 1)$, we are only interested in the nonnegative integers in the quadrant from (0, 0) through (p 1, p 1) that satisfy the equation mod p. Table 3.1.1 lists the points (other than O) that are part of $E_{23}(1,1)$. Fig 3.1.2 plots the points of $E_{23}(1,1)$; note that the points, with one exception, are symmetric about y = 11.5.

It can be shown that a finite abelian group can be defined based on the set $E_p(a, b)$ provided that $(x^3 + ax + b) \bmod p$ has no repeated factors. This is equivalent to the condition

$$(4a^3 + 27b^2) \bmod p \neq 0 \bmod p \qquad \textbf{- (Eqn 3.1.3)}$$

The rules for addition over $E_p(a, b)$ correspond to the algebraic technique described for elliptic curves defined over real number. For all points P, Q $\in$ $E_p(a, b)$;

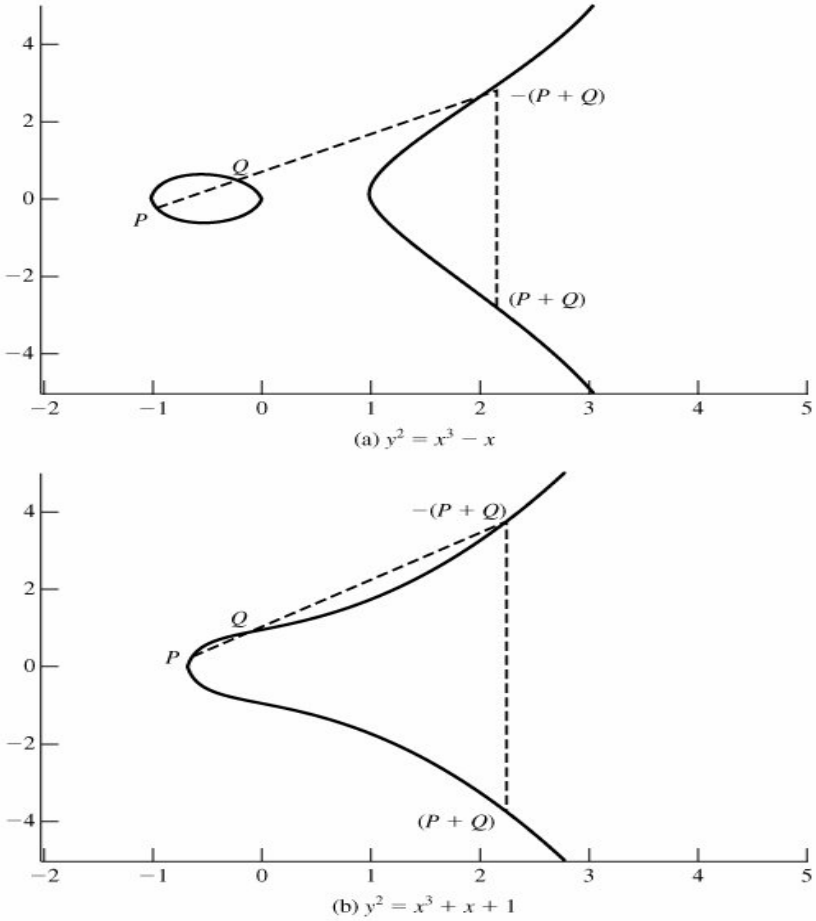


Fig 3.1.1 Examples of Elliptical Curves

(0, 1)	(6, 4)	(12, 19)
(0, 22)	(6, 19)	(13, 7)
(1, 7)	(7, 11)	(13, 16)
(1, 16)	(7, 12)	(17, 3)
(3, 10)	(9, 7)	(17, 20)
(3, 13)	(9, 16)	(18, 3)
(4, 0)	(11, 3)	(18, 20)
(5, 4)	(11, 20)	(19, 5)
(5, 19)	(12, 4)	(19, 18)

Table 3.1.1 Points on the Elliptic Curve $E_{23}(1,1)$

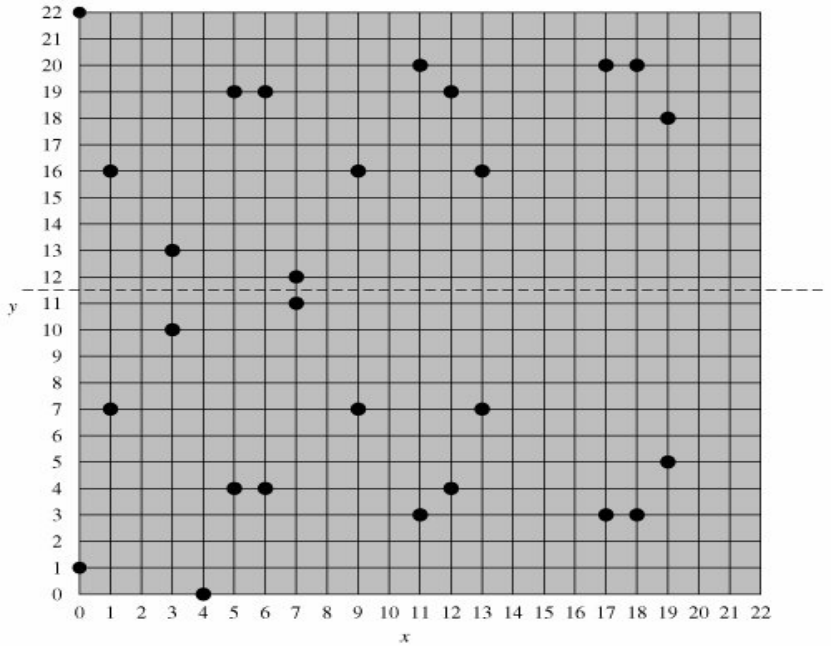


Fig 3.1.2 The Elliptic Curve $E_{23}(1,1)$

1. P + O = P.
2. If P = (x_P, y_P) then P + (x_P, y_P) = O. The point (x_P, y_P) is the negative of P, denoted as P. For example, in $E_{23}(1,1)$, for P = (13,7), we have P = (13, 7). But 7 mod 23 = 16. Therefore, P = (13, 16), which is also in $E_{23}(1,1)$.

3. If P = (x_P, y_Q) and Q = (x_Q, y_Q) with P ≠ Q, then R = P + Q = (x_R, y_R) is determined by the following rules:

$$x_R = (\lambda^2 x_P x_Q) \bmod p$$

$$y_R = (\lambda (x_P x_R) y_P) \bmod p$$

where

$$\lambda = \begin{cases} \left(\frac{y_Q - y_P}{x_Q - x_P}\right) mod\ p\ if\ P \neq Q \\ \left(\frac{3x^2P + a}{2y_P}\right) mod\ p\ if\ P = Q \end{cases}$$

4. Multiplication is defined as repeated addition; for example,

4P = P + P + P + P.

For example, let P = (3,10) and Q = (9,7) in $E_{23}(1,1)$. Then

$$\lambda = \left(\frac{7-10}{9-3}\right) mod\ 23 = \left(\frac{-3}{6}\right) mod\ 23 = \left(\frac{-1}{2}\right) mod\ 23 = 11$$

$x_R = (11^2\ 3\ 9) \bmod 23 = 17$

$y_R = (11(3\ 17)\ 10) \bmod 23 = 164 \bmod 23 = 20$

So P + Q = (17, 20). To find 2P,

$$\lambda = \left(\frac{3(3^2)+1}{2\ X\ 10}\right)\ mod\ 23 = \left(\frac{5}{20}\right) mod\ 23 = \left(\frac{1}{4}\right) mod\ 23 = 6$$

The last step in the preceding equation involves taking the multiplicative inverse of 4 in Z_{23}. This can be done using the extended Euclidean algorithm. To confirm, note that (6 x 4) mod 23 = 24 mod 23 = 1.

x_R = (6^2 3 3) mod 23 = 30 mod 23 = 7

y_R = (6(3 7) 10) mod 23 = (34) mod 23 = 12

and 2P = (7, 12).

For determining the security of various elliptic curve ciphers, it is of some interest to know the number the number of points in a finite abelian group defined over an elliptic curve. In the case of the finite group $E_p(a,b)$, the number of points N is bounded by

$$p + 1 - 2\sqrt{p} \leq N \leq p + 1 + 2\sqrt{p}$$

The number of points in $E_p(a, b)$ is approximately equal to the number of elements in Zp, namely p elements.

Elliptic Curves over GF(2^m)

A finite field GF(2^m) consists of 2^m elements, together with addition and multiplication operations that can be defined over polynomials. For elliptic curves over GF(2^m), we use a cubic equation in which the variables and coefficients all take on values in GF(2^m), for some number m, and in which calculations are performed using the rules of arithmetic in GF(2^m).

It turns out that the form of cubic equation appropriate for cryptographic applications for elliptic curves is somewhat different for GF(2^m) than for Z_p. The form is

$$y^2 + xy = x^3 + ax^2 + b \qquad \textbf{- (Eqn 3.1.4)}$$

where it is understood that the variables x and y and the coefficients a and b are elements of GF(2^m) of and that calculations are performed in GF(2^m).

Now consider the set $E_{2^m}(a, b)$ consisting of all pairs of integers (x, y) that satisfy Equation (3.1.4), together with a point at infinity O.

For example, let us use the finite field GF(2^4) with the irreducible polynomial f(x) = x^4 + x + 1. This yields a generator that satisfies f(g) = 0, with a value of g^4 = g + 1, or in binary 0010. We can develop the powers of g as follows:

g^0 = 0001	g^4 = 0011	g^8 = 0101	g^{12} = 1111
g^1 = 0010	g^5 = 0110	g^9 = 1010	g^{13} = 1101
g^2 = 0100	g^6 = 1100	g^{10} = 0111	g^{14} = 1001
g^3 = 1000	g^7 = 1011	g^{11} = 1110	g^{15} = 0001

For example, g^5 = (g^4)(g) = g^2 + g = 0110.

Now consider the elliptic curve y^2 + xy = x^3 + g^4x^2 + 1. In this case a = g^4 and b = g^0 = 1. One point that satisfies this equation is (g^5, g^3):

(g^3)^2 + (g^5)(g^3) = (g^5)^3 + (g^4)(g^5)^2 + 1

g^6 + g^8 = g^{15} + g^{14} + 1

1100 + 0101 = 0001 + 1001 + 0001

1001 = 1001

Table 3.1.2 lists the points (other than O) that are part of E_2^4 (g^4, 1). Figure 3.1.3 plots the points of E_2^4 (g^4, 1).

(0, 1)	(g^5, g^3)	(g^9, g^{13})
(1, g^6)	(g^5, g^{11})	(g^{10}, g)
(1, g^{13})	g^6, g^8)	(g^{10}, g^8)
(g^3, g^8)	(g^6, g^{14})	(g^{12},0)
(g^3, g^{13})	(g^9, g^{10})	(g^{12}, g^{12})

Table 3.1.2 Points on the Elliptical curve E_{24} (g^4, 1)

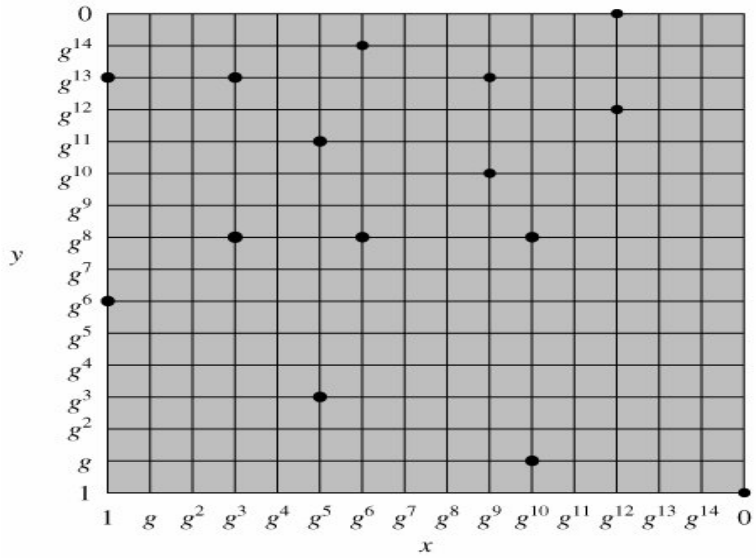


Fig 3.1.3 The Elliptic Curve E_2^4(g^4, 1)

It can be shown that a finite abelian group can be defined based on the set E_{2m} (a, b), provided that b ≠0. The rules for addition can be stated as follows. For all points P, Q Є E_2^m (a, b):

1. P + O = P.
2. If P = (x_P, y_P), then P + (x_P, x_p + y_P) = O. The point (x_P, x_P + y_P) is the negative of P, denoted as P.
3. If P = (x_P, y_P) and Q = (x_Q, y_Q) with P ≠Q and P ≠Q, then R = P + Q = (x_R, y_R) is determined by the following rules:

   $x_R = \lambda^2 + \lambda + x_P + x_Q + a$

   $y_R = \lambda(x_P + x_R) + x_R + y_P$

   where

   $$\lambda = \left(\frac{y_Q - y_P}{x_Q - x_P}\right)$$

4. If = (x_P, y_P) then R = 2P = (x_R, y_R) is determined by the following rules:

   $$x_R = \lambda^2 + \lambda + a$$

$$y_R = x_P^2 + (\lambda + 1)x_R$$

Where

$$\lambda = x_P + \frac{y_P}{x_P}$$

Elliptic Curve Cryptography

The addition operation in ECC is the counterpart of modular multiplication in RSA, and multiple addition is the counterpart of modular exponentiation. To form a cryptographic system using elliptic curves, we need to find a "hard problem" corresponding to factoring the product of two primes or taking the discrete logarithm.

Consider the equation Q = kP where Q, P Є $E_p(a, b)$ and k < p. It is relatively easy to calculate Q given k and P, but it is relatively hard to determine k given Q and P. This is called the discrete logarithm problem for elliptic curves.

We give an example taken from the Certicom Web site (www.certicom.com). Consider the group E_{23} (9, 17). This is the group defined by the equation y^2 mod 23 = (x^3 + 9x + 17) mod 23. What is the discrete logarithm k of Q = (4, 5) to the base P = (16.5)? The brute-force method is to compute multiples of P until Q is found.

Thus

P = (16, 5); 2P = (20, 20); 3P = (14, 14); 4P = (19, 20); 5P = (13, 10); 6P = (7, 3); 7P = (8, 7); 8P (12, 17); 9P = (4, 5).

Because 9P = (4, 5) = Q, the discrete logarithm Q = (4, 5) to the base P = (16, 5) is k = 9. In a real application, k would be so large as to make the brute-force approach infeasible.

In the remainder of this section, we show two approaches to ECC that give the flavor of this technique.

Analog of Diffie-Hellman Key Exchange

Key exchange using elliptic curves can be done in the following manner. First pick a large integer q, which is either a prime number p or an integer of the form 2^m and elliptic curve parameters a and b for Equation (3.1.2) or Equation (3.1.3). This defines the elliptic group of points $E_q(a, b)$. Next, pick a base point G = (x_1, y_1) in E_p (a, b) whose order is a very large value n. The order n of a point G on an elliptic curve is the smallest positive integer n such that nG = O. $E_q(a, b)$ and G are parameters of the cryptosystem known to all participants.

A key exchange between users A and B can be accomplished as follows (Figure 3.1.4):

1. A selects an integer n_A less than n. This is A's private key. A then generates a public key P_A = n_A x G; the public key is a point in Eq(a, b).
2. B similarly selects a private key n_B and computes a public key P_B.
3. A generates the secret key K = n_A x P_B. B generates the secret key K = n_B x P_A.

User A Key Generation	
Select private n_A	$n_A < n$
Calculate Public P_A	$P_A = n_A \times G$
User B Key Generation	
Select private n_B	$n_B < n$
Calculate Public P_B	$P_B = n_B \times G$
Calculation of Secret Key by User A	
$K = n_A \times P_B$	
Calculation of Secret Key by User B	
$K = n_B \times P_A$	

Fig 3.1.4 ECC Diffie-Hellman Key Exchange

The two calculations in step 3 produce the same result because

$$n_A \times P_B = n_A \times (n_B \times G) = n_B \times (n_A \times G) = n_B \times P_A$$

To break this scheme, an attacker would need to be able to compute k given G and kG, which is assumed hard.

As an example, take p = 211; Ep(0, 4), which is equivalent to the curve $y^2 = x^3$ 4; and G = (2, 2). One can calculate that 240G = O. A's private key is n_A = 121, so A's public key is P_A = 121(2, 2) = (115, 48). B's private key is n_B = 203, so B's public key is 203(2, 2) = (130, 203). The shared secret key is 121(130, 203) = 203(115, 48) = (161, 69).

Note that the secret key is a pair of numbers. If this key is to be used as a session key for conventional encryption, then a single number must be generated. We could simply use the x coordinates or some simple function of the x coordinate.

Elliptic Curve Encryption/Decryption

Several approaches to encryption/decryption using elliptic curves have been analyzed in the literature. In this subsection we look at perhaps the simplest. The first task in this system is to encode the plaintext message m to be sent as an x-y point P_m. It is the point P_m that will be encrypted as a ciphertext and subsequently decrypted. Note that we cannot simply encode the message as the x or y coordinate of a point, because not all such coordinates are in $E_q(a, b)$; for example, see Table 3.1.1. Again, there are several approaches to this encoding, which we will not address here, but suffice it to say that there are relatively straightforward techniques that can be used.

As with the key exchange system, an encryption/decryption system requires a point G and an elliptic group Eq(a, b) as parameters. Each user A selects a private key n_A and generates a public key $P_A = n_A \times G$.

To encrypt and send a message P_m to B, A chooses a random positive integer k and produces the ciphertext C_m consisting of the pair of points:

$$C_m = \{kG, P_m + kP_B\}$$

Note that A has used B's public key P_B. To decrypt the ciphertext, B multiplies the first point in the pair by B's secret key and subtracts the result from the second point:

$$P_m + kP_B \; n_B(kG) = P_m + k(n_B G) \; n_B(kG) = P_m$$

A has masked the message P_m by adding kP_B to it. Nobody but A knows the value of k, so even though P_B is a public key, nobody can remove the

mask kP_B. However, A also includes a "clue," which is enough to remove the mask if one knows the private key n_B. For an attacker to recover the message, the attacker would have to compute k given G and kG, which is assumed hard.

As an example of the encryption process, take p = 751; $E_p(1, 188)$, which is equivalent to the curve $y^2 = x^3$ x + 188; and G = (0, 376). Suppose that A wishes to send a message to B that is encoded in the elliptic point P_m = (562, 201) and that A selects the random number k = 386. B's public key is P_B = (201, 5). We have 386(0, 376) = (676, 558), and (562, 201) + 386(201, 5) = (385, 328). Thus A sends the cipher text {(676, 558), (385, 328)}.

Security of Elliptic Curve Cryptography

The security of ECC depends on how difficult it is to determine k given kP and P. This is referred to as the elliptic curve logarithm problem. The fastest known technique for taking the elliptic curve logarithm is known as the Pollard rho method. Table 3.1.3 compares various algorithms by showing comparable key sizes in terms of computational effort for cryptanalysis. As can be seen, a considerably smaller key size can be used for ECC compared to RSA. Furthermore, for equal key lengths, the computational effort required for ECC and RSA is comparable. Thus, there is a computational advantage to using ECC with a shorter key length than a comparably secure RSA.

Symmetric Scheme (key size in bits)	ECC-Based Scheme (size of n in bits)	RSA/DSA (modulus size in bits)
56	112	512
80	160	1024
112	224	2048
128	256	3072
92	384	7680
256	512	15360
Source: Certicom		

Table 3.1.3 Comparable Key Sizes in Terms of Computational Effort for Cryptanalysis

REFERENCES

1. William Stallings, Cryptography and Network Security, PHI Publishers
2. www.wikipedia.org

Review Questions:

1. Explain the application of Elliptical curves in public key cryptosystems.
2. Explain Elliptical curve encryption algorithm

2. MESSAGE AUTHENTICATION AND HASH FUNCTIONS

OBJECTIVES

The objective of this lesson is to introduce the requirements for authentication and digital signature, types of attacks to be countered. Survey of basic approaches, including the increasingly important area of secure hash functions. Some specific hash functions are examined.

MESSAGE AUTHENTICATION

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

MAC = C (K, M)

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.

A MAC algorithm accepts as input a secret key and an arbitrary-length message to be authenticated, and outputs a MAC (sometimes known as a *tag*). The MAC value protects both a message's data integrity as well as its authenticity, by allowing verifiers (who also possess the secret key) to detect any changes to the message content. Fig 3.2.1 gives the overview of MAC function.

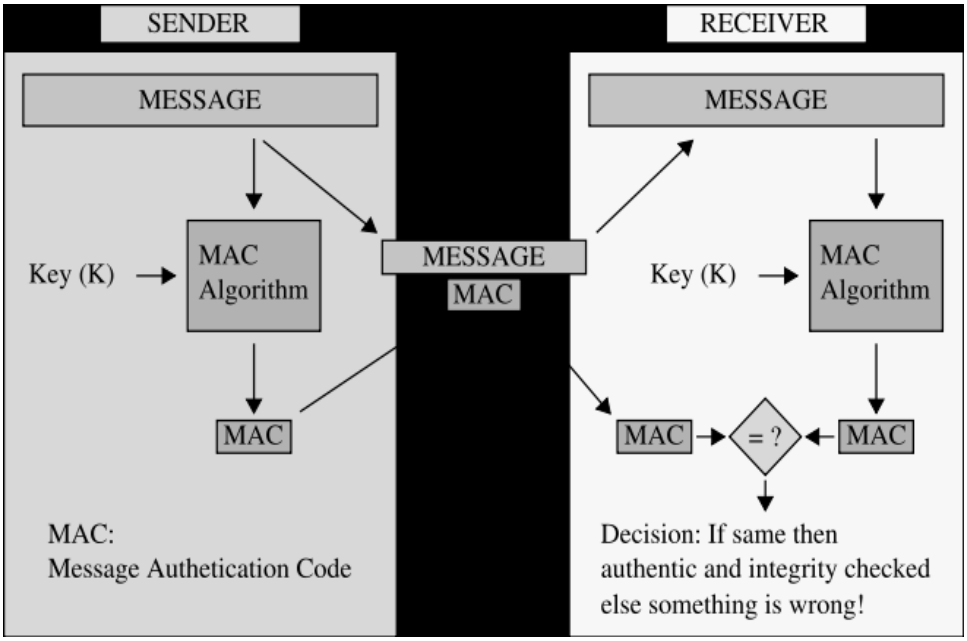


Fig 3.2.1 Overview of MAC functions.

Message authentication is a procedure to verify that received messages come from the alleged source and have not been altered. Message

authentication may also verify sequencing and timeliness. A digital signature is an authentication technique that also includes measures to counter repudiation by the source.

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 $2^{(k-1)}$ attempts for a k-bit key. In particular, for a ciphertext-only attack, the opponent, given ciphertext C, would perform $P_i = D(K_i, C)$ 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, due to the many-to-one nature of the function. 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 $MAC_1 = C(K, M_1)$, 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 is 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 $MAC_i = C(K_i, M_1)$ for all 2^k keys
  Number of matches $\approx 2^{(k-n)}$
- Round 2
  Given: M_2, $MAC_2 = C(K, M_2)$
  Compute $MAC_i = C(K_i, M_2)$ for the $2^{(k-n)}$ keys resulting from Round 1
  Number of matches $\approx 2^{(k-2xn)}$

and so on. On average, α rounds will be needed if k = α x 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.

Consider the following MAC algorithm. Let M = ($X_1||X_2||...||X_m$) be a message that is treated as a concatenation of 64-bit blocks X_i. Then define

$$\Delta(M) = X_1 \oplus X_2 \oplus ... \oplus X_m$$

$$C(K, M) = E(K, \Delta(M))$$

where ⊕is the exclusive-OR (XOR) operation and the encryption algorithm is DES in electronic codebook mode. Thus, the key length is 56 bits and the MAC length is 64 bits. If an opponent observes {M||C(K, M)}, a brute-force attempt to determine K will require at least 2^{56} encryptions. But the opponent can attack the system by replacing X_1 through X_{m-1} with any desired values Y_1 through Y_{m-1} and replacing X_m with Y_m where Y_m is calculated as follows:

$$Y_m = Y_1 \oplus Y_2 \oplus ... \oplus Y_{m1} \oplus \Delta(M)$$

The opponent can now concatenate the new message, which consists of Y_1 through Y_m, with the original MAC to form a message that will be accepted as authentic by the receiver. With this tactic, any message of length 64 x (m 1) bits can be fraudulently inserted.

Thus, in assessing the security of a MAC function, we need to consider the types of attacks that may be mounted against it. With that in mind, let us state the requirements for the function. Assume that an opponent knows the MAC function C but does not know K. Then the MAC function should satisfy the following requirements:

1. If an opponent observes M and C(K, M), it should be computationally infeasible for the opponent to construct a message M' such that C(K, M') = C(K, M).
2. C(K, M) should be uniformly distributed in the sense that for randomly chosen messages, M and M', the probability that C(K, M) = C(K, M') is 2^n, where n is the number of bits in the MAC.
3. Let M' be equal to some known transformation on M. That is, M' = f(M). For example, f may involve inverting one or more specific bits. In that case, Pr[C(K, M) = C(K, M')] = 2^n.

The first requirement speaks to the earlier example, in which an opponent is able to construct a new message to match a given MAC, even though the opponent does not know and does not learn the key. The second requirement deals with the need to thwart a brute-force attack based on chosen plaintext. That is, if we assume that the opponent does not know K but does have access to the MAC function and can present messages for MAC generation, then the opponent could try various messages until finding one that matches a given MAC. If the MAC function exhibits uniform distribution, then a brute-force method would require, on average, $2^{(n1)}$ attempts before finding a message that fits a given MAC.

The final requirement dictates that the authentication algorithm should not be weaker with respect to certain parts or bits of the message than others. If this were not the case, then an opponent who had M and C(K, M) could attempt variations on M at the known "weak spots" with a likelihood of early success at producing a new message that matched the old MAC.

Message Authentication Code Based on DES

The Data Authentication Algorithm, based on DES, has been one of the most widely used MACs for a number of years. The algorithm is both a FIPS publication (FIPS PUB 113) and an ANSI standard (X9.17). However, security weaknesses in this algorithm have been discovered and it is being replaced by newer and stronger algorithms.

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 are grouped into contiguous 64-bit blocks: D_1, D_2,..., 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 11.6):

$$O_1 = E(K, D_1)$$

$$O_2 = E(K, [D_2 \oplus O_1])$$

$$O_3 = (K, [D_3 \oplus O_2])$$

•

•

$$O_N = E(K, [D_N \oplus O_{N1}])$$

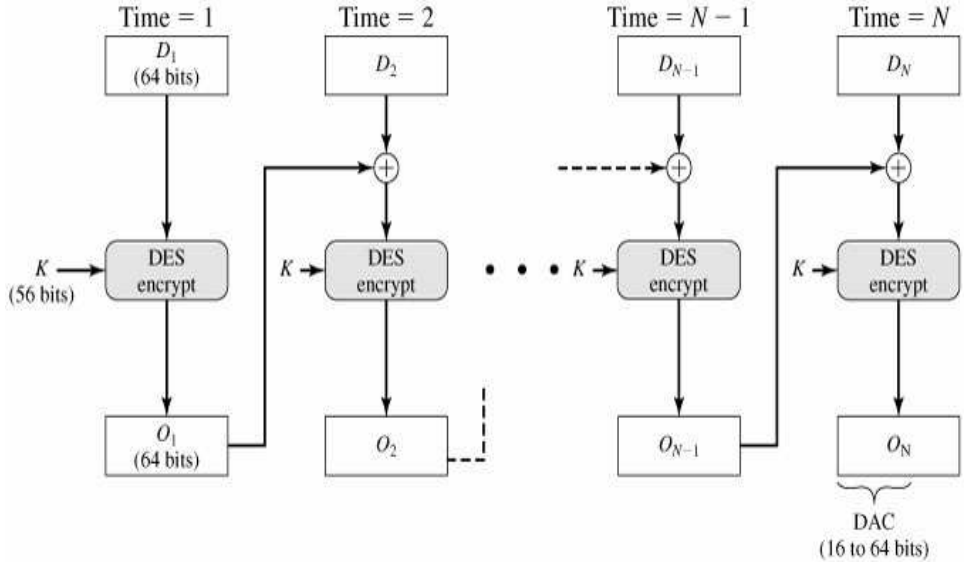


Fig 3.2.2 Data Authentication Algorithm (FIPS PUB 113)

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

HASH FUNCTIONS

A hash value h 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.

As hash functions are typically quite complex, it is useful to examine 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 value 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 ≠ x such that 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, the one-way property, states that 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 3.2.3e).

The secret value itself is not sent. But, 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 (Figures 3.2.3b and c). 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.

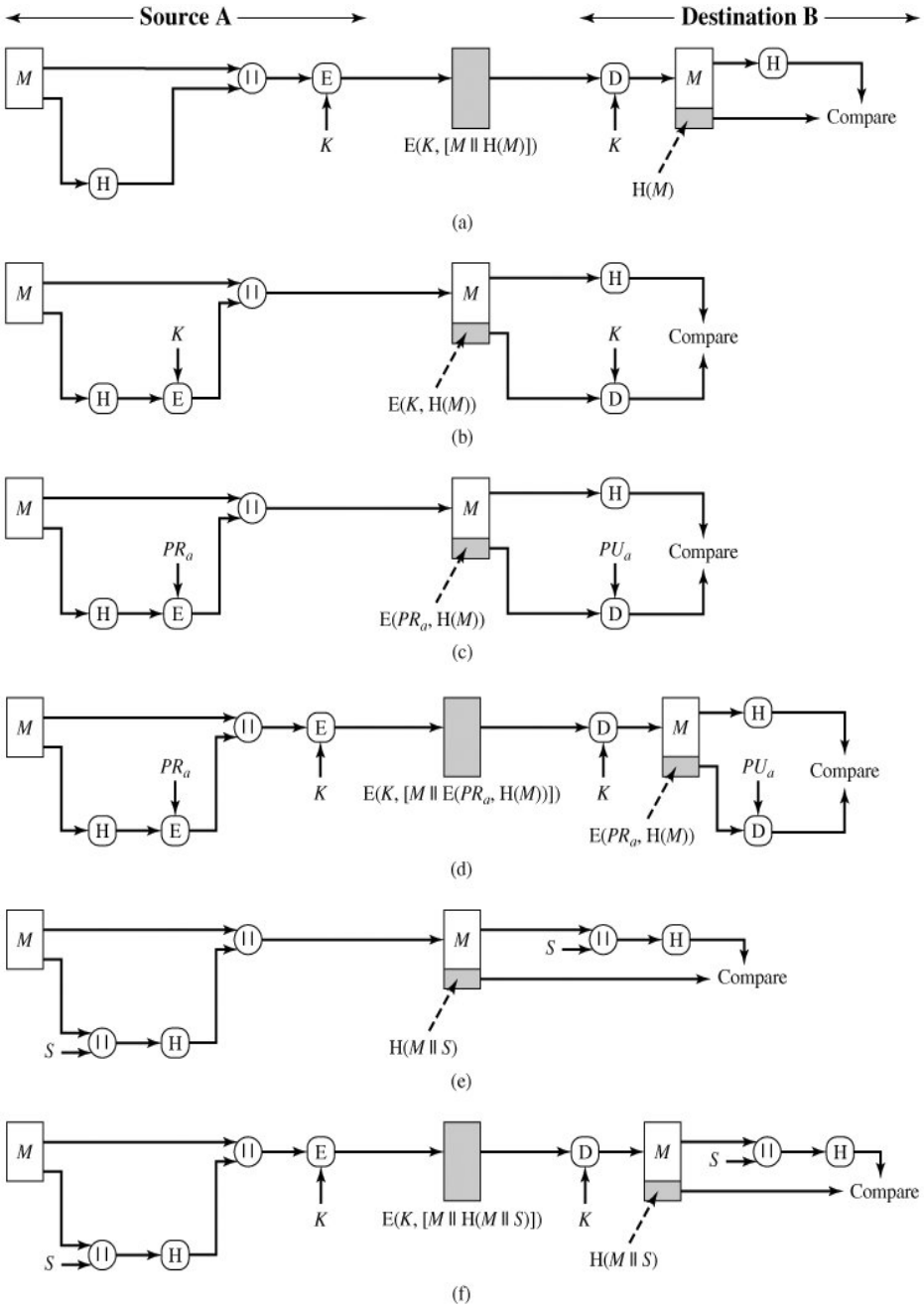


Fig 3.2.3 Basic Uses of Hash Function

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

Simple Hash Functions

All hash functions operate using the following general principles. The input (message, file, etc.) is viewed as a sequence of n-bit blocks. The input is processed one block at a time in an iterative fashion to produce an n-bit hash function.

One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of every block. This can be expressed as follows:

$C_i = b_{i1} \oplus b_{i1} \oplus ... \oplus b_{im}$

where

C_i = ith bit of the hash code, $1 \leq i \leq n$

m = number of n-bit blocks in the input

b_{ij} = ith bit in jth block

$\oplus$ = XOR operation

This operation produces a simple parity for each bit position and is known as a longitudinal redundancy check. It is reasonably effective for random data as a data integrity check. Each n-bit hash value is equally likely. Thus, the probability that a data error will result in an unchanged hash value is 2^n. With more predictably formatted data, the function is less effective. For example, in most normal text files, the high-order bit of each octet is always zero. So if a 128-bit hash value is used, instead of an effectiveness of 2^{128}, the hash function on this type of data has an effectiveness of 2^{112}.

A simple way to improve matters is to perform a one-bit circular shift, or rotation, on the hash value after each block is processed. The procedure can be summarized as follows:

1. Initially set the n-bit hash value to zero.
2. Process each successive n-bit block of data as follows:
    a. Rotate the current hash value to the left by one bit.
    b. XOR the block into the hash value.

This has the effect of "randomizing" the input more completely and overcoming any regularities that appear in the input. Fig 3.2.4 illustrates these two types of hash functions for 16-bit hash values.

Although the second procedure provides a good measure of data integrity, it is virtually useless for data security when an encrypted hash code is used with a plaintext message, as in Figures 3.2.3b and c. Given a message, it is an easy matter to produce a new message that yields that hash code: Simply prepare the desired alternate message and then append an n-bit block that forces the new message plus block to yield the desired hash code. Although a simple XOR or rotated XOR (RXOR) is insufficient if only the hash code is encrypted, you may still feel that such a simple function could be useful when the message as well as the hash code are encrypted (Figure 3.2.3a). But you must be careful. A technique originally proposed by the National Bureau of Standards used the simple XOR applied to 64-bit blocks of the message and then an encryption of the entire message that used the cipher block chaining (CBC) mode. We can define the scheme as follows:

Given a message consisting of a sequence of 64-bit blocks X_1, X_2,..., X_N, define the hash code C as the block-by-block XOR of all blocks and append the hash code as the final block:

$$\mathbf{C = X_{N+1} = X_1 \oplus X_2 \oplus ... \oplus X_N}$$

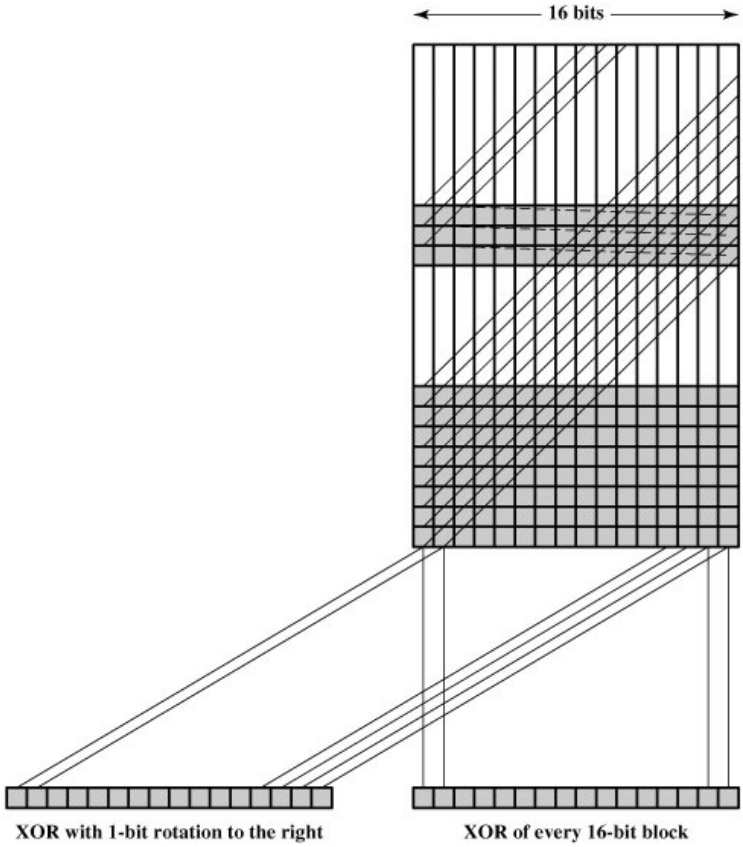


Figure 3.2.4 Two Simple Hash Functions

Next, encrypt the entire message plus hash code, using CBC mode to produce the encrypted message Y_1, Y_2,..., Y_{N+1}. points out several ways in which the ciphertext of this message can be manipulated in such a way that it is not detectable by the hash code. For example, by the definition of CBC, we have

$$X_1 = IV \oplus D(K, Y_1)$$

$$X_i = Y_{i1} \oplus D(K, Y_i)$$

$$X_{N+1} = Y_N \oplus D(K, Y_{N+1})$$

But X_{N+1} is the hash code:

$$X_{N+1} = X_1 \oplus X_2 \oplus ... \oplus X_N$$

$$= [IV \oplus D(K, Y_1)] \oplus [Y_1 \oplus D(K, Y_2)] \oplus ... \oplus [Y_{N1} \oplus ... \oplus D(K, Y_N)]$$

Because the terms in the preceding equation can be XORed in any order, it follows that the hash code would not change if the ciphertext blocks were permuted.

Block Chaining Techniques

A number of proposals have been made for hash functions based on using a cipher block chaining technique, but without the secret key. In one of the first such proposals a message M is divided into fixed-size

blocks M_1, M_2,..., M_N and use a symmetric encryption system such as DES to compute the hash code G as follows:

$$H_o = \text{initial value}$$

$$H_i = E(M_i, H_i, H_{i1})$$

$$G = H_N$$

This is similar to the CBC technique, but in this case there is no secret key. As with any hash code, this scheme is subject to the birthday attack, and if the encryption algorithm is DES and only a 64-bit hash code is produced, then the system is vulnerable.

Furthermore, another version of the birthday attack can be used even if the opponent has access to only one message and its valid signature and cannot obtain multiple signings. Here is the scenario; we assume that the opponent intercepts a message with a signature in the form of an encrypted hash code and that the unencrypted hash code is m bits long:

1. Use the algorithm defined at the beginning of this subsection to calculate the unencrypted hash code G.
2. Construct any desired message in the form Q_1, Q_2,..., Q_{N2}.
3. Compute for $H_i = E(Q_i, H_{i1})$ for $1 \leq i \leq (N\ 2)$.
4. Generate $2^{m/2}$ random blocks; for each block X, compute $E(X, H_{N2})$. Generate an additional $2^{m/2}$ random blocks; for each block Y, compute D(Y, G), where D is the decryption function corresponding to E.
5. Based on the birthday paradox, with high probability there will be an X and Y such that $E(X, H_{N2}) = D(Y, G)$.
6. Form the message Q_1, Q_2,..., Q_{N2}, X, Y. This message has the hash code G and therefore can be used with the intercepted encrypted signature.

This form of attack is known as a **meet-in-the-middle attack**.

$$H_i = E(M_i, H_{i1}) \oplus H_{i1}$$

Security of Hash Functions and Macs

Just as with symmetric 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 hash 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.

Message Authentication Codes

A brute-force attack on a MAC is a more difficult undertaking because it requires known message-MAC pairs. To attack a hash code, we 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 a 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, 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 2^k.

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 2^n. 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.

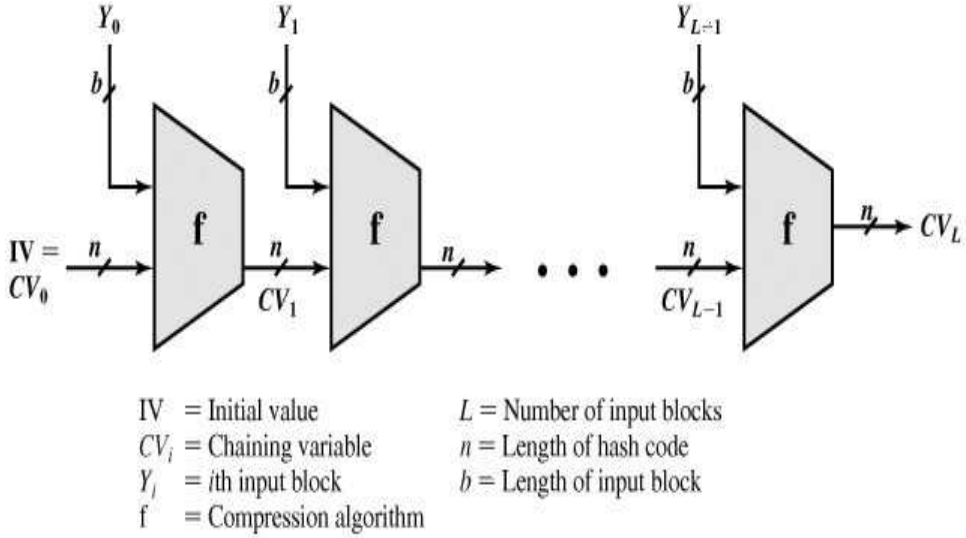
To summarize, the level of effort for brute-force attack on a MAC algorithm can be expressed as min(2^k, 2^n). The assessment of strength is similar to that for symmetric encryption algorithms. It would appear reasonable to require that the key length and MAC length satisfy a relationship such as min(k, 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 or MAC algorithm will require a cryptanalytic effort greater than or equal to the brute-force effort.

Hash Functions

In recent years, there has been considerable effort, and some successes, in developing cryptanalytic attacks on hash functions. Figure 3.2.5 shows the general structure of Secure Hash Code. This structure is also referred to as an iterated hash function was proposed by Merkle. This is the structure of most hash functions in use today, including SHA and Whirlpool. The hash function takes an input message and partitions it into L fixed-sized blocks 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.



IV = Initial value
CV_i = Chaining variable
Y_i = ith input block
f = Compression algorithm
L = Number of input blocks
n = Length of hash code
b = Length of input block

Fig 3.2.5 General Structure of Secure Hash Code

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. Often, b > n; hence the term compression. The hash function can be summarized as follows:

$$CV_o = IV = \text{initial n-bit value}$$

$$CV_i = f(CV_{i1}, Y_{i1})\ 1 \le i \le L$$

$$H(M) = CV_L$$

where the input to the hash function is a message M consisting of the blocks $Y_o, Y_1,..., Y_{L1.}$

The motivation for this iterative structure stems from the observation that if the compression function is collision resistant, then so is the resultant iterated hash function. Therefore, the structure can be used to 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. For any hash function there must exist collisions, because we are mapping a message of length at least equal to twice the block size b (because we must append a length field) into a hash code of length n, where b ≥ n. What is required is that it is computationally infeasible to find collisions.

SECURE HASH ALGORITHM (SHA)

The Secure Hash Algorithm (SHA) was developed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard and is generally referred to as SHA-1.

SHA-1 produces a hash value of 160 bits. In 2002, NIST defined three new versions of SHA, with hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA-512 (Table 3.2.1). These new versions have the same underlying structure and use the same types of modular arithmetic and logical binary operations as SHA-1. In 2005, NIST announced the intention to phase out approval of SHA-1 and move to a reliance on the other SHA versions by 2010. Shortly thereafter, a research team described an attack in which two separate messages could be found that deliver the same SHA-1 hash using 2^{69} operations, far fewer than the 2^{80} operations previously thought needed to find a collision with an SHA-1 hash. This result should hasten the transition to the other versions of SHA.

	SHA-1	SHA-256	SHA-384	SHA-512
Message digest size	160	256	384	512
Message size	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$
Block size	512	512	1024	1024
Word size	32	32	64	64
Number of steps	80	64	80	80
Security	80	128	192	256

Notes: 1. All sizes are measured in bits.
2. Security refers to the fact that a birthday attack on a message digest of size n produces a collision with a workfactor of approximately $2^{n/2}$

Table 3.2.1 Comparison of SHA Parameters

SHA-512 Logic

The algorithm takes as input a message with a maximum length of less than 2^{128} bits and produces as output a 512-bit message digest. The input is processed in 1024-bit blocks. Fig 3.2.6 depicts the overall processing of a message to produce a digest.

This follows the general structure depicted in Figure 3.2.5. The processing consists of the following steps:

- **Step 1: Append padding bits**. The message is padded so that its length is congruent to 896 modulo 1024 [length ≡ 896 (mod 1024)]. Padding is always added, even if the message is already of the desired length. Thus, the number of padding bits is in the range of 1 to 1024. The padding consists of a single 1-bit followed by the necessary number of 0-bits.
- **Step 2: Append length**. A block of 128 bits is appended to the message. This block is treated as an unsigned 128-bit integer (most significant byte first) and contains the length of the original message (before the padding).

The outcome of the first two steps yields a message that is an integer multiple of 1024 bits in length. In Figure 3.2.7, the expanded message is represented as the sequence of 1024-bit blocks M_1, M_2,..., M_N, so that the total length of the expanded message is N x 1024 bits.

- **Step 3: Initialize hash buffer.** A 512-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as eight 64-bit registers (a, b, c, d, e, f, g, h). These registers are initialized to the following 64-bit integers (hexadecimal values):

a=6A09E667F3BCC908
b=BB67AE8584CAA73B
c=3C6EF372FE94F82B
d=A54FF53A5F1D36F1
e=510E527FADE682D1
f=9B05688C2B3E6C1F

g=1F83D9ABFB41BD6B
h = 5BE0CDI9137E2179

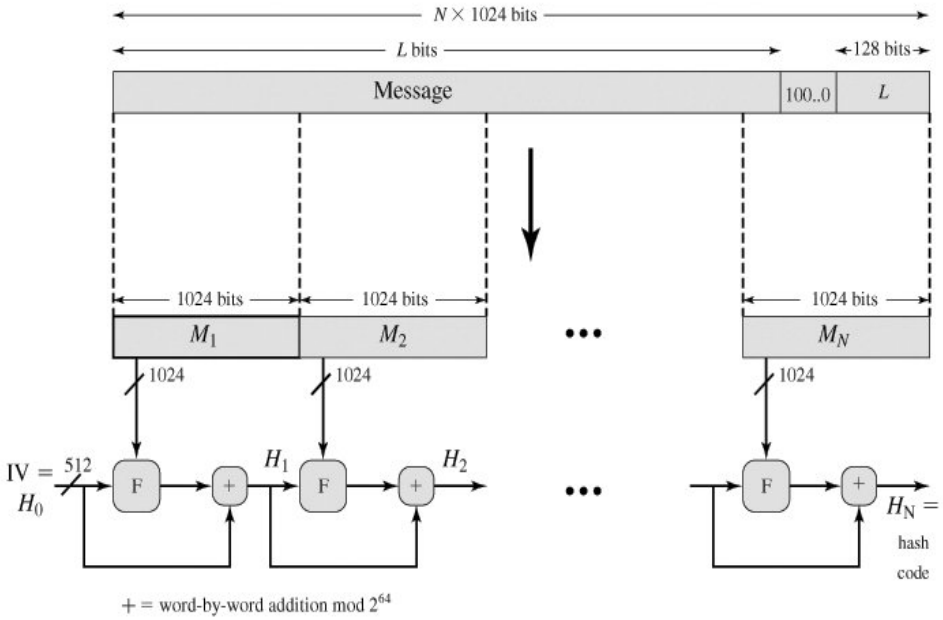


Figure 3.2.7 Message Digest Generation Using SHA-512

These values are stored in big-endian format, which is the most significant byte of a word in the low-address (leftmost) byte position. These words were obtained by taking the first sixty-four bits of the fractional parts of the square roots of the first eight prime numbers.

- **Step 4: Process message in 1024-bit (128-word) blocks**. The heart of the algorithm is a module that consists of 80 rounds; this module is labeled F in Figure 3.2.7. The logic is illustrated in Figure 3.2.8.

Each round takes as input the 512-bit buffer value abcdefgh, and updates the contents of the buffer. At input to the first round, the buffer has the value of the intermediate hash value H_{i-1}. Each round t makes use of a 64-bit value W_t derived from the current 1024-bit block being processed (M_i).These values are derived using a message schedule described subsequently. Each round also makes use of an additive constant K_t where $0 \leq t \leq 79$ indicates one of the 80 rounds. These words represent the first sixty-four bits of the fractional parts of the cube roots of the first eighty prime numbers. The constants provide a "randomized" set of 64-bit patterns, which should eliminate any regularities in the input data.

The output of the eightieth round is added to the input to the first round (H_{i-1})to produce H_i. The addition is done independently for each of the eight words in the buffer with each of the corresponding words in H_{i-1} using addition modulo 2^{64}.

- **Step 5: Output.** After all N 1024-bit blocks have been processed, the output from the Nth stage is the 512-bit message digest.

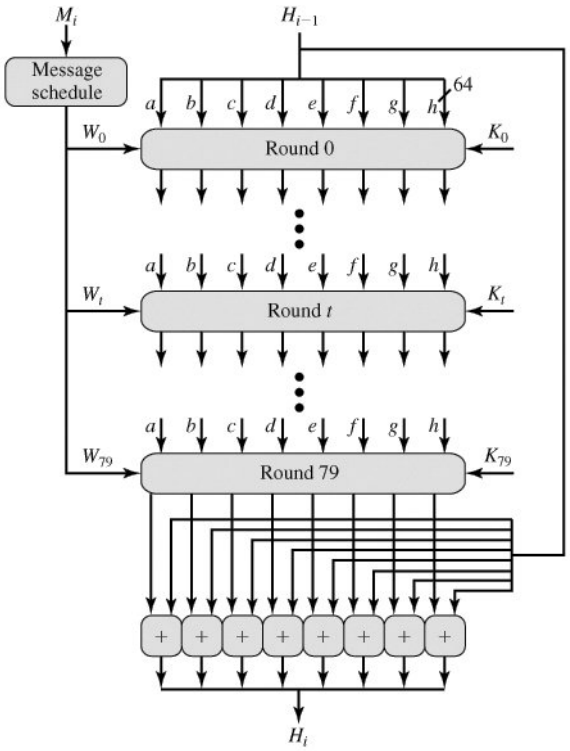


Figure 3.2.8 SHA-512 Processing of a Single 1024-Bit Block

We can summarize the behavior of SHA-512 as follows:

$$H_0 = IV$$

$$H_i = SUM_{64}(H_{i-1}, abcdefgh_i)$$

$$MD = H_N$$

where

IV = initial value of the abcdefgh buffer, defined in step 3

$abcdefgh_i$ = the output of the last round of processing of the ith message block

N = the number of blocks in the message (including padding and length fields)

SUM_{64} = Addition modulo 2^{64} performed separately on each word of the pair of inputs

MD = final message digest value

SHA-512 Round Function

The logic in each of the 80 steps is depicted in Fig 3.2.9.

Each round is defined by the following set of equations:

$$T_1 = h + Ch(e, f, g) + \left(\sum_{1}^{512} e\right) + W_t + K_t$$

$$T_2 = \left(\sum_{0}^{512} a\right) + Maj(a, b, c)$$

$$a = T_1 + T_2$$

$$b = a$$

$$c = b$$

$$d = c$$

$$e = d + T_1$$

$$f = e$$

$$g = f$$

$$h = g$$

Where

t = step number; $0 \le t \le 79$

Ch(e, f, g) = (e AND f) ⊕(NOT e AND g) the conditional function: If e then f else g

Maj(a, b, c) = (a AND b) ⊕(a AND c) (b AND c) the function is true only of the majority (two or three) of the arguments are true

$$\left(\sum_{0}^{512} a\right) = ROTR^{28}(a) \oplus ROTR^{34}(a) \oplus ROTR^{39}(a)$$

$$\left(\sum_{1}^{512} e\right) = ROTR^{14}(e) \oplus ROTR^{18}(e) \oplus ROTR^{41}(e)$$

$ROTR^n(x)$ = circular right shift (rotation) of the 64-bit argument x by n bits

W_t = a 64-bit word derived from the current 512-bit input block

K_t = a 64-bit additive constant

\+ = addition modulo 2^{64}

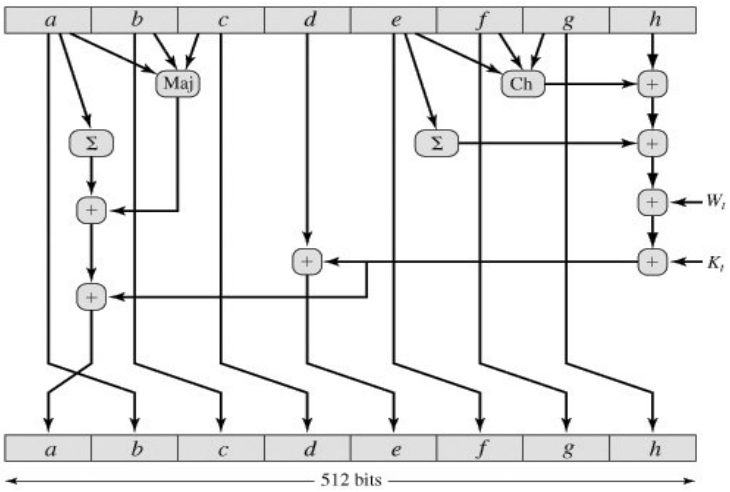


Fig 3.2.9 Elementary SHA-512 Operation (single round)

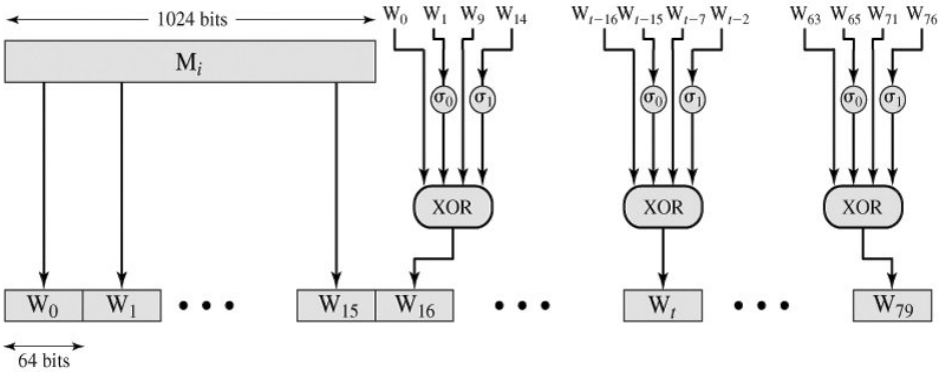


Fig 3.2.10 Creation of 80-word Input Sequence for SHA-512 Processing of Single Block

Thus, in the first 16 steps of processing, the value of W_t is equal to the corresponding word in the message block. For the remaining 64 steps, the value of W_t consists of the circular left shift by one bit of the XOR of four of the preceding values of W_t, with two of those values subjected to shift and rotate operations. This introduces a great deal of redundancy and interdependence into the message blocks that are compressed, which complicates the task of finding a different message block that maps to the same compression function output.

HMAC

In recent years, there has been increased interest in developing a MAC derived from a cryptographic hash function. The motivations for this interest are

1. Cryptographic hash functions such as MD5 and SHA-1 generally execute faster in software than symmetric block ciphers such as DES.
2. Library code for cryptographic hash functions is widely available.

With the development of AES and the more widespread availability of code for encryption algorithms, these considerations are less significant, but hash-based MACs continue to be widely used.

A hash function such as SHA was not designed for use as a MAC and cannot be used directly for that purpose because it does not rely on a secret key. There have been a number of proposals for the incorporation of a secret key into an existing hash algorithm. The approach that has received the most support is HMAC. HMAC has been issued as RFC 2104, has been chosen as the mandatory-to-implement MAC for IP security, and is used in other Internet protocols, such as SSL. HMAC has also been issued as a NIST standard (FIPS 198).

HMAC Design Objectives

The following are the design objectives for HMAC:

- To use, without modifications, available hash functions. In particular, hash functions that perform well in software, and for which code is freely and widely available.

- To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required.
- To preserve the original performance of the hash function without incurring a significant degradation.
- To use and handle keys in a simple way.
- To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions about the embedded hash function.

The first two objectives are important to the acceptability of HMAC. HMAC treats the hash function as a "black box." This has two benefits. First, an existing implementation of a hash function can be used as a module in implementing HMAC. In this way, the bulk of the HMAC code is prepackaged and ready to use without modification. Second, if it is ever desired to replace a given hash function in an HMAC implementation, all that is required is to remove the existing hash function module and drop in the new module. This could be done if a faster hash function were desired. More important, if the security of the embedded hash function were compromised, the security of HMAC could be retained simply by replacing the embedded hash function with a more secure one (e.g., replacing SHA with Whirlpool).

The last design objective in the preceding list is, in fact, the main advantage of HMAC over other proposed hash-based schemes. HMAC can be proven secure provided that the embedded hash function has some reasonable cryptographic strengths. We return to this point later in this section, but first we examine the structure of HMAC.

HMAC Algorithm

Fig 3.2.11 illustrates the overall operation of HMAC.

Define the following terms:

H = embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)

IV = initial value input to hash function

M = message input to HMAC(including the padding specified in the embedded hash function)

Y_i = ith block of M, $0 \leq i \leq (L - 1)$

L = number of blocks in M

b = number of bits in a block

n = length of hash code produced by embedded hash function

K = secret key recommended length is ≥ n; if key length is greater than b; the key is input to the hash function to produce an n-bit key

K^+ = K padded with zeros on the left so that the result is b bits in length

H = embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)

ipad = 00110110 (36 in hexadecimal) repeated b/8 times

opad = 01011100 (5C in hexadecimal) repeated b/8 times

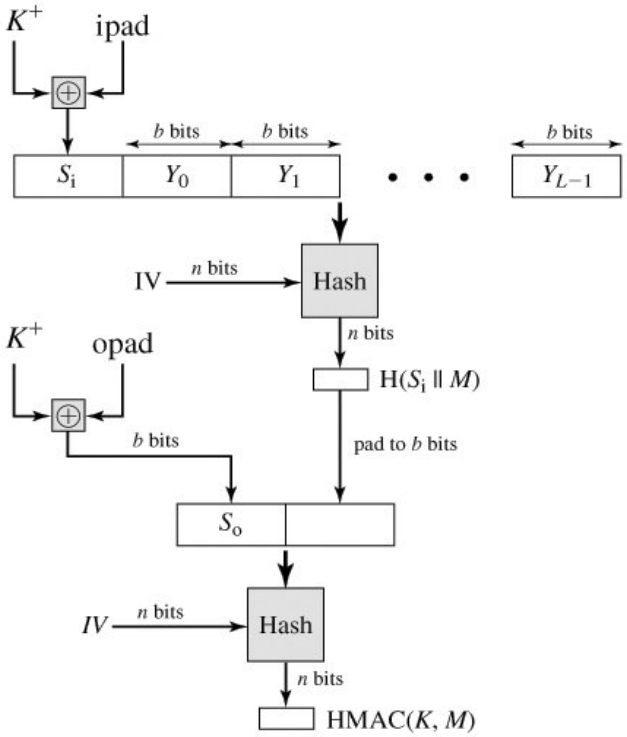


Fig 3.2.11 HMAC Structure

Then HMAC can be expressed as follows:

HMAC(K,M) = H[(K^+ ⊕opad)||H[(K^+ ⊕ipad)||M]]

In words,

1. Append zeros to the left end of K to create a b-bit string K^+(e.g., if K is of length 160 bits and b = 512 then K will be appended with 44 zero bytes 0 x 00).
2. XOR (bitwise exclusive-OR) K^+ with ipad to produce the b-bit block S_i.
3. Append M to S_i.
4. Apply H to the stream generated in step 3.
5. XOR K^+ with opad to produce the b-bit block S_o
6. Append the hash result from step 4 to S_o
7. Apply H to the stream generated in step 6 and output the result.

Note that the XOR with ipad results in flipping one-half of the bits of K. Similarly, the XOR with opad results in flipping one-half of the bits of K, but a different set of bits. In effect, by passing S_i and S_o through the compression function of the hash algorithm, we have pseudorandomly generated two keys from K.

HMAC should execute in approximately the same time as the embedded hash function for long messages. HMAC adds three executions of the

hash compression function (for S_i,S_o and the block produced from the inner hash).

A more efficient implementation is possible, as shown in Fig 3.2.12. Two quantities are precomputed:

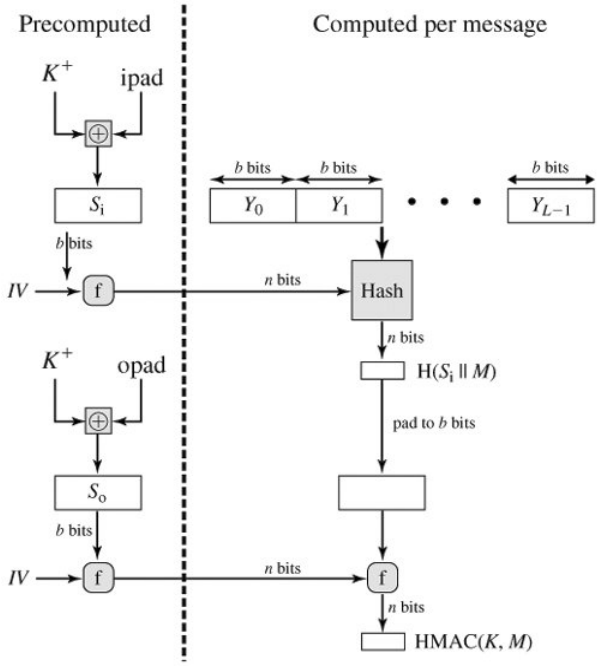


Fig 3.2.12 Efficient Implementation of HMAC

$$f(IV, (K^+ \oplus ipad))$$
$$f(IV, (K^+ \oplus opad))$$

where f(cv, block) is the compression function for the hash function, which takes as arguments a chaining variable of n bits and a block of b bits and produces a chaining variable of n bits. These quantities only need to be computed initially and every time the key changes. In effect, the precomputed quantities substitute for the initial value (IV) in the hash function. With this implementation, only one additional instance of the compression function is added to the processing normally produced by the hash function. This more efficient implementation is especially worthwhile if most of the messages for which a MAC is computed are short.

CMAC

Cipher-based Message Authentication Code (CMAC) mode of operation has been adopted by NIST for use with AES and triple DES. It is specified in NIST Special Publication 800-38B.

Consider the operation of CMAC when the message is an integer multiple n of the cipher block length b. For AES, b = 128 and for triple DES, b = 64. The message is divided into n blocks, M_1, M_2,...,M_n. The algorithm makes use of a k-bit encryption key K and an n-bit constant K_1. For AES, the key size k is 128, 192, or 256 bits; for triple DES, the key size is 112 or 168 bits. CMAC is calculated as follows (Figure 3.2.13a):

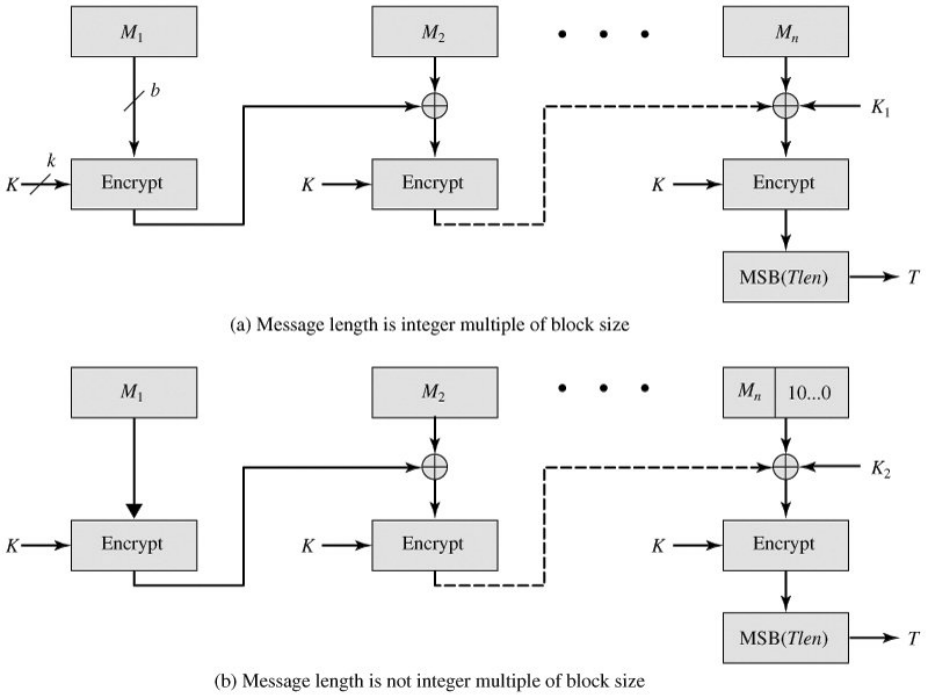


Fig 3.2.13 Cipher-Based Message Authentication Code (CMAC)

$$C_1 = E(K,M_1)$$

$$C_2 = E(K,[M_2 \oplus C_1])$$

$$C_3 = E(K,[M_3 \oplus C_2])$$

.

.

.

$$C_n = E(K,[M_n \oplus C_{n1} \oplus K_1])$$

$$T = MSB_{Tlen}(C_n)$$

where

T = message authentication code, also referred to as the tag

Tlen = bit length of T

$MSB_s(X)$ = the s leftmost bits of the bit string X

If the message is not an integer multiple of the cipher block length, then the final block is padded to the right (least significant bits) with a 1 and as many 0s as necessary so that the final block is also of length b. The

CMAC operation then proceeds as before, except that a different n-bit key K_2 is used instead of K_1.

The two n-bit keys are derived from the k-bit encryption key as follows:

$$\mathbf{L = E(K, 0^n)}$$

$$\mathbf{K_1 = L \cdot x}$$

$$\mathbf{K_2 = L \cdot x^2 = (L \cdot x) \cdot x}$$

where multiplication (·) is done in the finite field (2^n) and x and x^2 are first and second order polynomials that are elements of GF(2^n) Thus the binary representation of x consists of n - 2 zeros followed by 10; the binary representation of x^2 consists of n - 3 zeros followed by 100. The finite field is defined with respect to an irreducible polynomial that is lexicographically first among all such polynomials with the minimum possible number of nonzero terms. For the two approved block sizes, the polynomials are and $x^{64} + x^4 + x^3 + x + 1$ and $x^{128} + x^7 + x^2 + x + 1$.

To generate K_1 and K_2 the block cipher is applied to the block that consists entirely of 0 bits. The first subkey is derived from the resulting cipher text by a left shift of one bit, and, conditionally, by XORing a constant that depends on the block size. The second subkey is derived in the same manner from the first subkey.

REFERENCES

1. William Stallings, Cryptography and Network Security, PHI Publishers
2. www.wikipedia.org

Review Questions:

1. Explain about authentication algorithms
2. Explain about HMAC and CMAC algorithms in detail
3. Write a short note on message authentication codes

3. DIGITAL SIGNATURES

OBJECTIVE

An important type of authentication is the digital signature. This lesson examines the techniques used to construct digital signatures and looks at an important standard, the Digital Signature Standard (DSS). The various authentication techniques based on digital signatures are building blocks in putting together authentication algorithms. The design of such algorithms involves the analysis of subtle attacks that can defeat many apparently secure protocols.

INTRODUCTION

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.

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 Fig 3.2.3. 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 verify the author and the date and time of the signature.
- It must to authenticate the contents at the time of the signature.
- It 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 3.2.3c or d, satisfies these requirements.

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

- **direct**
- **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 or by encrypting a hash code of the message with the sender's private key.

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 (symmetric encryption). 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 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 follows.

- 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. The use of a trusted system might satisfy this requirement.

Table 3.3.1gives several examples of arbitrated digital signatures. In the first, symmetric 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 ID_X 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 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:

$E(K_{ay}, [ID_X||M||E(K_{xa}, [ID_X||H(M)])])$

(1) $X \rightarrow A$: $M\|\|E(K_{xa}, [ID_X\|\|H(M)])$	
(2) $A \rightarrow Y$: $E(K_{ay}, [ID_X\|\|M\|\|E(K_{xa}, [ID_X\|\|H(M)])\|\|T])$	
(a) Conventional Encryption, Arbiter Sees Message	
(1) $X \rightarrow A$: $ID_X\|\|E(K_{xy}, M)\|\|E(K_{xa}, [ID_X\|\|H(E(K_{xy}, M))])$	
(2) $A \rightarrow Y$: $E(K_{ay},[ID_X\|\|E(K_{xy}, M)])\|\|E(K_{xa}, [ID_X\|\|H(E(K_{xy}, M))\|\|T])$	
(b) Conventional Encryption, Arbiter Does Not See Message	
(1) $X \rightarrow A$: $ID_X\|\|E(PR_x, [ID_X\|\|E(PU_y, E(PR_x, M))])$	
(2) $A \rightarrow Y$: $E(PR_a, [ID_X\|\|E(PU_y, E(PR_x, M))\|\|T])$	
(c) Public-Key Encryption, Arbiter Does Not See Message	
Notation:	
X	= sender
Y	= recipient
A	= Arbiter

(1) X → A: M||E(K_{xa}, [ID_X||H(M)])	
M	= message
T	= timestamp

Table 3.3.1 Arbitrated Digital Signature Techniques

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

  $E(K_{xa}, [ID_X||H(M)])$.
- Y must trust A to send $E(K_{ay}, [ID_X||M||E(K_{xa}, [ID_X||H(M)])||T])$ 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 3.3.1b 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 3.3.1c. In this case, X double encrypts a message M first with X's private key, PR_x and then with Y's public key, PU_y. This is a signed, secret version of the message. This signed message, together with X's identifier, is encrypted again with PR_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 PR_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 PR_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 PR_x is compromised, assuming that PR_a is not compromised. Finally, the content of the message from X to Y is secret from A and anyone else. However, this final scheme involves encryption of the message twice with a public-key algorithm. We discuss more practical approaches subsequently.

Authentication Protocols

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

Mutual authentication protocols constitute most important applications. Such protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys.

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.

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 timestamped 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 symmetric 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 general 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.

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

Symmetric Encryption Approaches

A two-level hierarchy of symmetric encryption keys can be used to provide confidentiality for communication in a distributed environment. In general, this strategy involves the use of a trusted key distribution center (KDC). Each party in the network shares a secret key, known as a master key, with the KDC. The KDC is responsible for generating keys to be used for a short time over a connection between two parties, known as session keys, and for distributing those keys using the master keys to protect the distribution. This approach is quite common. As an example, we look at the Kerberos system.

Fig 3.3.1 illustrates a proposal initially put forth by Needham/Schroeder for secret key distribution using a KDC that includes authentication features. The protocol can be summarized as follows:

1. $A \rightarrow KDC: \quad ID_A||ID_B||N_1$
2. $KDC \rightarrow A: \quad E(K_a, [K_s||ID_B||N_1||E(K_b, [K_s||ID_A])])$
3. $A \rightarrow B: \quad E(K_b, [K_s||ID_A])$

1. A → KDC: $ID_A||ID_B||N_1$
4. A → A: $E(K_s, N_2)$
5. A → B: $E(K_s, f(N_2))$

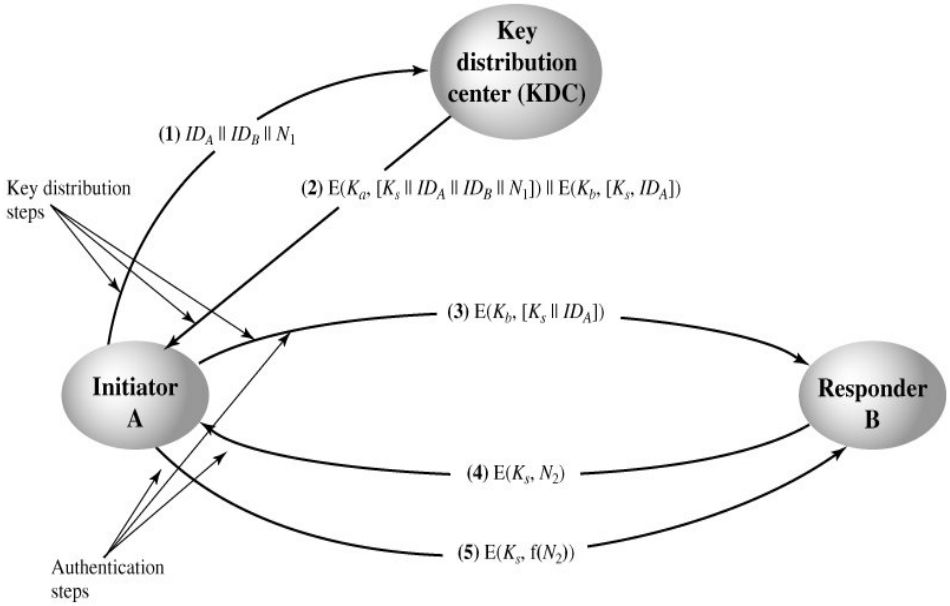


Fig 3.3.1 Secret Key Distribution Using a KDC

Secret keys K_a and K_b are shared between A and the KDC and B and the KDC, respectively. The purpose of the protocol is to distribute securely a session key K_s to A and B. A securely acquires a new session key in step 2. The message in step 3 can be decrypted, and hence understood, only by B. Step 4 reflects B's knowledge of K_s, and step 5 assures B of A's knowledge of K_s and assures B that this is a fresh message because of the use of the nonce N_2. The purpose of steps 4 and 5 is to prevent a certain type of replay attack. In particular, if an opponent is able to capture the message in step 3 and replay it, this might in some fashion disrupt operations at B.

Despite the handshake of steps 4 and 5, the protocol is still vulnerable to a form of replay attack. Suppose that an opponent, X, has been able to compromise an old session key. Admittedly, this is a much more unlikely occurrence than that an opponent has simply observed and recorded step 3. Nevertheless, it is a potential security risk. X can impersonate A and trick B into using the old key by simply replaying step 3. Unless B remembers indefinitely all previous session keys used with A, B will be unable to determine that this is a replay. If X can intercept the handshake message, step 4, then it can impersonate A's response, step 5. From this point on, X can send bogus messages to B that appear to B to come from A using an authenticated session key.

Denning proposes to overcome this weakness by a modification to the Needham/Schroeder protocol that includes the addition of a timestamp to steps 2 and 3. Her proposal assumes that the master keys, K_a and K_b are secure, and it consists of the following steps:

1. $A \rightarrow KDC: \quad ID_A||ID_B$
2. $KDC \rightarrow A: \quad E(K_a, [K_s||ID_B||T||E(K_b, [K_s||ID_A||T])])$
3. $A \rightarrow B: \quad E(K_b, [K_s||ID_A||T])$
4. $B \rightarrow A: \quad E(K_s, N_1)$
5. $A \rightarrow B: \quad E(K_s, f(N_1))$

T is a timestamp that assures A and B that the session key has only just been generated. Thus, both A and B know that the key distribution is a fresh exchange. A and B can verify timeliness by checking that

$$\textbf{|Clock T| < } \Delta t_1 + \Delta t_2$$

where Δt_1 is the estimated normal discrepancy between the KDC's clock and the local clock (at A or B) and Δt_2 is the expected network delay time. Each node can set its clock against some standard reference source. Because the timestamp T is encrypted using the secure master keys, an opponent, even with knowledge of an old session key, cannot succeed because a replay of step 3 will be detected by B as untimely.

Steps 4 and 5 were not included in the original presentation but were added later. These steps confirm the receipt of the session key at B.

The Denning protocol seems to provide an increased degree of security compared to the Needham/Schroeder protocol. However, a new concern is that this new scheme requires reliance on clocks that are synchronized throughout the network. A risk is involved based on the fact that the distributed clocks can become unsynchronized as a result of sabotage on or faults in the clocks or the synchronization mechanism. The problem occurs when a sender's clock is ahead of the intended recipient's clock. In this case, an opponent can intercept a message from the sender and replay it later when the timestamp in the message becomes current at the recipient's site. This replay could cause unexpected results. Gong refers to such attacks as **suppress-replay attacks**.

One way to counter suppress-replay attacks is to enforce the requirement that parties regularly check their clocks against the KDC's clock. The other alternative, which avoids the need for clock synchronization, is to rely on handshaking protocols using nonces. This latter alternative is not vulnerable to a suppress-replay attack because the nonces the recipient will choose in the future are unpredictable to the sender. The Needham/Schroeder protocol relies on nonces only but, as we have seen, has other vulnerabilities.

An attempt is made to respond to the concerns about suppress-replay attacks and at the same time fix the problems in the Needham/Schroeder protocol. Subsequently, an inconsistency in this latter protocol was noted and an improved strategy was presented. The protocol is as follows:

1. $A \rightarrow B: \quad ID_A||N_a$

1. $A \rightarrow B$: $ID_A||N_a$
2. $B \rightarrow KDC$: $ID_B||N_b||E(K_b, [ID_A||N_a||T_b])$
3. $KDC \rightarrow A$: $E(K_a, [ID_B||N_a||K_s||T_b])||E(K_b,[ID_A||K_s||T_b])||N_b$
4. $A \rightarrow B$: $E(K_b, [ID_A||K_s||T_b])||E(K_s, N_b)$

Let us follow this exchange step by step.

1. A initiates the authentication exchange by generating a nonce, N_a, and sending that plus its identifier to B in plaintext. This nonce will be returned to A in an encrypted message that includes the session key, assuring A of its timeliness.

2. B alerts the KDC that a session key is needed. Its message to the KDC includes its identifier and a nonce, N_b This nonce will be returned to B in an encrypted message that includes the session key, assuring B of its timeliness. B's message to the KDC also includes a block encrypted with the secret key shared by B and the KDC. This block is used to instruct the KDC to issue credentials to A; the block specifies the intended recipient of the credentials, a suggested expiration time for the credentials, and the nonce received from A.

3. The KDC passes on to A B's nonce and a block encrypted with the secret key that B shares with the KDC. The block serves as a "ticket" that can be used by A for subsequent authentications, as will be seen. The KDC also sends to A a block encrypted with the secret key shared by A and the KDC. This block verifies that B has received A's initial message (ID_B) and that this is a timely message and not a replay (N_a) and it provides A with a session key (K_s) and the time limit on its use (T_b).

4. A transmits the ticket to B, together with the B's nonce, the latter encrypted with the session key. The ticket provides B with the secret key that is used to decrypt $E(K_s, N_b)$ to recover the nonce. The fact that B's nonce is encrypted with the session key authenticates that the message came from A and is not a replay.

This protocol provides an effective, secure means for A and B to establish a session with a secure session key. Furthermore, the protocol leaves A in possession of a key that can be used for subsequent authentication to B, avoiding the need to contact the authentication server repeatedly. Suppose that A and B establish a session using the aforementioned protocol and then conclude that session. Subsequently, but within the time limit established by the protocol, A desires a new session with B. The following protocol ensues:

1. $A \rightarrow B$: $E(K_b, [ID_A||K_s||T_b])||N'_a$

1. $A \rightarrow B: \quad E(K_b, [ID_A||K_s||T_b])||N'_a$
2. $B \rightarrow A: \quad N'_b||E(K_s, N'_a)$
3. $A \rightarrow B: \quad E(K_s, N'_b)$

When B receives the message in step 1, it verifies that the ticket has not expired. The newly generated nonces N'_a and N'_b assure each party that there is no replay attack.

In all the foregoing, the time specified in T_b is a time relative to B's clock. Thus, this timestamp does not require synchronized clocks because B checks only self-generated timestamps.

Public-Key Encryption Approaches

A protocol using timestamps is provided in [DENN81]:

1. $A \rightarrow AS: \quad ID_A||ID_B$
2. $AS \rightarrow A: \quad E(PR_{as}, [ID_A||PU_a||T])||E(PR_{as}, [ID_B||PU_b||T])$
3. $A \rightarrow B: \quad E(PR_{as}, [ID_A||PU_a||T])||E(PR_{as}, [ID_B||PU_b||T])$ $||E(PU_b, E(PR_a, [K_s||T]))$

In this case, the central system is referred to as an authentication server (AS), because it is not actually responsible for secret key distribution. Rather, the AS provides public-key certificates. The session key is chosen and encrypted by A; hence, there is no risk of exposure by the AS. The timestamps protect against replays of compromised keys.

This protocol is compact but requires synchronization of clocks. Another approach, proposed by Woo and Lam, makes use of nonces. The protocol consists of the following steps:

1. $A \rightarrow KDC: \quad ID_A||ID_B$
2. $KDC \rightarrow A: \quad E(PR_{auth}, [ID_B||PU_b])$
3. $A \rightarrow B: \quad E(PU_b, [N_a||ID_A])$
4. $B \rightarrow KDC: \quad ID_A||ID_B||E(PU_{auth}, N_a)$
5. $KDC \rightarrow B: \quad E(PR_{auth}, [ID_A||PU_a])||E(PU_b, E(PR_{auth}, [N_a||K_s||ID_B]))$
6. $B \rightarrow A: \quad E(PU_a, E(PR_{auth}, [(N_a||K_s||ID_B)||N_b]))$
7. $A \rightarrow B: \quad E(K_s, N_b)$

In step 1, A informs the KDC of its intention to establish a secure connection with B. The KDC returns to A a copy of B's public-key certificate (step 2). Using B's public key, A informs B of its desire to communicate and sends a nonce N_a (step 3). In step 4, B asks the KDC for A's public-key certificate and requests a session key; B includes A's nonce so that the KDC can stamp the session key with that nonce. The nonce is protected using the KDC's public key. In step 5, the KDC returns to B a copy of A's public-key certificate, plus the information $\{N_a, K_s, ID_B\}$.

This information basically says that K_s is a secret key generated by the KDC on behalf of B and tied to N_a; the binding of K_s and N_a will assure A that K_s is fresh. This triple is encrypted, using the KDC's private key, to allow B to verify that the triple is in fact from the KDC. It is also encrypted using B's public key, so that no other entity may use the triple in an attempt to establish a fraudulent connection with A. In step 6, the triple $\{N_a, K_s, ID_B\}$, still encrypted with the KDC's private key, is relayed to A, together with a nonce N_b generated by B. All the foregoing are encrypted using A's public key. A retrieves the session key K_s and uses it to encrypt N_b and return it to B. This last message assures B of A's knowledge of the session key.

1. $A \rightarrow KDC: \quad ID_A||ID_B$
2. $KDC \rightarrow A: \quad E(PR_{auth}, [ID_B||PU_b])$
3. $A \rightarrow B: \quad E(PU_b, [N_a||ID_A])$
4. $B \rightarrow KDC: \quad ID_A||ID_B||E(PU_{auth}, N_a)$
5. $KDC \rightarrow B: \quad E(PR_{auth}, [ID_A||PU_a])||E(PU_b, E(PR_{auth}, [N_a||K_s||ID_A||ID_B])$
6. $B \rightarrow A: \quad E(PU_a, E(PR_{auth}, [(N_a||K_s||ID_A||ID_B)||N_b]))$
7. $A \rightarrow B: \quad E(K_s, N_b)$

The identifier of A, ID_A, is added to the set of items encrypted with the KDC's private key in steps 5 and 6. This binds the session key K_s to the identities of the two parties that will be engaged in the session. This inclusion of ID_A accounts for the fact that the nonce value N_a is considered unique only among all nonces generated by A, not among all nonces generated by all parties. Thus, it is the pair $\{ID_A, N_a\}$ that uniquely identifies the connection request of A.

In both this example and the protocols described earlier, protocols that appeared secure were revised after additional analysis. These examples highlight the difficulty of getting things right in the area of authentication.

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 online 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 is often desirable that the mail-handling protocol not require access to the plaintext form 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.

Symmetric Encryption Approach

Using symmetric encryption, the decentralized key distribution impractical. This scheme requires the sender to issue a request to the intended recipient, await a response that includes a session key, and only then send the message.

With some refinement, the KDC strategy illustrated in Fig 3.3.1 is a candidate for encrypted electronic mail. Because we wish to avoid requiring that the recipient (B) be on line at the same time as the sender (A), steps 4 and 5 must be eliminated. For a message with content M, the sequence is as follows:

1. $A \rightarrow KDC: \quad ID_A||ID_B||N_1$
2. $KDC \rightarrow A: \quad E(K_a, [K_s||ID_B||N_1||E(K_b, [K_s||ID_A])])$
3. $A \rightarrow B: \quad E(K_b, [K_s||ID_A])||E(K_s, M)$

This approach guarantees that only the intended recipient of a message will be able to read it. It also provides a level of authentication that the sender is A. As specified, the protocol does not protect against replays. Some measure of defense could be provided by including a timestamp with the message. However, because of the potential delays in the e-mail process, such timestamps may have limited usefulness.

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, authentication), or both. These approaches require that either the sender know the recipient's public key (confidentiality) or 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:

$$A \rightarrow B: E(PU_b, K_s)||E(K_s, M)$$

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

$$A \rightarrow B: M||E(PR_a, H(M))$$

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 requeues 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:

$$A \rightarrow B: E(PU_b, [M||E(PR_a, H(M))])$$

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

$$A \rightarrow B: M||E(PR_a, H(M))||E(PR_{as}, [T||ID_A||PU_a])$$

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.

Digital Signature Standard

The National Institute of Standards and Technology (NIST) has published Federal Information Processing Standard FIPS 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 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.

Fig 3.3.2 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.

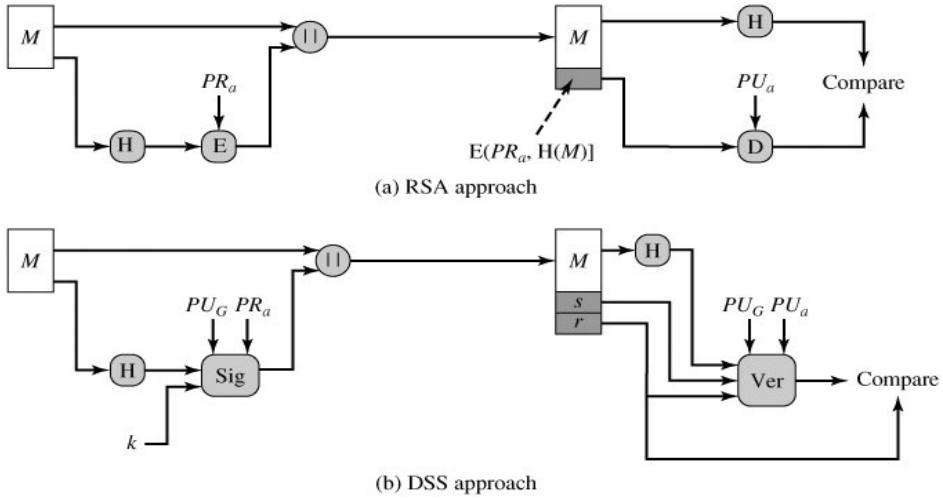


Figure 3.3.2 Two Approaches to Digital Signatures

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 (PR_a)and a set of parameters known to a group of communicating principals. We can consider this set to constitute a global public key (PU_G). The result is a signature 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 (PU_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 .

Table 13.2 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 $h^{(p1)/q}$ mod p where h is an integer between 1 and (p 1) with the restriction that g must be greater than 1.

Global Public-Key Components	
p	prime number where $2^{L-1} < p < 2^L$ for 512 ≤ L ≤ 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	= $h^{(p-1)/q}$ mod p, where h is any integer with 1 < h < (p 1) such that $h^{(p-1)/q}$ mod p > 1

Global Public-Key Components	
User's Private Key	
x	random or pseudorandom integer with 0 < x < q
User's Public Key	
y	= g^x mod p
User's Per-Message Secret Number	
k	= random or pseudorandom integer with 0 < k < q
Signing	
r	= (g^k mod p) mod q
s	= [k^{-1} (H(M) + xr)] mod q
Signature = (r, s)	
Verifying	
w	= (s')$^{-1}$ mod q
u1	= [H(M')w] mod q
u2	=(r')w mod q
v	= [(g^{u1} y^{u2}) mod p] mod 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

Table 3.3.2 The Digital Signature Algorithm (DSA)

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 pseudorandomly. The public key is calculated from the private key as y = g^x mod 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.

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 Table 3.3.2. 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.

Fig 3.3.3 depicts the functions of signing and verifying.

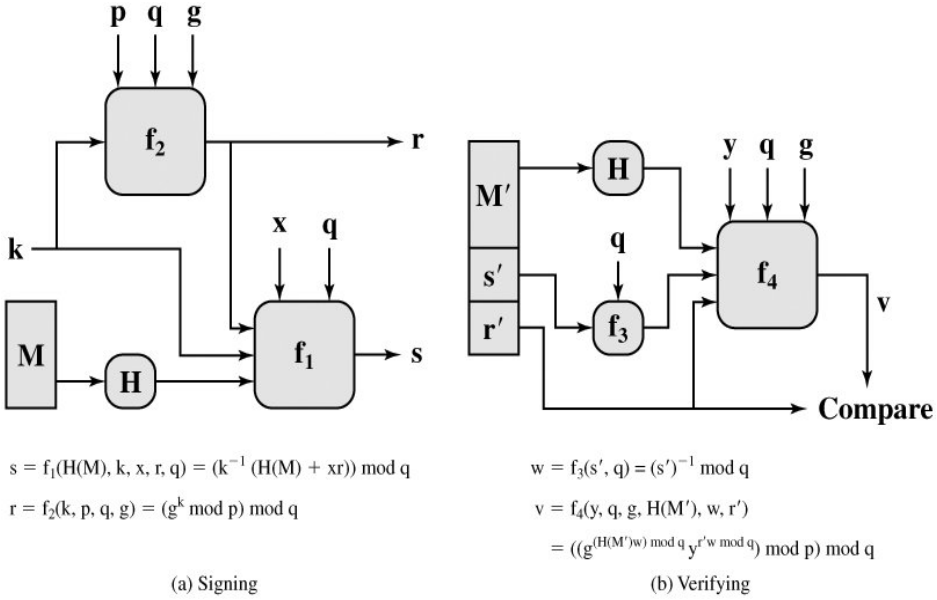


Fig 3.3.3 DSS Signing and Verifying

The structure of the algorithm, as revealed in Fig 3.3.3, 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 q) 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.

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.

REFERENCES

1. William Stallings, Cryptography and Network Security, PHI Publishers
2. www.wikipedia.org

Review Questions:

1. Explain Digital Signature Standard in Detail
2. Explain the Key Distribution mechanism in detail
3. Explain the process of DSS verification and Signing

4. KEY MANAGEMENT SCHEMES

OBJECTIVES

One of the major roles of public-key encryption has been to address the problem of key distribution. The objective of this chapter is to cover two distinct aspects of public-key cryptography, namely, the distribution of public keys and the use of public-key encryption to distribute secret keys

Distribution of Public Keys

Several techniques have been proposed for the distribution of public keys. Virtually all these proposals can be grouped into the following general schemes:

- Public announcement
- Publicly available directory
- Public-key authority
- Public-key certificates

Public Announcement of Public Keys

On the face of it, the point of public-key encryption is that the public key is public. Thus, if there is some broadly accepted public-key algorithm, such as RSA, any participant can send his or her public key to any other participant or broadcast the key to the community at large (Fig3.4.1). For example, because of the growing popularity of PGP which makes use of RSA, many PGP users have adopted the practice of appending their public key to messages that they send to public forums, such as USENET newsgroups and Internet mailing lists.

Fig 3.4.1 Uncontrolled Public-Key Distribution

Although this approach is convenient, it has a major weakness. Anyone can forge such a public announcement. That is, some user could pretend to be user A and send a public key to another participant or broadcast such a public key. Until such time as user A discovers the forgery and alerts other participants, the forger is able to read all encrypted messages intended for A and can use the forged keys for authentication.

Publicly Available Directory

A greater degree of security can be achieved by maintaining a publicly available dynamic directory of public keys. Maintenance and distribution of the public directory would have to be the responsibility of some trusted

entity or organization (Fig 3.4.2). Such a scheme would include the following elements:

1. The authority maintains a directory with a {name, public key} entry for each participant.
2. Each participant registers a public key with the directory authority. Registration would have to be in person or by some form of secure authenticated communication.
3. A participant may replace the existing key with a new one at any time, either because of the desire to replace a public key that has already been used for a large amount of data, or because the corresponding private key has been compromised in some way.
4. Participants could also access the directory electronically. For this purpose, secure, authenticated communication from the authority to the participant is mandatory.

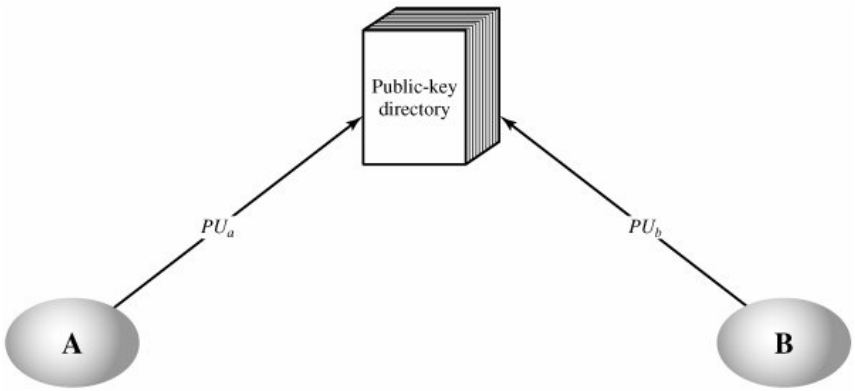


Fig 3.4.2 Public-Key Publication

This scheme is clearly more secure than individual public announcements but still has vulnerabilities. If an adversary succeeds in obtaining or computing the private key of the directory authority, the adversary could authoritatively pass out counterfeit public keys and subsequently impersonate any participant and eavesdrop on messages sent to any participant. Another way to achieve the same end is for the adversary to tamper with the records kept by the authority.

Public-Key Authority

Stronger security for public-key distribution can be achieved by providing tighter control over the distribution of public keys from the directory. A typical scenario is illustrated in Fig 3.4.3. As before, the scenario assumes that a central authority maintains a dynamic directory of public keys of all participants. In addition, each participant reliably knows a public key for the authority, with only the authority knowing the corresponding private key. The following steps occur:

1. A sends a timestamped message to the public-key authority containing a request for the current public key of B.

2. The authority responds with a message that is encrypted using the authority's private key, PR_{auth} Thus, A is able to decrypt the message using the authority's public key. Therefore, A is assured that the message originated with the authority. The message includes the following:

- B's public key, PU_b which A can use to encrypt messages destined for B
- The original request, to enable A to match this response with the corresponding earlier request and to verify that the original request was not altered before reception by the authority
- The original timestamp, so A can determine that this is not an old message from the authority containing a key other than B's current public key

3. A stores B's public key and also uses it to encrypt a message to B containing an identifier of A (ID_A) and a nonce (N_1), which is used to identify this transaction uniquely.

4, B retrieves A's public key from the authority in the same manner as A

5. retrieved B's public key.

At this point, public keys have been securely delivered to A and B, and they may begin their protected exchange. However, two additional steps are desirable:

6. B sends a message to A encrypted with PU_a and containing A's nonce (N_1) as well as a new nonce generated by B (N_2) Because only B could have decrypted message (3), the presence of N_1 in message (6) assures A that the correspondent is B.

7. A returns N_2, encrypted using B's public key, to assure B that its correspondent is A.

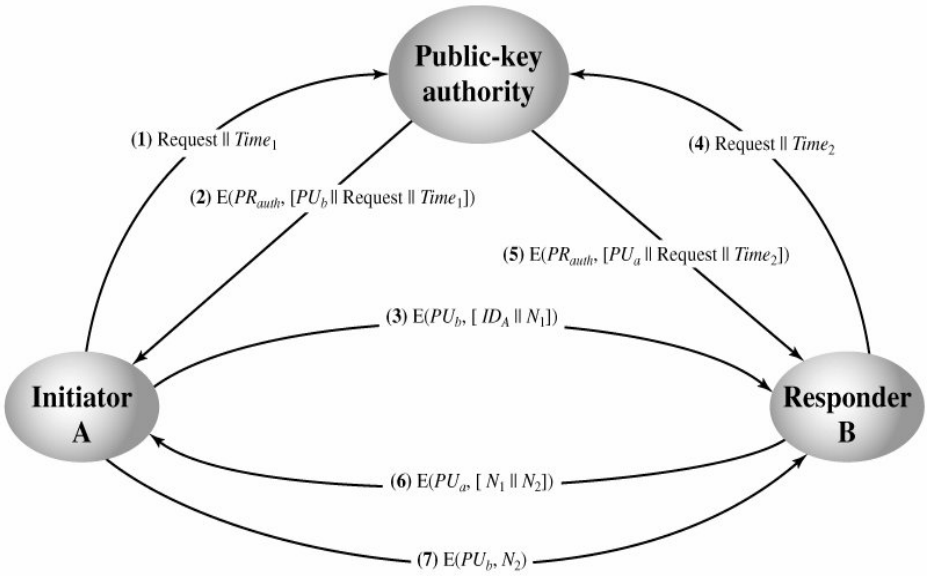


Fig3.4.3 Public-Key Distribution Scenario

Thus, a total of seven messages are required. However, the initial four messages need be used only infrequently because both A and B can

save the other's public key for future use, a technique known as caching. Periodically, a user should request fresh copies of the public keys of its correspondents to ensure currency.

Public-Key Certificates

The scenario of Fig 3.4.3 has some drawbacks. The public-key authority could be somewhat of a bottleneck in the system, for a user must appeal to the authority for a public key for every other user that it wishes to contact. As before, the directory of names and public keys maintained by the authority is vulnerable to tampering.

An alternative approach is to use certificates that can be used by participants to exchange keys without contacting a public-key authority, in a way that is as reliable as if the keys were obtained directly from a public-key authority. In essence, a certificate consists of a public key plus an identifier of the key owner, with the whole block signed by a trusted third party. Typically, the third party is a certificate authority, such as a government agency or a financial institution that is trusted by the user community. A user can present his or her public key to the authority in a secure manner, and obtain a certificate. The user can then publish the certificate. Anyone needed this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature. A participant can also convey its key information to another by transmitting its certificate. Other participants can verify that the certificate was created by the authority. We can place the following requirements on this scheme:

1. Any participant can read a certificate to determine the name and public key of the certificate's owner.
2. Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.
3. Only the certificate authority can create and update certificates.
4. Any participant can verify the currency of the certificate.

A certificate scheme is illustrated in Fig 3.4.4. Each participant applies to the certificate authority, supplying a public key and requesting a certificate.

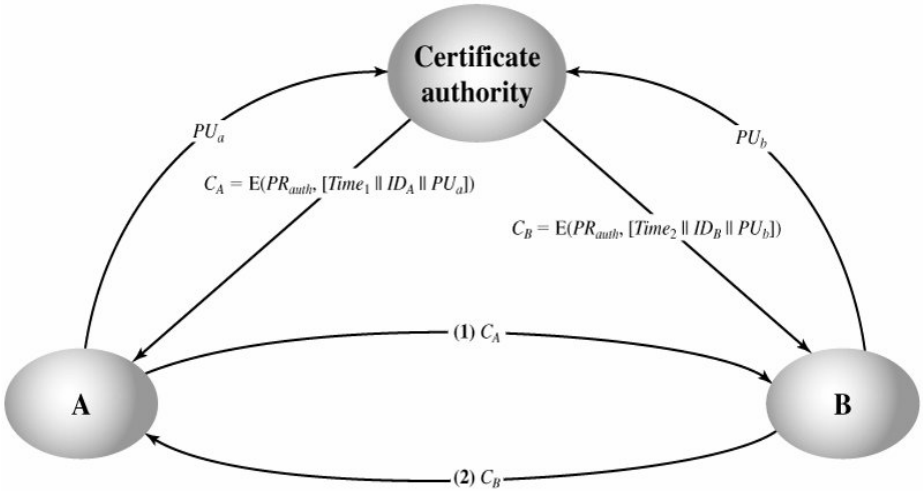


Fig 3.4.4 Exchange of Public-Key Certificates

Application must be in person or by some form of secure authenticated communication. For participant A, the authority provides a certificate of the form

$$C_A = E(PR_{auth}, [T||ID_A||PU_a])$$

where PR_{auth} is the private key used by the authority and T is a timestamp. A may then pass this certificate on to any other participant, who reads and verifies the certificate as follows:

$$D(PU_{auth}, C_A) = D(PU_{auth}, E(PR_{auth}, [T||ID_A||PU_a])) = (T||ID_A||PU_a)$$

The recipient uses the authority's public key, PU_{auth} to decrypt the certificate. Because the certificate is readable only using the authority's public key, this verifies that the certificate came from the certificate authority. The elements ID_A and PU_a provide the recipient with the name and public key of the certificate's holder. The timestamp T validates the currency of the certificate. The timestamp counters the following scenario. A's private key is learned by an adversary. A generates a new private/public key pair and applies to the certificate authority for a new certificate. Meanwhile, the adversary replays the old certificate to B. If B then encrypts messages using the compromised old public key, the adversary can read those messages.

In this context, the compromise of a private key is comparable to the loss of a credit card. The owner cancels the credit card number but is at risk until all possible communicants are aware that the old credit card is obsolete. Thus, the timestamp serves as something like an expiration date. If a certificate is sufficiently old, it is assumed to be expired.

One scheme has become universally accepted for formatting public-key certificates: the X.509 standard. X.509 certificates are used in most network security applications, including IP security, secure sockets layer (SSL), secure electronic transactions (SET), and S/MIME.

Distribution of Secret Keys Using Public-Key Cryptography

Public-key encryption provides for the distribution of secret keys to be used for conventional encryption.

Simple Secret Key Distribution

An extremely simple scheme is illustrated in Fig 3.4.5. If A wishes to communicate with B, the following procedure is employed:

1. A generates a public/private key pair $\{PU_a, PR_a\}$ and transmits a message to B consisting of PU_a and an identifier of A, ID_A.
2. B generates a secret key, K_s, and transmits it to A, encrypted with A's public key.
3. A computes $D(PR_a, E(PU_a, K_s))$ to recover the secret key. Because only A can decrypt the message, only A and B will know the identity of K_s.
4. A discards PU_a and PR_a and B discards PU_a.

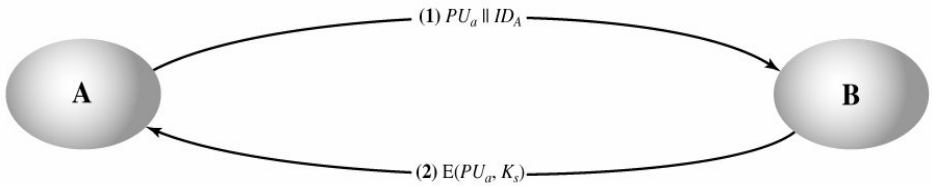


Fig 3.4.5 Simple Use of Public-Key Encryption to Establish a Session Key

A and B can now securely communicate using conventional encryption and the session key K_s. At the completion of the exchange, both A and B discard K_s. Despite its simplicity, this is an attractive protocol. No keys exist before the start of the communication and none exist after the completion of communication. Thus, the risk of compromise of the keys is minimal. At the same time, the communication is secure from eavesdropping.

The protocol depicted above is insecure against an adversary who can intercept messages and then either relay the intercepted message or substitute another message. Such an attack is known as a **man-in-the-middle attack**. In this case, E, has control of the intervening communication channel, then E can compromise the communication in the following fashion without being detected:

1. A generates a public/private key pair $\{PU_a, PR_a\}$ and transmits a message intended for B consisting of PU_a and an identifier of A, ID_A.
2. E intercepts the message, creates its own public/private key pair $\{PU_e, PR_e\}$ and transmits $PU_e||ID_A$ to B.
3. B generates a secret key, K_s, and transmits $E(PU_e, K_s)$.
4. E intercepts the message, and learns K_s by computing $D(PR_e, E(PU_e, K_s))$.
5. E transmits $E(PU_a, K_s)$ to A.

The result is that both A and B know K_s and are unaware that K_s has also been revealed to E. A and B can now exchange messages using K_s E no longer actively interferes with the communications channel but simply eavesdrops. Knowing K_s E can decrypt all messages, and both A and B are unaware of the problem. Thus, this simple protocol is only useful in an environment where the only threat is eavesdropping.

Secret Key Distribution with Confidentiality and Authentication

Figure 3.4.6 provides protection against both active and passive attacks. We begin at a point when it is assumed that A and B have exchanged public keys by one of the schemes described earlier in this section. Then the following steps occur:

1. A uses B's public key to encrypt a message to B containing an identifier of A (ID_A) and a nonce (N_1), which is used to identify this transaction uniquely.

2. B sends a message to A encrypted with PU_a and containing A's nonce (N_1) as well as a new nonce generated by B (N_2) Because only B could have decrypted message (1), the presence of N_1 in message

(2) assures A that the correspondent is B.

3. A returns N_2 encrypted using B's public key, to assure B that its correspondent is A.
4. A selects a secret key K_s and sends $M = E(PU_b, E(PR_a, K_s))$ to B. Encryption of this message with B's public key ensures that only B can read it; encryption with A's private key ensures that only A could have sent it.
5. B computes $D(PU_a, D(PR_b, M))$ to recover the secret key.

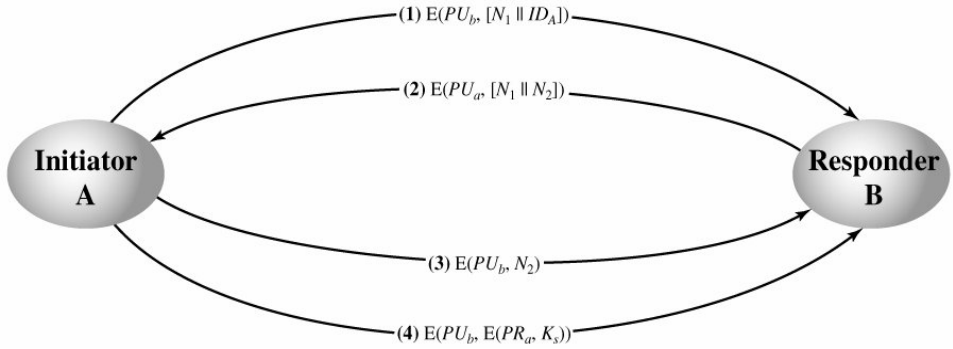


Fig 3.4.6 Public-Key Distribution of Secret Keys

Notice that the first three steps of this scheme are the same as the last three steps of Fig3.4.3. The result is that this scheme ensures both confidentiality and authentication in the exchange of a secret key.

A Hybrid Scheme

Yet another way to use public-key encryption to distribute secret keys is a hybrid approach in use on IBM mainframes. This scheme retains the use of a key distribution center (KDC) that shares a secret master key with each user and distributes secret session keys encrypted with the master key. A public key scheme is used to distribute the master keys. The following rationale is provided for using this three-level approach:

- Performance: There are many applications, especially transaction-oriented applications, in which the session keys change frequently. Distribution of session keys by public-key encryption could degrade overall system performance because of the relatively high computational load of public-key encryption and decryption. With a three-level hierarchy, public-key encryption is used only occasionally to update the master key between a user and the KDC.
- Backward compatibility: The hybrid scheme is easily overlaid on an existing KDC scheme, with minimal disruption or software changes.

The addition of a public-key layer provides a secure, efficient means of distributing master keys. This is an advantage in a configuration in which a single KDC serves a widely distributed set of users.

REFERENCES

1. William Stallings, Cryptography and Network Security, PHI Publishers

Review Questions:

1. Explain the concept of key distribution with confidentiality and authentication

UNIT - IV

1. KERBEROS

2. DIRECTORY AUTHENTICATION SERVICES

3. ELECTRONIC MAIL SECURITY

4. WEB SECURITY

5. SYSTEM SECURITY

6. FIREWALLS

1.KERBEROS

OBJECTIVE

The aim of this lesson is to examine some of the authentication functions that have been developed to support application-level authentication. We begin by looking at one of the earliest and also one of the most widely used services: **Kerberos**. Kerberos is an authentication service designed for use in a distributed environment.

INTRODUCTION

Kerberos is a network authentication protocol. It is designed to provide strong authentication for client/server applications by using secret-key cryptography. *The Internet is an insecure place*. Many of the protocols used in the Internet do not provide any security. Tools to "sniff" passwords off of the network are in common use by malicious hackers. Thus, applications which send an unencrypted password over the network are extremely vulnerable. Some sites attempt to use firewalls to solve their network security problems. Unfortunately, firewalls assume that "the bad guys" are on the outside, which is often a very bad assumption. Most of the really damaging incidents of computer crime are carried out by insiders. Firewalls also have a significant disadvantage in that they restrict how your users can use the Internet. In many places, these restrictions are simply unrealistic and unacceptable.

Kerberos was created by MIT as a solution to these network security problems. The Kerberos protocol uses strong cryptography so that a client can prove its identity to a server (and vice versa) across an insecure network connection. After a client and server has used Kerberos to prove their identity, they can also encrypt all of their communications to assure privacy and data integrity as they go about their business. Kerberos uses as its basis the Needham-Schroeder protocol. It makes use of a trusted third party, termed a key distribution center (KDC), which consists of two logically separate parts: an Authentication Server (AS) and a Ticket Granting Server (TGS). Kerberos works on the basis of "tickets" which serve to prove the identity of users.

The KDC maintains a database of secret keys; each entity on the network, whether a client or a server, shares a secret key known only to itself and to the KDC. Knowledge of this key serves to prove an entity's identity. For communication between two entities, the KDC generates a session key which they can use to secure their interactions.

The first published report on Kerberos listed the following requirements:

- **Secure:** A network eavesdropper should not be able to obtain the necessary information to impersonate a user. More generally, Kerberos should be strong enough that a potential opponent does not find it to be the weak link.
- **Reliable:** For all services that rely on Kerberos for access control, lack of availability of the Kerberos service means lack of availability of the supported services. Hence, Kerberos should be highly reliable and should employ distributed server architecture, with one system able to back up another.
- **Transparent:** Ideally, the user should not be aware that authentication is taking place, beyond the requirement to enter a password.
- **Scalable:** The system should be capable of supporting large numbers of clients and servers. This suggests a modular, distributed architecture.

Kerberos Version 4

Version 4 of Kerberos makes use of DES, in a rather elaborate protocol, to provide the authentication service. We adopt a strategy to build up to the full protocol by looking first at several hypothetical dialogues. Each successive dialogue adds additional complexity to counter security vulnerabilities revealed in the preceding dialogue.

A Simple Authentication Dialogue

In an unprotected network environment, any client can apply to any server for service. The obvious security risk is that of impersonation. An opponent can pretend to be another client and obtain unauthorized privileges on server machines. To counter this threat, servers must be able to confirm the identities of clients who request service. Each server can be required to undertake this task for each client/server interaction, but in an open environment, this places a substantial burden on each server.

An alternative is to use an authentication server (AS) that knows the passwords of all users and stores these in a centralized database. In addition, the AS shares a unique secret key with each server. These keys have been distributed physically or in some other secure manner. Consider the following hypothetical dialogue:

(1) $C \rightarrow AS : \quad ID_C \| P_C \| ID_V$

(2) $C \rightarrow AS : \quad TICKET$

(3) $C \rightarrow V : \quad ID_C \| TICKET$

Ticket = $E(K_v, [ID_C||AD_C||ID_V])$

where,

C = Client

AS = Authentication Server

V = Server

ID_C = Identifier of User on C

ID_V = Identifier of V

P_C = Password of User on C

AD_C = Network Address of C

K_V = Secret Encryption key shared by AS and V

In this scenario, the user logs on to a workstation and requests access to server V. The client module C in the user's workstation requests the user's password and then sends a message to the AS that includes the user's ID, the server's ID, and the user's password. The AS checks its database to see if the user has supplied the proper password for this user ID and whether this user is permitted access to server V. If both tests are passed, the AS accepts the user as authentic and must now convince the server that this user is authentic. To do so, the AS creates a ticket that contains the user's ID and network address and the server's ID. This ticket is encrypted using the secret key shared by the AS and this server. This ticket is then sent back to C. Because the ticket is encrypted, it cannot be altered by C or by an opponent.

With this ticket, C can now apply to V for service. C sends a message to V containing C's ID and the ticket. V decrypts the ticket and verifies that the user ID in the ticket is the same as the unencrypted user ID in the message. If these two match, the server considers the user authenticated and grants the requested service.

Each of the ingredients of message (3) is significant. The ticket is encrypted to prevent alteration or forgery. The server's ID (ID_V) is included in the ticket so that the server can verify that it has decrypted the ticket properly. ID_C is included in the ticket to indicate that this ticket has been issued on behalf of C. Finally, AD_C serves to counter the following threat. An opponent could capture the ticket transmitted in message (2), then use the name ID_C and transmit a message of form (3) from another workstation. The server would receive a valid ticket that matches the user ID and grant access to the user on that other workstation. To prevent this attack, the AS includes in the ticket the network address from which the original request came. Now the ticket is valid only if it is transmitted from the same workstation that initially requested the ticket.

Disadvantages

There are two particular problems in the Simple Authentication Dialogue.

- First, we would like to minimize the number of times that a user has to enter a password. Suppose each ticket can be used only once. If user C logs on to a workstation in the morning and wishes to check his or her mail at a mail server, C must supply a password to get a ticket for the mail server. If C wishes to check the mail several times during the day, each attempt requires reentering the password. We can improve matters by saying that tickets are reusable. For a single logon session, the workstation can store the mail server ticket after it is received and use it on behalf of the user for multiple accesses to the mail server. But, under this scheme it remains the case that a user would need a new ticket for every different service. If a user wished to access a print server, a mail server, a file server, and so on, the first instance of each access would require a new ticket and hence require the user to enter the password.

- The second problem is that the earlier scenario involved a plaintext transmission of the password [message (1)]. An eavesdropper could capture the password and use any service accessible to the victim.

A More Secure Authentication Dialogue

To solve the problems in the Simple Authentication Dialogue, we introduce a scheme for avoiding plaintext passwords and a new server, known as the ticket-granting server (TGS). The hypothetical scenario is as follows:

Once Per User Logon Session:

(1) $C \rightarrow AS \quad : \quad ID_C \| ID_{tgs}$

(2) $AS \rightarrow C \quad : \quad E_{K_C}[Ticket_{tgs}]$

Once Per Type of Service:

(3) $C \rightarrow TGS \; : \quad ID_C \| ID_V \| Ticket_{tgs}$

(4) $TGS \rightarrow C \; : \quad Ticket_V$

Once Per Service Session

(5) $C \rightarrow V \quad : \quad ID_C \| Ticket_V$

$Ticket_{tgs} = E(K_{tgs}, [ID_C||AD_C||ID_{tgs}||TS_1||Lifetime_1])$

$Ticket_v = E(K_v, [ID_C||AD_C||ID_v||TS_2||Lifetime_2])$

The new service, TGS, issues tickets to users who have been authenticated to AS. Thus, the user first requests a ticket-granting ticket ($Ticket_{tgs}$) from the AS. The client module in the user workstation saves this ticket. Each time the user requires access to a new service, the client applies to the TGS, using the ticket to authenticate itself. The TGS then grants a ticket for the particular service. The client saves each service-granting ticket and uses it to authenticate its user to a server each time a particular service is requested. Let us look at the details of this scheme:

1. The client requests a ticket-granting ticket on behalf of the user by sending its user's ID and password to the AS, together with the TGS ID, indicating a request to use the TGS service.

2. The AS responds with a ticket that is encrypted with a key that is derived from the user's password. When this response arrives at the client, the client prompts the user for his or her password, generates the key, and attempts to decrypt the incoming message. If the correct password is supplied, the ticket is successfully recovered.

   Because only the correct user should know the password, only the correct user can recover the ticket. Thus, we have used the password to obtain credentials from Kerberos without having to transmit the password in plaintext. The ticket itself consists of the ID and network address of the user, and the ID of the TGS. This corresponds to the first scenario. The idea is that the client can use this ticket to request multiple service-granting tickets. So the ticket-granting ticket is to be reusable. However, we do not wish an opponent to be able to capture the ticket and use it. Consider the following scenario: An opponent captures the login ticket and waits until the user has logged off his or her workstation. Then the opponent either gains access to that workstation or configures his workstation with the same network address as that of the victim. The opponent would be able to reuse the ticket to spoof the TGS. To counter this, the ticket includes a timestamp, indicating the date and time at which the ticket was issued, and a lifetime, indicating the length of time for which the ticket is valid (e.g., eight hours). Thus, the client now has a reusable ticket and need not bother the user for a password for each new service request. Finally, note that the ticket-granting ticket is encrypted with a secret key known only to the AS and the TGS. This prevents alteration of the ticket. The ticket is reencrypted with a key based on the user's password. This assures that the ticket can be recovered only by the correct user, providing the authentication.

Now that the client has a ticket-granting ticket, access to any server can be obtained with steps 3 and 4:

3. The client requests a service-granting ticket on behalf of the user. For this purpose, the client transmits a message to the TGS containing the user's ID, the ID of the desired service, and the ticket-granting ticket.

4. The TGS decrypts the incoming ticket and verifies the success of the decryption by the presence of its ID. It checks to make sure that the lifetime has not expired. Then it compares the user ID and network address with the incoming information to authenticate the user. If the user is permitted access to the server V, the TGS issues a ticket to grant access to the requested service.

   The service-granting ticket has the same structure as the ticket-granting ticket. Indeed, because the TGS is a server, we would expect that the same elements are needed to authenticate a client to the TGS and to authenticate a client to an application server. Again, the ticket contains a timestamp and lifetime. If the user wants access to the same service at a later time, the client can simply use the previously acquired service-granting ticket and need not bother the user for a password. Note that the ticket is encrypted with a secret key (K_v) known only to the TGS and the server, preventing alteration.

Finally, with a particular service-granting ticket, the client can gain access to the corresponding service with step 5:

5. The client requests access to a service on behalf of the user. For this purpose, the client transmits a message to the server containing the user's ID and the service-granting ticket. The server authenticates by using the contents of the ticket.

This new scenario satisfies the two requirements of only one password query per user session and protection of the user password.

Disadvantages

Although this scenario enhances security compared to the first attempt, two additional problems remain.

- The heart of the first problem is the lifetime associated with the ticket-granting ticket. If this lifetime is very short (e.g., minutes), then the user will be repeatedly asked for a password. If the lifetime is long (e.g., hours), then an opponent has a greater opportunity for replay. An opponent could eavesdrop on the network and capture a copy of the ticket-granting ticket and then wait for the legitimate user to log out. Then the opponent could forge the legitimate user's network address and send the message of step (3) to the TGS. This would give the opponent unlimited access to the resources and files available to the legitimate user. Similarly, if an opponent captures a service-granting ticket and uses it before it expires, the opponent has

access to the corresponding service. Therefore, a network service (the TGS or an application service) must be able to prove that the person using a ticket is the same person to whom that ticket was issued.

- The second problem is that there may be a requirement for servers to authenticate themselves to users. Without such authentication, an opponent could sabotage the configuration so that messages to a server were directed to another location. The false server would then be in a position to act as a real server and capture any information from the user and deny the true service to the user.

The Version 4 Authentication Dialogue

Table 4.1.1 shows the Kerberos Protocol.

First, consider the problem of captured ticket-granting tickets and the need to determine that the ticket presenter is the same as the client for whom the ticket was issued. The threat is that an opponent will steal the ticket and use it before it expires. To get around this problem, let us have the AS provide both the client and the TGS with a secret piece of information in a secure manner. Then the client can prove its identity to the TGS by revealing the secret information, again in a secure manner. An efficient way of accomplishing this is to use an encryption key as the secure information; this is referred to as a session key in Kerberos. Table 4.1.1a shows the technique for distributing the session key.

The client sends a message to the AS requesting access to the TGS. The AS responds with a message, encrypted with a key derived from the user's password (K_c) that contains the ticket. The encrypted message also contains a copy of the session key, $K_{c,\ tgs}$, where the subscripts indicate that this is a session key for C and TGS. Because this session key is inside the message encrypted with K_c, only the user's client can read it. The same session key is included in the ticket, which can be read only by the TGS. Thus, the session key has been securely delivered to both C and the TGS.

(1) C → AS	$ID_c \| \| ID_{tgs} \| \| TS_1$
(2) AS → C	$E(K_c, [K_{c,tgs} \| \| ID_{tgs} \| \| TS_2 \| \| Lifetime_2 \| \| Ticket_{tgs}])$
	$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| \| ID_c \| \| AD_c \| \| ID_{tgs} \| \| TS_2 \| \| Lifetime_2])$

(a) Authentication Service Exchange to obtain ticket-granting ticket

(3) C → TGS	$ID_v \| \| Ticket_{tgs} \| \| Authenticator_c$
(4) TGS → C	$E(K_{c,tgs}, [K_{c,v} \| \| ID_v \| \| TS_4 \| \| Ticket_v])$
	$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| \| ID_C \| \| AD_C \| \| ID_{tgs} \| \| TS_2 \| \| Lifetime_2])$ $Ticket_v = E(K_v, [K_{c,v} \| \| ID_C \| \| AD_C \| \| ID_v \| \| TS_4 \| \| Lifetime_4])$ $Authenticator_c = E(K_{c,tgs}, [ID_C \| \| AD_C \| \| TS_3])$
(b) Ticket-Granting Service Exchange to obtain service-granting ticket	
(5) C →V	$Ticket_v \| \| Authenticator_c$
(6) V →C	$E(K_{c,v}, [TS_5 + 1])$ (for mutual authentication)
	$Ticket_v = E(K_v, [K_{c,v} \| \| ID_c \| \| AD_c \| \| ID_v \| \| TS_4 \| \| Lifetime_4])$ $Authenticator_c = E(K_{c,v},[ID_c \| \| AD_C \| \| TS_5])$
(c) Client/Server Authentication Exchange to obtain service	

Table 4.1.1 Summary of Kerberos V4 Message Exchanges

Several additional pieces of information have been added to this first phase of the dialogue. Message (1) includes a timestamp, so that the AS knows that the message is timely. Message (2) includes several elements of the ticket in a form accessible to C. This enables C to confirm that this ticket is for the TGS and to learn its expiration time.

Armed with the ticket and the session key, C is ready to approach the TGS. As before, C sends the TGS a message that includes the ticket plus the ID of the requested service (message (3) in Table 4.1.1b). In addition, C transmits an authenticator, which includes the ID and address of C's user and a timestamp. Unlike the ticket, which is reusable, the authenticator is intended for use only once and has a very short lifetime. The TGS can decrypt the ticket with the key that it shares with the AS. This ticket indicates that user C has been provided with the session key $K_{c,tgs}$. In effect, the ticket says, "Anyone who uses $K_{c,tgs}$ must be C." The TGS uses the session key to decrypt the authenticator. The TGS can then check the name and address from the authenticator with that of the

ticket and with the network address of the incoming message. If all match, then the TGS is assured that the sender of the ticket is indeed the ticket's real owner. In effect, the authenticator says, "At time TS_3, I hereby use $K_{c,tgs}$." Note that the ticket does not prove anyone's identity but is a way to distribute keys securely. It is the authenticator that proves the client's identity. Because the authenticator can be used only once and has a short lifetime, the threat of an opponent stealing both the ticket and the authenticator for presentation later is countered.

The reply from the TGS, in message (4), follows the form of message (2). The message is encrypted with the session key shared by the TGS and C and includes a session key to be shared between C and the server V, the ID of V, and the timestamp of the ticket. The ticket itself includes the same session key.

C now has a reusable service-granting ticket for V. When C presents this ticket, as shown in message (5), it also sends an authenticator. The server can decrypt the ticket, recover the session key, and decrypt the authenticator.

If mutual authentication is required, the server can reply as shown in message (6) of Table 4.1.1. The server returns the value of the timestamp from the authenticator, incremented by 1, and encrypted in the session key. C can decrypt this message to recover the incremented timestamp. Because the message was encrypted by the session key, C is assured that it could have been created only by V. The contents of the message assure C that this is not a replay of an old reply.

Finally, at the end of this process, the client and server share a secret key. This key can be used to encrypt future messages between the two or to exchange a new random session key for that purpose.

Fig 4.1.1 provides a simplified overview of the action.

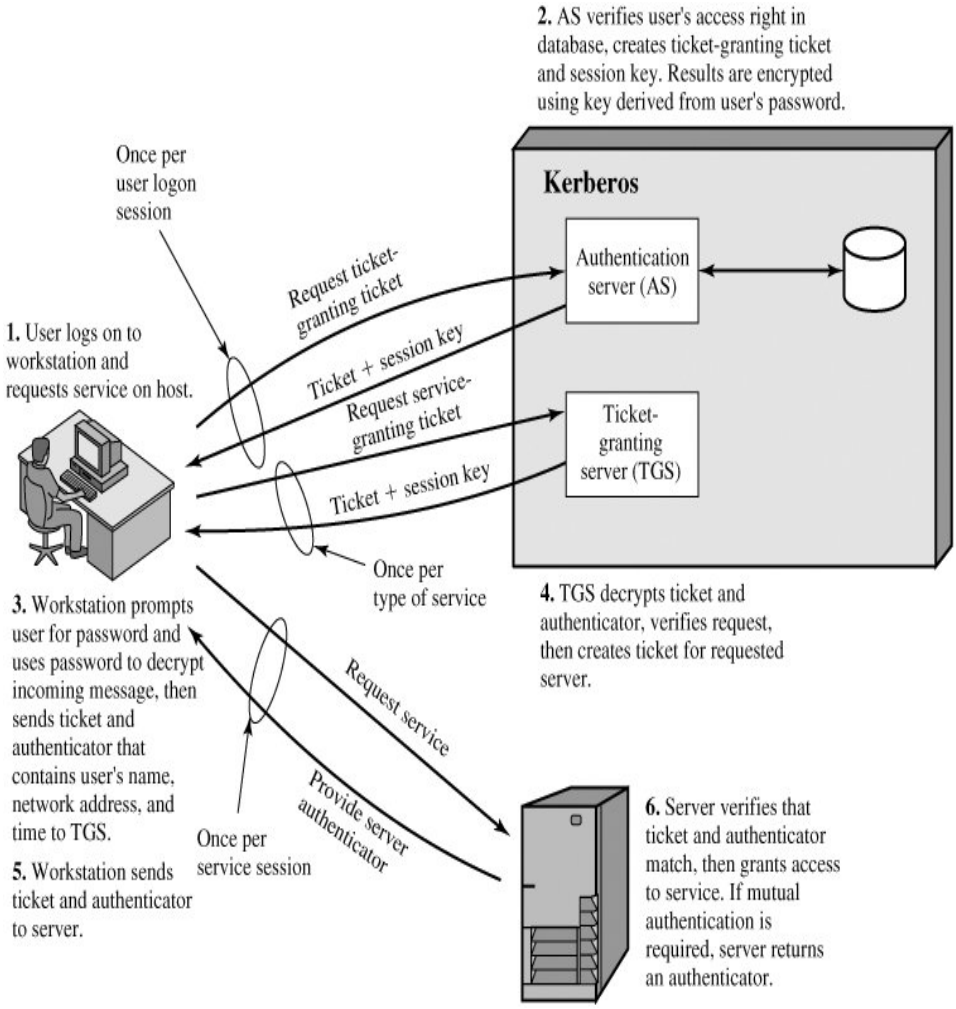


Fig 4.1.1 Overview of Kerberos

Kerberos Realms and Multiple Kerberi

A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers requires the following:

1. The Kerberos server must have the user ID and hashed passwords of all participating users in its database. All users are registered with the Kerberos server.

2. The Kerberos server must share a secret key with each server. All servers are registered with the Kerberos server.

Such an environment is referred to as a **Kerberos realm**. A Kerberos realm is a set of managed nodes that share the same Kerberos database. The Kerberos database resides on the Kerberos master computer system, which should be kept in a physically secure room. A read-only copy of the Kerberos database might also reside on other Kerberos

computer systems. However, all changes to the database must be made on the master computer system. Changing or accessing the contents of a Kerberos database requires the Kerberos master password. A related concept is that of a Kerberos principal, which is a service or user that is known to the Kerberos system. Each Kerberos principal is identified by its principal name. Principal names consist of three parts: *a service or user name, an instance name, and a realm name*

Networks of clients and servers under different administrative organizations typically constitute different realms. That is, to have users and servers in one administrative domain registered with a Kerberos server elsewhere. However, users in one realm may need access to servers in other realms, and some servers may be willing to provide service to users from other realms, provided that those users are authenticated.

Kerberos provides a mechanism for supporting such interrealm authentication. For two realms to support interrealm authentication, a third requirement is added:

3. The Kerberos server in each interoperating realm shares a secret key with the server in the other realm. The two Kerberos servers are registered with each other.

The scheme requires that the Kerberos server in one realm trust the Kerberos server in the other realm to authenticate its users. Furthermore, the participating servers in the second realm must also be willing to trust the Kerberos server in the first realm.

With these ground rules in place, we can describe the mechanism as follows (Fig 4.1.2): A user wishing service on a server in another realm needs a ticket for that server. The user's client follows the usual procedures to gain access to the local TGS and then requests a ticket-granting ticket for a remote TGS (TGS in another realm). The client can then apply to the remote TGS for a service-granting ticket for the desired server in the realm of the remote TGS.

The details of the exchanges illustrated in Fig 4.1.2 are as follows:

(1) $C \rightarrow AS$: $ID_c||ID_{tgs}||TS_1$

(2) $AS \rightarrow C$: $E(K_c, [K_{c,tgs}||ID_{tgs}||TS_2||Lifetime_2||Ticket_{tgs}])$

(3) $C \rightarrow TGS$: $ID_{tgsrem}||Ticket_{tgs}||Authenticator_c$

(4) $TGS \rightarrow C$: $E(K_{c,tgs}, [K_{c,tgsrem}||ID_{tgsrem}||TS_4||Ticket_{tgsrem}])$

(5) $C \rightarrow TGS_{rem}$: $ID_{vrem}||Ticket_{tgsrem}||Authenticator_c$

(1) C → AS : $ID_c||ID_{tgs}||TS_1$

(6) TGS_{rem} →C : $E(K_{c,tgsrem}, [K_{c,vrem}||ID_{vrem}||TS_6||Ticket_{vrem}])$

(7) C → V_{rem} : $Ticket_{vrem}||Authenticator_c$

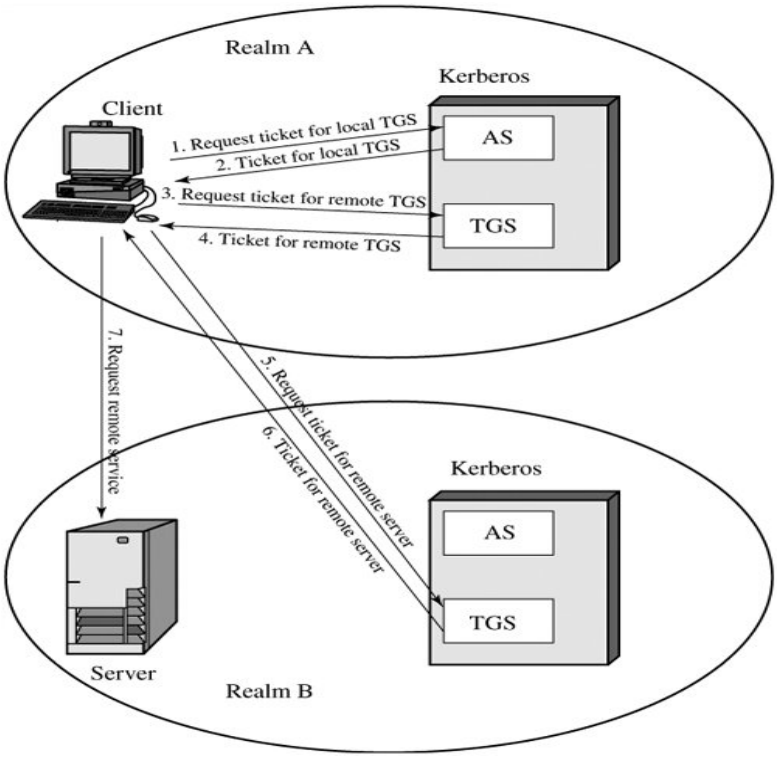


Fig 4.1.2. Request for Service in Another Realm

The ticket presented to the remote server (V_{rem}) indicates the realm in which the user was originally authenticated. The server chooses whether to honor the remote request.

One problem presented by the foregoing approach is that it does not scale well to many realms. If there are N realms, then there must be N(N 1)/2 secure key exchanges so that each Kerberos realm can interoperate with all other Kerberos realms.

Kerberos Version 5

Kerberos Version 5 is specified in RFC 1510 and provides a number of improvements over version 4. Version 5 is intended to address the

limitations of version 4 in two areas: environmental shortcomings and technical deficiencies. Overviews of the changes from version 4 to version 5 are as follows.

Differences between Versions 4 and 5

Kerberos Version 4 was developed for use within the Project Athena environment and, accordingly, did not fully address the need to be of general purpose. This led to the following environmental shortcomings:

1. **Encryption system dependence:** Version 4 requires the use of DES. Export restriction on DES as well as doubts about the strength of DES were thus of concern. In version 5, ciphertext is tagged with an encryption type identifier so that any encryption technique may be used. Encryption keys are tagged with a type and a length, allowing the same key to be used in different algorithms and allowing the specification of different variations on a given algorithm.

2. **Internet protocol dependence:** Version 4 requires the use of Internet Protocol (IP) addresses. Other address types, such as the ISO network address, are not accommodated. Version 5 network addresses are tagged with type and length, allowing any network address type to be used.

3. **Message byte ordering:** In version 4, the sender of a message employs a byte ordering of its own choosing and tags the message to indicate least significant byte in lowest address or most significant byte in lowest address. This techniques works but does not follow established conventions. In version 5, all message structures are defined using Abstract Syntax Notation One (ASN.1) and Basic Encoding Rules (BER), which provide an unambiguous byte ordering.

4. **Ticket lifetime:** Lifetime values in version 4 are encoded in an 8-bit quantity in units of five minutes. Thus, the maximum lifetime that can be expressed is $2^8 \times 5 = 1280$ minutes, or a little over 21 hours. This may be inadequate for some applications (e.g., a long-running simulation that requires valid Kerberos credentials throughout execution). In version 5, tickets include an explicit start time and end time, allowing tickets with arbitrary lifetimes.

5. **Authentication forwarding:** Version 4 does not allow credentials issued to one client to be forwarded to some other host and used by some other client. This capability would enable a client to access a server and have that server access another server on behalf of the client. For example, a client issues a request to a print server that then accesses the client's file from a file server, using the client's credentials for access. Version 5 provides this capability.

6. **Interrealm authentication:** In version 4, interoperability among N realms requires on the order of N^2 Kerberos-to-Kerberos relationships, as described earlier. Version 5 supports a method that requires fewer relationships, as described shortly.

The Version 5 Authentication Dialogue

Table 4.1.3 summarizes the basic version 5 dialogue.

First, consider the authentication service exchange. Message (1) is a client request for a ticket-granting ticket. As before, it includes the ID of the user and the TGS. The following new elements are added:

- **Realm:** Indicates realm of user
- **Options:** Used to request that certain flags be set in the returned ticket
- **Times:** Used by the client to request the following time settings in the ticket:

from: the desired start time for the requested ticket

till: the requested expiration time for the requested ticket

rtime: requested renew-till time

(1) C → AS	$Options\|\|ID_c\|\|Realm_c\|\|ID_{tgs}\|\|Times\|\|Nonce_1$
(2) AS → C	$Realm_c\|\|ID_C\|\|Ticket_{tgs}\|\|E(K_c, [K_{c,tgs}\|\|Times\|\|Nonce_1\|\|Realm_{tgs}\|\|ID_{tgs}])$
	$Ticket_{tgs} = E(K_{tgs}, [Flags\|\|K_{c,tgs}\|\|Realm_c\|\|ID_c\|\|AD_c\|\|Times])$
(a) Authentication Service Exchange to obtain ticket-granting ticket	
(3) C → TGS	$Options\|\|ID_v\|\|Times\|\|\|\|Nonce_2\|\|Ticket_{tgs}\|\|Authenticator_c$
(4) TGS →C	$Realm_c\|\|ID_c\|\|Ticket_v\|\|E(K_{c,tgs}, [K_{c,v}\|\|Times\|\|Nonce_2\|\|Realm_v\|\|ID_v])$ $Ticket_{tgs} = E(K_{tgs}, [Flags\|\|K_{C,tgs}\|\|Realm_c\|\|ID_C\|\|AD_C\|\|Times])$

(1) C → AS	$Options\|\|ID_c\|\|Realm_c\|\|ID_{tgs}\|\|Times\|\|Nonce_1$
	$Ticket_v = E(K_v, [Flags\|\|K_{c,v}\|\|Realm_c\|\|ID_C\|\|AD_c\|\|Times])$ $Authenticator_c = E(K_{c,tgs}, [ID_C\|\|Realm_c\|\|TS_1])$
(b) Ticket-Granting Service Exchange to obtain service-granting ticket	
(5) C → V	$Options\|\|Ticket_v\|\|Authenticator_c$
(6) V → C	$E_{Kc,v}$ $[TS_2\|\|Subkey\|\|Seq\#]$ $Ticket_v = E(K_v, [Flags\|\|K_{c,v}\|\|Realm_c\|\|ID_C\|\|AD_C\|\|Times])$ $Authenticator_c = E(K_{c,v},[ID_C\|\|Realm_c\|\|TS_2\|\|Subkey\|\|Seq\#])$
(c) Client/Server Authentication Exchange to obtain service	

Table 4.1.3 Summary of Kerberos Version 5 Message Exchanges

- **Nonce**: A random value to be repeated in message (2) to assure that the response is fresh and has not been replayed by an opponent

Message (2) returns a ticket-granting ticket, identifying information for the client, and a block encrypted using the encryption key based on the user's password. This block includes the session key to be used between the client and the TGS, times specified in message (1), the nonce from message (1), and TGS identifying information. The ticket itself includes the session key, identifying information for the client, the requested time values, and flags that reflect the status of this ticket and the requested options. These flags introduce significant new functionality to version 5.

Let us now compare the ticket-granting service exchange for versions 4 and 5. We see that message (3) for both versions include an authenticator, a ticket, and the name of the requested service. In addition, version 5 includes requested times and options for the ticket and a nonce, all with functions similar to those of message (1). The authenticator itself is essentially the same as the one used in version 4.

Message (4) has the same structure as message (2), returning a ticket plus information needed by the client, the latter encrypted with the session key now shared by the client and the TGS.

Finally, for the client/server authentication exchange, several new features appear in version 5. In message (5), the client may request as

an option that mutual authentication is required. The authenticator includes several new fields as follows:

- **Subkey**: The client's choice for an encryption key to be used to protect this specific application session. If this field is omitted, the session key from the ticket ($K_{c,v}$) is used.

- **Sequence number**: An optional field that specifies the starting sequence number to be used by the server for messages sent to the client during this session. Messages may be sequence numbered to detect replays.

If mutual authentication is required, the server responds with message (6). This message includes the timestamp from the authenticator. Note that in version 4, the timestamp was incremented by one. This is not necessary in version 5 because the nature of the format of messages is such that it is not possible for an opponent to create message (6) without knowledge of the appropriate encryption keys. The subkey field, if present, overrides the subkey field, if present, in message (5). The optional sequence number field specifies the starting sequence number to be used by the client.

Ticket Flags

The flags field included in tickets in version 5 supports expanded functionality compared to that available in version 4. Table 4.1.4 summarizes the flags that may be included in a ticket.

INITIAL	This ticket was issued using the AS protocol and not issued based on a ticket-granting ticket.
PRE-AUTHENT	During initial authentication, the client was authenticated by the KDC before a ticket was issued.
HW-AUTHENT	The protocol employed for initial authentication required the use of hardware expected to be possessed solely by the named client.
RENEWABLE	Tells TGS that this ticket can be used to obtain a replacement ticket that expires at a later date.
MAY-POSTDATE	Tells TGS that a postdated ticket may be issued based on this ticket-granting ticket.
POSTDATED	Indicates that this ticket has been postdated; the end server can check the authtime field to see when the original authentication occurred.
INVALID	This ticket is invalid and must be validated by the KDC before use.

INITIAL	This ticket was issued using the AS protocol and not issued based on a ticket-granting ticket.
PROXIABLE	Tells TGS that a new service-granting ticket with a different network address may be issued based on the presented ticket.
PROXY	Indicates that this ticket is a proxy.
FORWARDABLE	Tells TGS that a new ticket-granting ticket with a different network address may be issued based on this ticket-granting ticket.
FORWARDED	Indicates that this ticket has either been forwarded or was issued based on authentication involving a forwarded ticket-granting ticket.

Table 4.1.4 Kerberos Version 5 Flags

The INITIAL flag indicates that this ticket was issued by the AS, not by the TGS. When a client requests a service-granting ticket from the TGS, it presents a ticket-granting ticket obtained from the AS. In version 4, this was the only way to obtain a service-granting ticket. Version 5 provides the additional capability that the client can get a service-granting ticket directly from the AS. The utility of this is as follows: A server, such as a password-changing server, may wish to know that the client's password was recently tested.

The PRE-AUTHENT flag, if set, indicates that when the AS received the initial request [message (1)], it authenticated the client before issuing a ticket. The exact form of this preauthentication is left unspecified. As an example, the MIT implementation of version 5 has encrypted timestamp preauthentication, enabled by default. When a user wants to get a ticket, it has to send to the AS a preauthentication block containing a random confounder, a version number, and a timestamp, encrypted in the client's password-based key. The AS decrypts the block and will not send a ticket-granting ticket back unless the timestamp in the preauthentication block is within the allowable time skew (time interval to account for clock drift and network delays). Another possibility is the use of a smart card that generates continually changing passwords that are included in the preauthenticated messages. The passwords generated by the card can be based on a user's password but be transformed by the card so that, in effect, arbitrary passwords are used. This prevents an attack based on easily guessed passwords. If a smart card or similar device was used, this is indicated by the HW-AUTHENT flag.

When a ticket has a long lifetime, there is the potential for it to be stolen and used by an opponent for a considerable period. If a short lifetime is used to lessen the threat, then overhead is involved in acquiring new tickets. In the case of a ticket-granting ticket, the client would either have to store the user's secret key, which is clearly risky, or repeatedly ask the

user for a password. A compromise scheme is the use of renewable tickets. A ticket with the RENEWABLE flag set includes two expiration times: one for this specific ticket and one that is the latest permissible value for an expiration time. A client can have the ticket renewed by presenting it to the TGS with a requested new expiration time. If the new time is within the limit of the latest permissible value, the TGS can issue a new ticket with a new session time and a later specific expiration time. The advantage of this mechanism is that the TGS may refuse to renew a ticket reported as stolen.

A client may request that the AS provide a ticket-granting ticket with the MAY-POSTDATE flag set. The client can then use this ticket to request a ticket that is flagged as POSTDATED and INVALID from the TGS. Subsequently, the client may submit the postdated ticket for validation. This scheme can be useful for running a long batch job on a server that requires a ticket periodically. The client can obtain a number of tickets for this session at once, with spread-out time values. All but the first ticket are initially invalid. When the execution reaches a point in time when a new ticket is required, the client can get the appropriate ticket validated. With this approach, the client does not have to repeatedly use its ticket-granting ticket to obtain a service-granting ticket.

In version 5, it is possible for a server to act as a proxy on behalf of a client, in effect adopting the credentials and privileges of the client to request a service from another server. If a client wishes to use this mechanism, it requests a ticket-granting ticket with the PROXIABLE flag set. When this ticket is presented to the TGS, the TGS is permitted to issue a service-granting ticket with a different network address; this latter ticket will have its PROXY flag set. An application receiving such a ticket may accept it or require additional authentication to provide an audit trail.

The proxy concept is a limited case of the more powerful forwarding procedure. If a ticket is set with the FORWARDABLE flag, a TGS can issue to the requestor a ticket-granting ticket with a different network address and the FORWARDED flag set. This ticket can then be presented to a remote TGS. This capability allows a client to gain access to a server on another realm without requiring that each Kerberos maintain a secret key with Kerberos servers in every other realm. For example, realms could be structured hierarchically. Then a client could walk up the tree to a common node and then back down to reach a target realm. Each step of the walk would involve forwarding a ticket-granting ticket to the next TGS in the path.

REFERENCES

1. William Stallings, Cryptography and Network Security, PHI Publishers
2. www.wikipedia.org

Review Questions:

1. Explain the Kerberos and write note on Kerberos V4 authentication dialogue
2. Explain the operational details of Kerberos Realm.

2. DIRECTORY AUTHENTICATION SERVICES

OBJECTIVE

In this lesson we examine the X.509 directory authentication service. This standard is important as part of the directory service that it supports, but is also a basic building block used in other standards, such as S/MIME

DIRECTORY SERVICES

In software engineering, a directory is similar to a dictionary. It enables the look up of a name and information associated with that name. As a word in a dictionary may have multiple definitions, in a directory, a name may be associated with multiple, different, pieces of information. Likewise, as a word may have different parts and different definitions, a name in a directory may have many different types of data. Based on this rudimentary explanation of a directory, a **directory service** is simply the software system that stores, organizes and provides access to information in a directory.

Directory services were part of an Open Systems Interconnection (OSI) initiative to get everyone in the industry to agree to common network standards to provide multi-vendor interoperability. In the 1980s the ITU and ISO came up with a set of standards - X.500, for directory services, initially to support the requirements of inter-carrier electronic messaging and network name lookup. The Lightweight Directory Access Protocol, LDAP, is based on the directory information services of X.500, but uses the TCP/IP stack and a string encoding scheme of the X.500 protocol DAP, giving it more relevance on the Internet. X.509 is an ITU-T standard for a Public Key Infrastructure (PKI) for single sign-on and Privilege Management Infrastructure (PMI). X.509 specifies, amongst other things, standard formats for public key certificates, certificate revocation lists, attribute certificates, and a certification path validation algorithm.

X.509 Authentication Service

X.509 defines a framework for the provision of authentication services by the X.500 directory to its users. Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority. In addition, X.509 defines alternative authentication protocols based on the use of public-key certificates.

X.509 is an important standard because the certificate structure and authentication protocols defined in X.509 are used in a variety of contexts. For example, the X.509 certificate format is used in S/MIME, IP Security, and SSL/TLS and SET.

X.509 is based on the use of public-key cryptography and digital signatures. The standard does not dictate the use of a specific algorithm but recommends RSA. The digital signature scheme is assumed to require the use of a hash function. Again, the standard does not dictate a specific hash algorithm. Fig 4.2.1 illustrates the generation of a public-key certificate.

Certificates

The heart of the X.509 scheme is the public-key certificate associated with each user. These user certificates are assumed to be created by some trusted certification authority (CA) and placed in the directory by the CA or by the user. The directory server itself is not responsible for the creation of public keys or for the certification function; it merely provides an easily accessible location for users to obtain certificates.

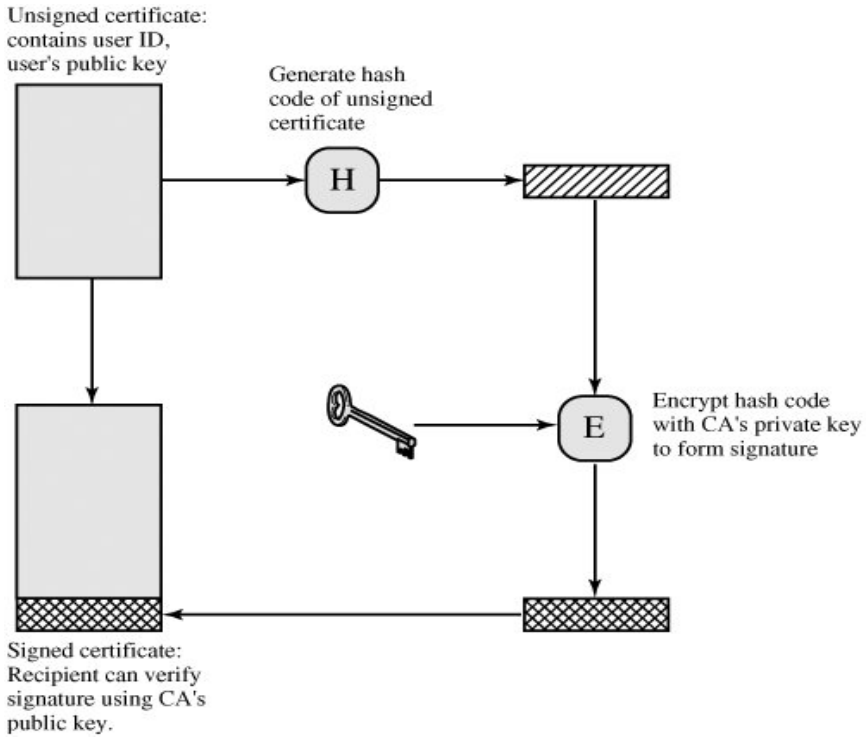


Fig 4.2.1 public-Key Certificate use

Figure 4.2.2a shows the general format of a certificate, which includes the following elements:

- **Version:** Differentiates among successive versions of the certificate format; the default is version 1. If the Issuer Unique

Identifier or Subject Unique Identifier are present, the value must be version 2. If one or more extensions are present, the version must be version 3.

- **Serial number:** An integer value, unique within the issuing CA, that is unambiguously associated with this certificate.

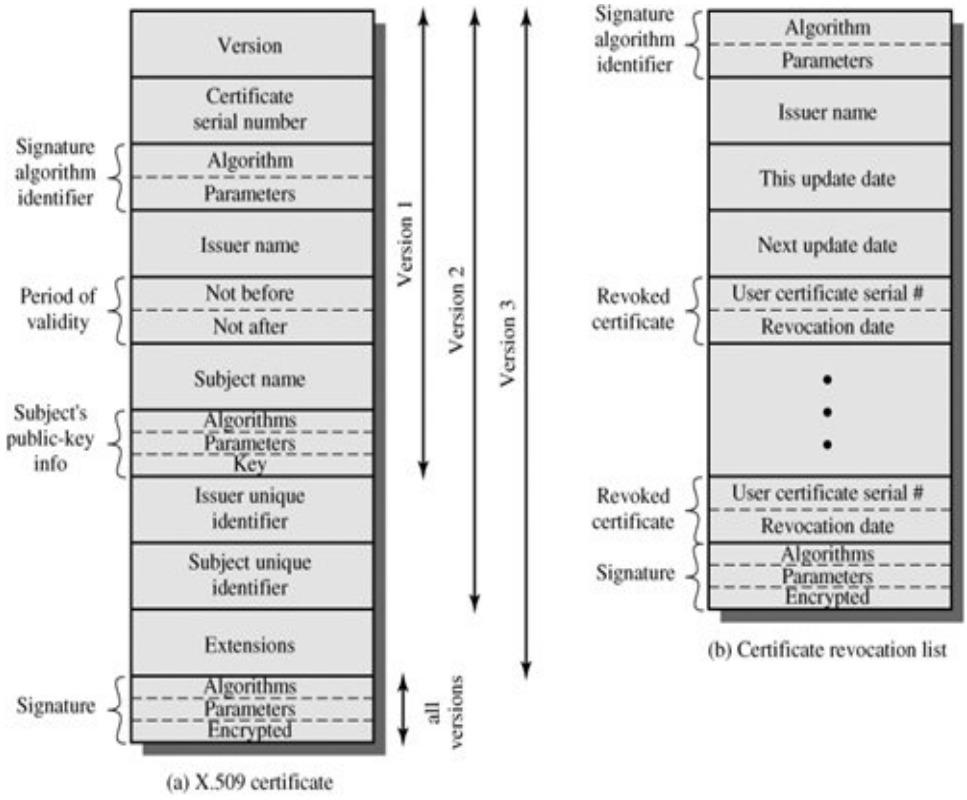


Fig 4.2.2 X.509 Formats

- **Signature algorithm identifier**: The algorithm used to sign the certificate, together with any associated parameters. Because this information is repeated in the Signature field at the end of the certificate, this field has little, if any, utility.

- **Issuer name:** X.500 name of the CA that created and signed this certificate.

- **Period of validity:** Consists of two dates: the first and last on which the certificate is valid.

- **Subject name:** The name of the user to whom this certificate refers. That is, this certificate certifies the public key of the subject who holds the corresponding private key.

- **Subject's public-key information:** The public key of the subject, plus an identifier of the algorithm for which this key is to be used, together with any associated parameters.

- **Issuer unique identifier:** An optional bit string field used to identify uniquely the issuing CA in the event the X.500 name has been reused for different entities.

- **Subject unique identifier:** An optional bit string field used to identify uniquely the subject in the event the X.500 name has been reused for different entities.

- **Extensions:** A set of one or more extension fields. Extensions were added in version 3 and are discussed later in this section.

- **Signature:** Covers all of the other fields of the certificate; it contains the hash code of the other fields, encrypted with the CA's private key. This field includes the signature algorithm identifier.

The unique identifier fields were added in version 2 to handle the possible reuse of subject and/or issuer names over time. These fields are rarely used.

The standard uses the following notation to define a certificate:

$$CA\langle\langle A\rangle\rangle = CA \{V, SN, AI, CA, T_A, A, Ap\}$$

where,

Y = the certificate of user X issued by certification authority
<<X>> Y

Y {I} = the signing of I by Y. It consists of I with an encrypted hash code appended

The CA signs the certificate with its private key. If the corresponding public key is known to a user, then that user can verify that a certificate signed by the CA is valid.

Obtaining a User's Certificate

User certificates generated by a CA have the following characteristics:

- Any user with access to the public key of the CA can verify the user public key that was certified.

- No party other than the certification authority can modify the certificate without this being detected.

Because certificates are unforgeable, they can be placed in a directory without the need for the directory to make special efforts to protect them.

If all users subscribe to the same CA, then there is a common trust of that CA. All user certificates can be placed in the directory for access by all users. In addition, a user can transmit his or her certificate directly to other users. In either case, once B is in possession of A's certificate, B has confidence that messages it encrypts with A's public key will be secure from eavesdropping and that messages signed with A's private key are unforgeable.

If there is a large community of users, it may not be practical for all users to subscribe to the same CA. Because it is the CA that signs certificates, each participating user must have a copy of the CA's own public key to verify signatures. This public key must be provided to each user in an absolutely secure (with respect to integrity and authenticity) way so that the user has confidence in the associated certificates. Thus, with many users, it may be more practical for there to be a number of CAs, each of which securely provides its public key to some fraction of the users.

Now suppose that A has obtained a certificate from certification authority X_1 and B has obtained a certificate from CA X_2. If A does not securely know the public key of X_2, then B's certificate, issued by X_2, is useless to A. A can read B's certificate, but A cannot verify the signature. However, if the two CAs have securely exchanged their own public keys, the following procedure will enable A to obtain B's public key:

1. A obtains, from the directory, the certificate of X_2 signed by X_1. Because A securely knows X_1's public key, A can obtain X_2's public key from its certificate and verify it by means of X_1's signature on the certificate.

2. A then goes back to the directory and obtains the certificate of B signed by X_2 Because A now has a trusted copy of X_2's public key, A can verify the signature and securely obtain B's public key.

A has used a chain of certificates to obtain B's public key. In the notation of X.509, this chain is expressed as

$$X_1\langle\langle X_2\rangle\rangle\ X_2\ \langle\langle B\rangle\rangle$$

In the same fashion, B can obtain A's public key with the reverse chain:

$$X_2\langle\langle X_1\rangle\rangle\ X_1\ \langle\langle A\rangle\rangle$$

This scheme need not be limited to a chain of two certificates. An arbitrarily long path of CAs can be followed to produce a chain. A chain with N elements would be expressed as

$$X_1\langle\langle X_2\rangle\rangle\ X_2\ \langle\langle X_3\rangle\rangle \ldots\ X_N\langle\langle B\rangle\rangle$$

In this case, each pair of CAs in the chain (X_i, X_{i+1}) must have created certificates for each other.

All these certificates of CAs by CAs need to appear in the directory, and the user needs to know how they are linked to follow a path to another user's public-key certificate. X.509 suggests that CAs be arranged in a hierarchy so that navigation is straightforward.

Figure 4.2.3 is an example of X.509 hierarchy. The connected circles indicate the hierarchical relationship among the CAs; the associated boxes indicate certificates maintained in the directory for each CA entry. The directory entry for each CA includes two types of certificates:

- **Forward certificates:** Certificates of X generated by other CAs
- **Reverse certificates:** Certificates generated by X that are the certificates of other CAs
- In this example, user A can acquire the following certificates from the directory to establish a certification path to B:
    - **X<<W>> W <<V>> V <<Y>> <<Z>> Z <<B>>**

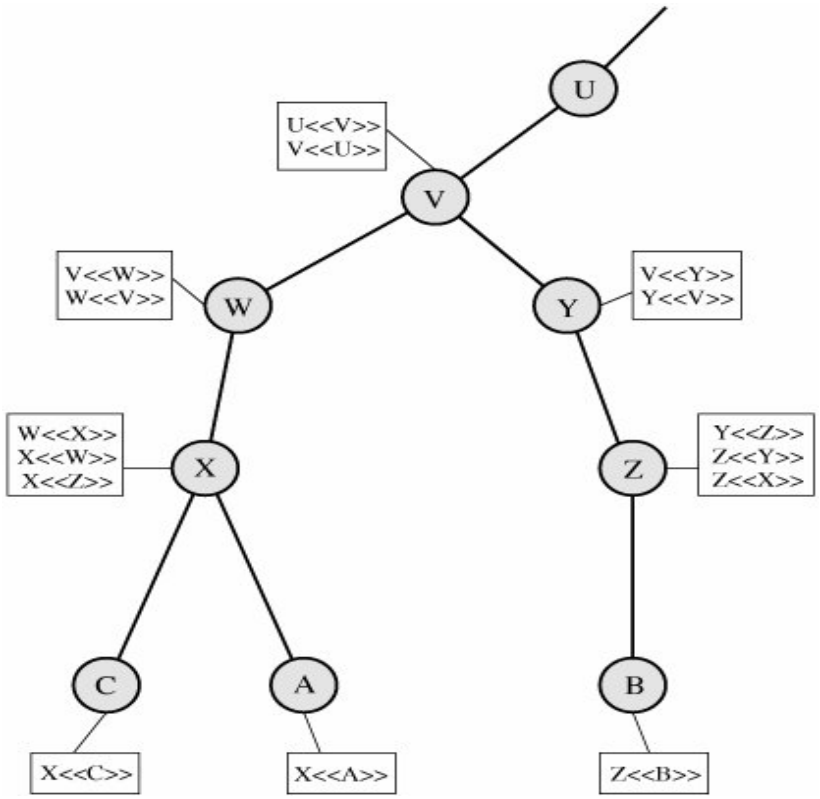


Fig 4.2.3 X.509 Hierarchy: A Hypothetical Example

When A has obtained these certificates, it can unwrap the certification path in sequence to recover a trusted copy of B's public key. Using this public key, A can send encrypted messages to B. If A wishes to receive encrypted messages back from B, or to sign messages sent to B, then B will require A's public key, which can be obtained from the following certification path:

Z<<Y>> Y <<V>> V <<W>> W <<X>>X <<A>>

B can obtain this set of certificates from the directory, or A can provide them as part of its initial message to B.

Revocation of Certificates

We know that each certificate includes a period of validity, much like a credit card. Typically, a new certificate is issued just before the expiration of the old one. In addition, it may be desirable on occasion to revoke a certificate before it expires, for one of the following reasons:

1. The user's private key is assumed to be compromised.
2. The user is no longer certified by this CA.
3. The CA's certificate is assumed to be compromised.

Each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA, including both those issued to users and to other CAs. These lists should also be posted on the directory.

Each certificate revocation list (CRL) posted to the directory is signed by the issuer and includes (Fig 4.2.2b) the issuer's name, the date the list was created, the date the next CRL is scheduled to be issued, and an entry for each revoked certificate. Each entry consists of the serial number of a certificate and revocation date for that certificate. Because serial numbers are unique within a CA, the serial number is sufficient to identify the certificate.

When a user receives a certificate in a message, the user must determine whether the certificate has been revoked. The user could check the directory each time a certificate is received. To avoid the delays (and possible costs) associated with directory searches, it is likely that the user would maintain a local cache of certificates and lists of revoked certificates.

Authentication Procedures

X.509 also includes three alternative authentication procedures that are intended for use across a variety of applications. All these procedures make use of public-key signatures. It is assumed that the two parties know each other's public key, either by obtaining each other's certificates

from the directory or because the certificate is included in the initial message from each side. Fig 4.2.4 illustrates the three procedures.

One-Way Authentication

One way authentication involves a single transfer of information from one user (A) to another (B), and establishes the following:

1. The identity of A and that the message was generated by A
2. That the message was intended for B
3. The integrity and originality (it has not been sent multiple times) of the message

Note that only the identity of the initiating entity is verified in this process, not that of the responding entity. At a minimum, the message includes a timestamp t_A, a nonce r_A and the identity of B and is signed with A's private key. The timestamp consists of an optional generation time and an expiration time. This prevents delayed delivery of messages. The nonce can be used to detect replay attacks. The nonce value must be unique within the expiration time of the message. Thus, B can store the nonce until it expires and reject any new messages with the same nonce.

For pure authentication, the message is used simply to present credentials to B. The message may also include information to be conveyed. This information, sgnData, is included within the scope of the signature, guaranteeing its authenticity and integrity. The message may also be used to convey a session key to B, encrypted with B's public key.

Two-Way Authentication

In addition to the three elements listed above, two-way authentication establishes the following elements:

4. The identity of B and that the reply message was generated by B
5. That the message was intended for A
6. The integrity and originality of the reply

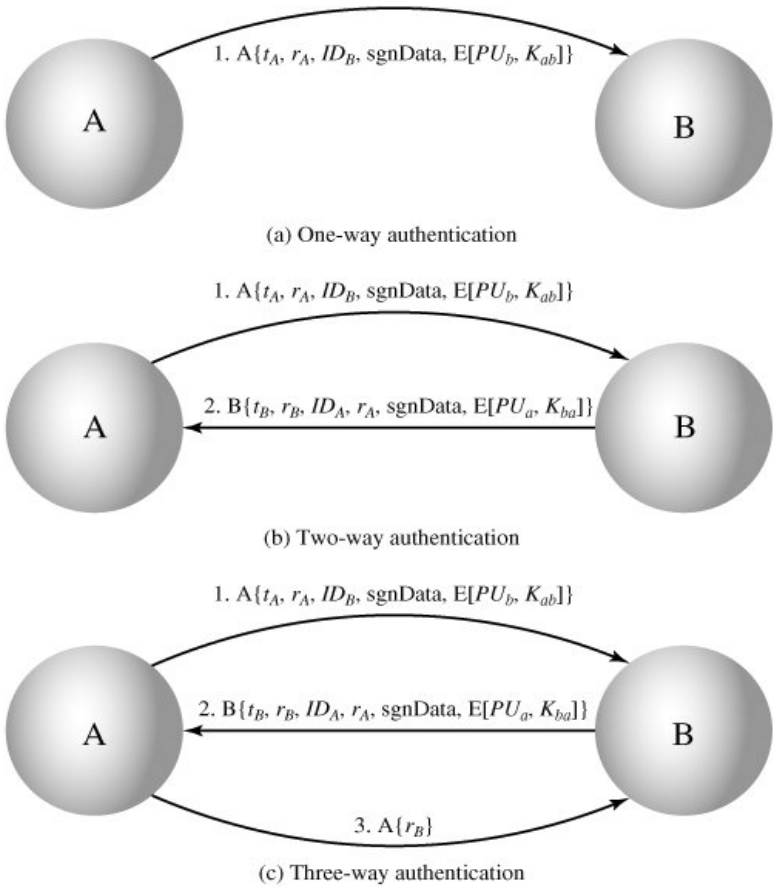


Fig 4.2.4 X.509 Strong Authentication Procedures

Two-way authentication thus permits both parties in a communication to verify the identity of the other. The reply message includes the nonce from A, to validate the reply. It also includes a timestamp and nonce generated by B. As before, the message may include signed additional information and a session key encrypted with A's public key.

Three-Way Authentication

In three-way authentication, a final message from A to B is included, which contains a signed copy of the nonce r_B. The intent of this design is that timestamps need not be checked: Because both nonces are echoed back by the other side, each side can check the returned nonce to detect replay attacks. This approach is needed when synchronized clocks are not available.

Drawbacks of X.509 Version 2

The X.509 version 2 format does not convey all of the information that recent design and implementation experience has shown to be needed.

The following requirements are not satisfied by version 2:

1. The Subject field is inadequate to convey the identity of a key owner to a public-key user. X.509 names may be relatively short and lacking in obvious identification details that may be needed by the user.

2. The Subject field is also inadequate for many applications, which typically recognize entities by an Internet e-mail address, a URL, or some other Internet-related identification.

3. There is a need to indicate security policy information. This enables a security application or function, such as IPSec, to relate an X.509 certificate to a given policy.

4. There is a need to limit the damage that can result from a faulty or malicious CA by setting constraints on the applicability of a particular certificate.

5. It is important to be able to identify different keys used by the same owner at different times. This feature supports key life cycle management, in particular the ability to update key pairs for users and CAs on a regular basis or under exceptional circumstances.

X.509 Version 3

Version 3 includes a number of optional extensions that may be added to the version 2 format. Each extension consists of an extension identifier, a criticality indicator, and an extension value. The criticality indicator indicates whether an extension can be safely ignored. If the indicator has a value of TRUE and an implementation does not recognize the extension, it must treat the certificate as invalid.

The certificate extensions fall into three main categories: key and policy information, subject and issuer attributes, and certification path constraints.

Key and Policy Information

These extensions convey additional information about the subject and issuer keys, plus indicators of certificate policy. A certificate policy is a named set of rules that indicates the applicability of a certificate to a particular community and/or class of application with common security requirements. For example, a policy might be applicable to the authentication of electronic data interchange (EDI) transactions for the trading of goods within a given price range.

This area includes the following:

- **Authority key identifier:** Identifies the public key to be used to verify the signature on this certificate or CRL. Enables distinct keys of the same CA to be differentiated. One use of this field is to handle CA key pair updating.

- **Subject key identifier:** Identifies the public key being certified. Useful for subject key pair updating. Also, a subject may have multiple key pairs and, correspondingly, different certificates for different purposes (e.g., digital signature and encryption key agreement).

- **Key usage:** Indicates a restriction imposed as to the purposes for which, and the policies under which, the certified public key may be used. May indicate one or more of the following: digital signature, nonrepudiation, key encryption, data encryption, key agreement, CA signature verification on certificates, CA signature verification on CRLs.

- **Private-key usage period:** Indicates the period of use of the private key corresponding to the public key. Typically, the private key is used over a different period from the validity of the public key. For example, with digital signature keys, the usage period for the signing private key is typically shorter than that for the verifying public key.

- **Certificate policies:** Certificates may be used in environments where multiple policies apply. This extension lists policies that the certificate is recognized as supporting, together with optional qualifier information.

- **Policy mappings:** Used only in certificates for CAs issued by other CAs. Policy mappings allow an issuing CA to indicate that one or more of that issuer's policies can be considered equivalent to another policy used in the subject CA's domain.

Certificate Subject and Issuer Attributes

These extensions support alternative names, in alternative formats, for a certificate subject or certificate issuer and can convey additional information about the certificate subject, to increase a certificate user's confidence that the certificate subject is a particular person or entity. For example, information such as postal address, position within a corporation, or picture image may be required.

The extension fields in this area include the following:

- **Subject alternative name:** Contains one or more alternative names, using any of a variety of forms. This field is important for supporting certain applications, such as electronic mail, EDI, and IPSec, which may employ their own name forms.

- **Issuer alternative name:** Contains one or more alternative names, using any of a variety of forms.

- **Subject directory attributes:** Conveys any desired X.500 directory attribute values for the subject of this certificate.

Certification Path Constraints

These extensions allow constraint specifications to be included in certificates issued for CAs by other CAs. The constraints may restrict the types of certificates that can be issued by the subject CA or that may occur subsequently in a certification chain.

The extension fields in this area include the following:

- **Basic constraints:** Indicates if the subject may act as a CA. If so, a certification path length constraint may be specified.
- **Name constraints:** Indicates a name space within which all subject names in subsequent certificates in a certification path must be located.
- **Policy constraints:** Specifies constraints that may require explicit certificate policy identification or inhibit policy mapping for the remainder of the certification path.

REFERENCES

1. William Stallings, Cryptography and Network Security, PHI Publishers
2. www.wikipedia.org

Review Questions:

1. Write a short note on X.509 authentication service.
2. Explain about X.509 authentication format in detail.
3. Write a short note on different authentication procedures.

3. ELECTRONIC MAIL SECURITY

OBJECTIVE

This lesson focuses on one of the most heavily used distributed application, the electronic mail. There is increasing interest in providing authentication and confidentiality services as part of an electronic mail facility. This lesson looks at the two approaches likely to dominate electronic mail security in the near future. Pretty Good Privacy (PGP) is a widely used scheme that does not depend on any organization or authority. Thus, it is as well suited to individual, personal use as it is to incorporation in network configurations operated by organizations. S/MIME (Secure/Multipurpose Internet Mail Extension) was developed specifically to be an Internet Standard.

INTRODUCTION

In virtually all distributed environments, electronic mail is the most heavily used network-based application. It is also the only distributed application that is widely used across all architectures and vendor platforms. Users expect to be able to, and do, send mail to others who are connected directly or indirectly to the Internet, regardless of host operating system or communications suite.

Pretty Good Privacy

PGP is a remarkable phenomenon. Largely the effort of Phil Zimmermann, PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications. In essence, Zimmermann has done the following:

1. Selected the best available cryptographic algorithms as building blocks

2. Integrated these algorithms into a general-purpose application that is independent of operating system and processor and that is based on a small set of easy-to-use commands

3. Made the package and its documentation, including the source code, freely available via the Internet, bulletin boards, and commercial networks such as AOL (America On Line)

4. Entered into an agreement with a company (Viacrypt, now Network Associates) to provide a fully compatible, low-cost commercial version of PGP

PGP has grown explosively due to number of reasons:

1. It is available free worldwide in versions that run on a variety of platforms, including Windows, UNIX, Macintosh, and many more. In addition, the commercial version satisfies users who want a product that comes with vendor support.

2. It is based on algorithms that have survived extensive public review and are considered extremely secure. Specifically, the package includes RSA, DSS, and Diffie-Hellman for public-key encryption; CAST-128, IDEA, and 3DES for symmetric encryption; and SHA-1 for hash coding.

3. It has a wide range of applicability, from corporations that wish to select and enforce a standardized scheme for encrypting files and messages to individuals who wish to communicate securely with others worldwide over the Internet and other networks.

4. It was not developed by, nor is it controlled by, any governmental or standards organization. For those with an instinctive distrust of "the establishment," this makes PGP attractive.

5. PGP is now on an Internet standards track (RFC 3156We begin with an overall look at the operation of PGP. Next, we examine how cryptographic keys are created and stored. Then, we address the vital issue of public key management.

Notation

Most of the notation used in this chapter has been used before, but a few terms are new. It is perhaps best to summarize those at the beginning. The following symbols are used:

$\mathbf{K_S}$ = Session Key used in symmetric encryption scheme

$\mathbf{PR_s}$ = Private key of users A, used in public-key encryption scheme

$\mathbf{PU_s}$ = Public Key of user A, used in public-key encryption scheme

EP = Public Key Encryption

DP = Public key Decryption

EC = Symmetric Encryption

DC = Symmetric Decryption

H = Hash Function

|| = Concatenation

Z = Compression using ZIP algorithm

R64 = Conversion to Radix 64 ASCII format

The PGP documentation often uses the term *secret key* to refer to a key paired with a public key in a public-key encryption scheme.

Operational Description

The actual operation of PGP, as opposed to the management of keys, consists of five services: authentication, confidentiality, compression, e-mail compatibility, and segmentation (Table 4.3.1). Let us examine each of these.

Authentication

Figure 4.3.1a illustrates the digital signature service provided by PGP. The sequence is as follows:

1. The sender creates a message.
2. SHA-1 is used to generate a 160-bit hash code of the message.
3. The hash code is encrypted with RSA using the sender's private key, and the result is prepended to the message.
4. The receiver uses RSA with the sender's public key to decrypt and recover the hash code.
5. The receiver generates a new hash code for the message and compares it with the decrypted hash code. If the two match, the message is accepted as authentic.
6.

Function	**Algorithms**	**Used Description**
Digital signature	DSS/SHA or RSA/SHA	A hash code of a message is created using SHA-1. This message digest is encrypted using DSS or RSA with the sender's private key and included with the message.
Message encryption	CAST or IDEA or Three-key Triple DES with Diffie-Hellman or RSA	A message is encrypted using CAST-128 or IDEA or 3DES with a one-time session key generated by the sender. The session key is encrypted using Diffie-Hellman or RSA with the recipient's public key and included with the message.
Compression	ZIP	A message may be compressed, for storage or transmission, using ZIP.

Function	Algorithms	Used Description
Email compatibility	Radix 64 conversion	To provide transparency for email applications, an encrypted message may be converted to an ASCII string using radix 64 conversion.
Segmentation		To accommodate maximum message size limitations, PGP performs segmentation and reassembly.

Table 4.3.1 Summary of PGP Services

The combination of SHA-1 and RSA provides an effective digital signature scheme. Because of the strength of RSA, the recipient is assured that only the possessor of the matching private key can generate the signature. Because of the strength of SHA-1, the recipient is assured that no one else could generate a new message that matches the hash code and, hence, the signature of the original message. As an alternative, signatures can be generated using DSS/SHA-1.

Confidentiality

Another basic service provided by PGP is confidentiality, which is provided by encrypting messages to be transmitted or to be stored locally as files. In both cases, the symmetric encryption algorithm CAST-128 may be used. Alternatively, IDEA or 3DES may be used. The 64-bit cipher feedback (CFB) mode is used.

In PGP, each symmetric key is used only once. That is, a new key is generated as a random 128-bit number for each message. Thus, although this is referred to in the documentation as a session key, it is in reality a one-time key. Because it is to be used only once, the session key is bound to the message and transmitted with it. To protect the key, it is encrypted with the receiver's public key. Figure 4.7b illustrates the sequence, which can be described as follows:

1. The sender generates a message and a random 128-bit number to be used as a session key for this message only.

2. The message is encrypted, using CAST-128 (or IDEA or 3DES) with the session key.

3. The session key is encrypted with RSA, using the recipient's public key, and is prepended to the message.

4. The receiver uses RSA with its private key to decrypt and recover the session key.

5. The session key is used to decrypt the message.

As an alternative to the use of RSA for key encryption, PGP provides an option referred to as Diffie-Hellman. PGP uses a variant of Diffie-Hellman that does provide encryption/decryption, known as ElGamal.

Observations

- First, to reduce encryption time the combination of symmetric and public-key encryption is used in preference to simply using RSA or ElGamal to encrypt the message directly: CAST-128 and the other symmetric algorithms are substantially faster than RSA or ElGamal.

- Second, the use of the public-key algorithm solves the session key distribution problem, because only the recipient is able to recover the session key that is bound to the message. Note that we do not need a session key exchange protocol, because we are not beginning an ongoing session. Rather, each message is a one-time independent event with its own key. Furthermore, given the store-and-forward nature of electronic mail, the use of handshaking to assure that both sides have the same session key is not practical.

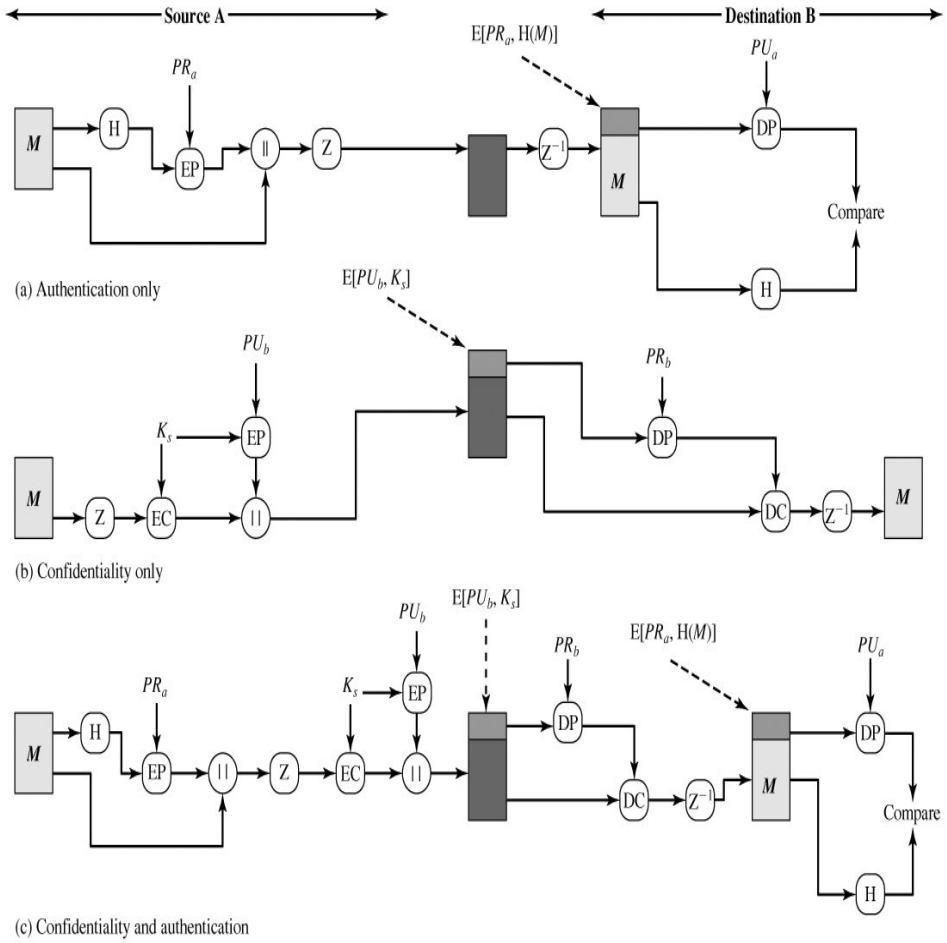


Fig 4.3.1PGP Cryptographic Functions

- Finally, the use of one-time symmetric keys strengthens what is already a strong symmetric encryption approach. Only a small amount of plaintext is encrypted with each key, and there is no relationship among the keys. Thus, to the extent that the public-key algorithm is secure, the entire scheme is secure. To this end, PGP provides the user with a range of key size options from 768 to 3072 bits (the DSS key for a signature is limited to 1024 bits).

Confidentiality and Authentication

As Figure 4.3.1c illustrates, both services may be used for the same message. First, a signature is generated for the plaintext message and prepended to the message. Then the plaintext message plus signature is encrypted using CAST-128 (or IDEA or 3DES), and the session key is encrypted using RSA (or ElGamal). This sequence is preferable to the opposite: encrypting the message and then generating a signature for the encrypted message. It is generally more convenient to store a signature with a plaintext version of a message.

In essence, when both services are used, the sender first signs the message with its own private key, then encrypts the message with a session key, and then encrypts the session key with the recipient's public key.

Compression

As a default, PGP compresses the message after applying the signature but before encryption. This has the benefit of saving space both for e-mail transmission and for file storage.

The placement of the compression algorithm, indicated by Z for compression and Z^{-1} for decompression in Figure 4.3.1, is critical:

1. The signature is generated before compression for two reasons:

    a. It is preferable to sign an uncompressed message so that one can store only the uncompressed message together with the signature for future verification. If one signed a compressed document, then it would be necessary either to store a compressed version of the message for later verification or to recompress the message when verification is required.

    b. Even if one were willing to generate dynamically a recompressed message for verification, PGP's compression algorithm presents a difficulty. The algorithm is not deterministic; various implementations of the algorithm achieve different tradeoffs in running speed versus compression ratio and, as a result, produce different compressed forms. However, these different compression algorithms are interoperable because any version of the algorithm can correctly decompress the output of any other version. Applying the hash function and signature after compression would constrain all PGP implementations to the same version of the compression algorithm.

2. Message encryption is applied after compression to strengthen cryptographic security. Because the compressed message has less redundancy than the original plaintext, cryptanalysis is more difficult.

Cryptographic Keys and Key Rings

PGP makes use of four types of keys: one-time session symmetric keys, public keys, private keys, and passphrase-based symmetric keys (explained subsequently). Three separate requirements can be identified with respect to these keys:

1. A means of generating unpredictable session keys is needed.

2. We would like to allow a user to have multiple public-key/private-key pairs. One reason is that the user may wish to change his or her key pair from time to time. When this happens, any messages in the pipeline will be constructed with an obsolete key. Furthermore, recipients will know only the old public key until an update reaches them. In addition to the need to change keys over time, a user may wish to have multiple key pairs at a given time to interact with different groups of correspondents or simply to enhance security by limiting the amount of material encrypted with any one key. The upshot of all this is that there is not a one-to-one correspondence between users and their public keys. Thus, some means is needed for identifying particular keys.

3. Each PGP entity must maintain a file of its own public/private key pairs as well as a file of public keys of correspondents.

Figure 4.3.2 shows the format of a transmitted PGP message. A message consists of three components: the message component, a signature (optional), and a session key component (optional).

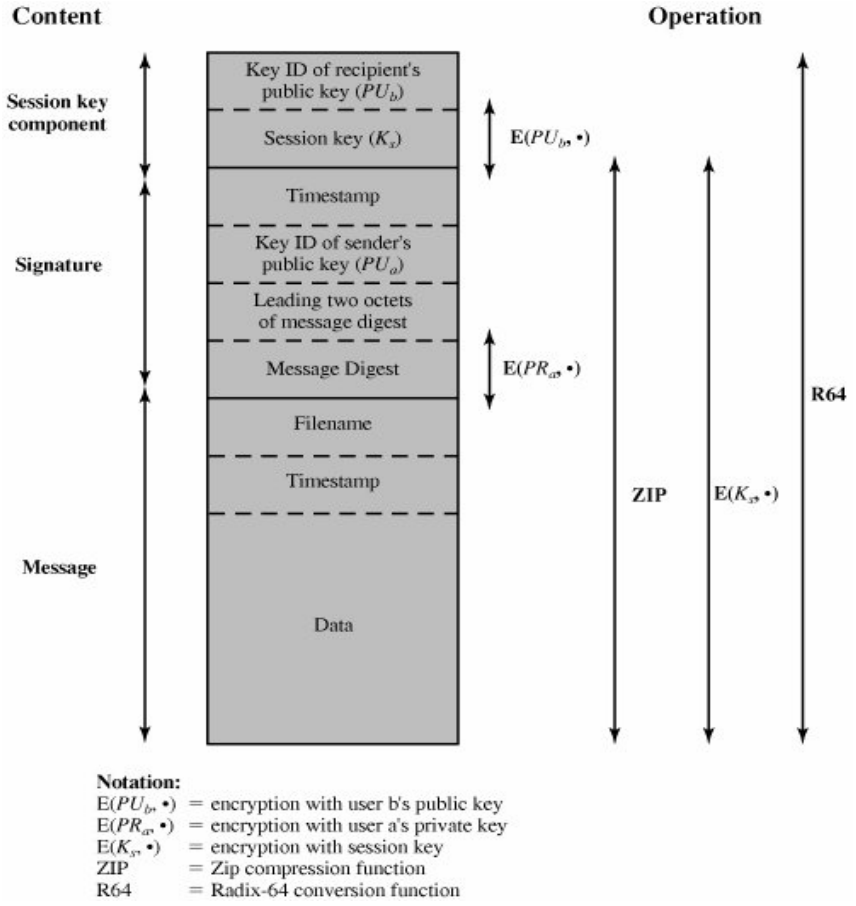


Fig 4.3.2 General Format of PGP Message (from A to B)

The message component includes the actual data to be stored or transmitted, a filename and a timestamp that specifies the time of creation.

The signature component includes the following:

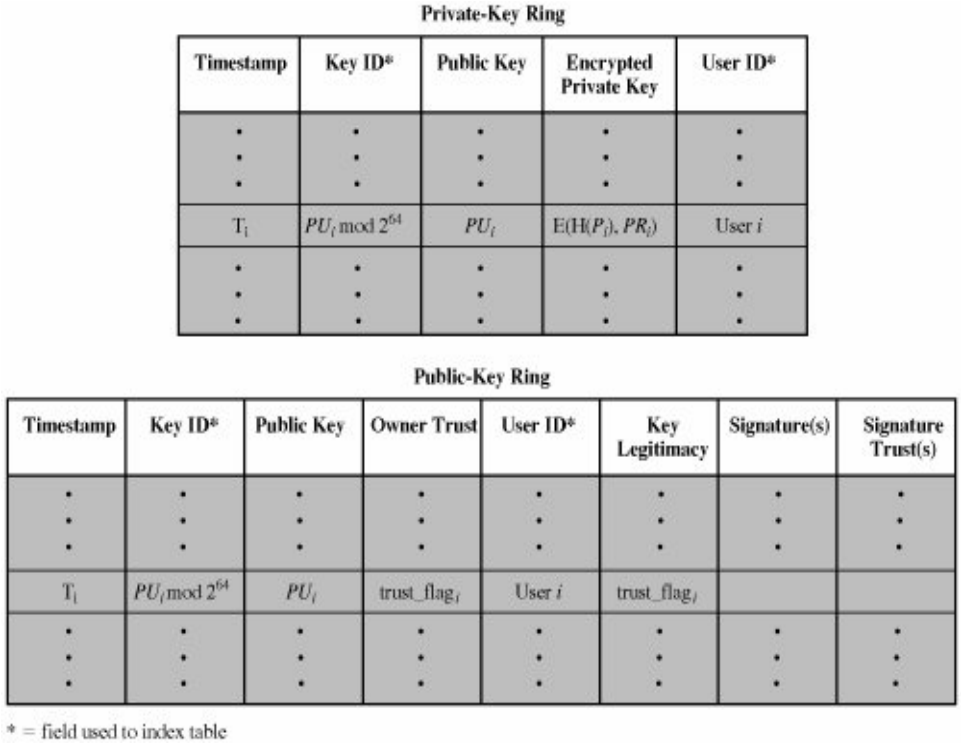
- **Timestamp:** The time at which the signature was made.
- **Message digest:** The 160-bit SHA-1 digest, encrypted with the sender's private signature key. The digest is calculated over the signature timestamp concatenated with the data portion of the message component. The inclusion of the signature timestamp in the digest assures against replay types of attacks. The exclusion of the filename and timestamp portions of the message component ensures that detached signatures are exactly the same as attached signatures prefixed to the message. Detached signatures are calculated on a separate file that has none of the message component header fields.
- **Leading two octets of message digest:** To enable the recipient to determine if the correct public key was used to decrypt the message digest for authentication, by comparing this plaintext copy of the first two octets with the first two octets of the decrypted digest. These octets also serve as a 16-bit frame check sequence for the message.
- **Key ID of sender's public key:** Identifies the public key that should be used to decrypt the message digest and, hence, identifies the private key that was used to encrypt the message digest.

The message component and optional signature component may be compressed using ZIP and may be encrypted using a session key. The session key component includes the session key and the identifier of the recipient's public key that was used by the sender to encrypt the session key. The entire block is usually encoded with radix-64 encoding.

Key Rings

The key IDs are critical to the operation of PGP and two key IDs are included in any PGP message that provides both confidentiality and authentication. These keys need to be stored and organized in a systematic way for efficient and effective use by all parties. The scheme used in PGP is to provide a pair of data structures at each node, one to store the public/private key pairs owned by that node and one to store the public keys of other users known at this node. These data structures are referred to, respectively, as the private-key ring and the public-key ring.

Figure 4.3.3 shows the general structure of a private-key ring. We can view the ring as a table, in which each row represents one of the public/private key pairs owned by this user.



Private-Key Ring

Timestamp	Key ID*	Public Key	Encrypted Private Key	User ID*
. . .	. . .	. . .	. . .	. . .
T_i	$PU_i \bmod 2^{64}$	PU_i	$E(H(P_i), PR_i)$	User i
. . .	. . .	. . .	. . .	. . .

Public-Key Ring

Timestamp	Key ID*	Public Key	Owner Trust	User ID*	Key Legitimacy	Signature(s)	Signature Trust(s)
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
T_i	$PU_i \bmod 2^{64}$	PU_i	$trust\_flag_i$	User i	$trust\_flag_i$		
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

* = field used to index table

Fig 4.3.3 General Structure of Private- and Public-Key Rings

Figure 4.3.3 also shows the general structure of a public-key ring. This data structure is used to store public keys of other users that are known to this user. For the moment, let us ignore some fields shown in the table and describe the following fields:

- **Timestamp:** The date/time when this entry was generated.
- **Key ID:** The least significant 64 bits of the public key for this entry.
- **Public Key:** The public key for this entry.
- **User ID:** Identifies the owner of this key. Multiple user IDs may be associated with a single public key.

The public-key ring can be indexed by either User ID or Key ID

PGP Message Transmission

Consider message transmission (Figure 4.3.4) and assume that the message is to be both signed and encrypted. The sending PGP entity performs the following steps:

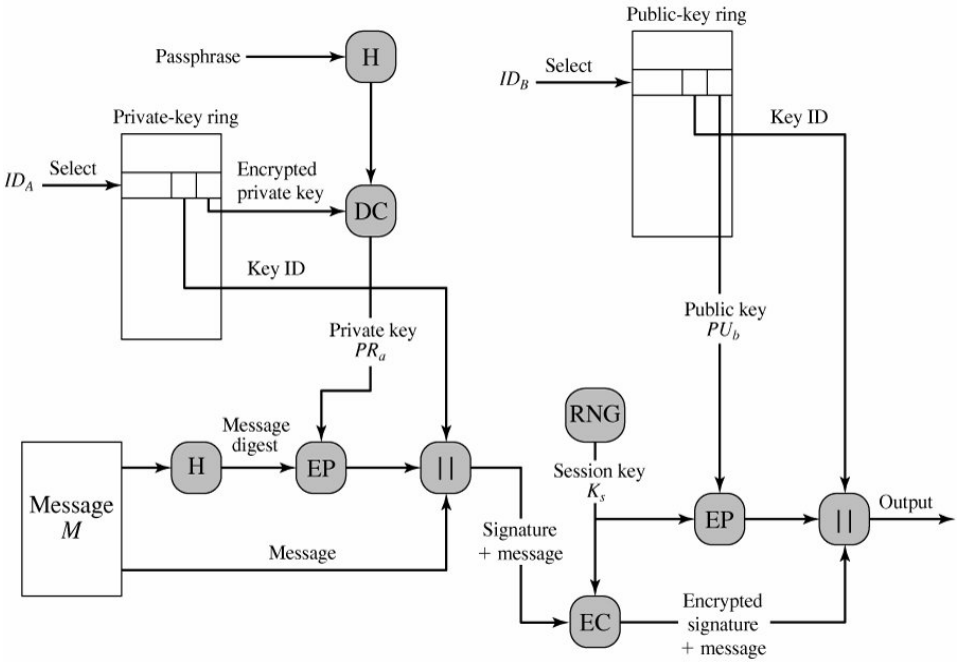


Fig 4.3.4 PGP Message Generation (from User A to User B; no compression or radix 64 conversion)

1. Signing the message

a. PGP retrieves the sender's private key from the private-key ring using your_userid as an index. If your_userid was not provided in the command, the first private key on the ring is retrieved.

b. PGP prompts the user for the passphrase to recover the unencrypted private key.

c. The signature component of the message is constructed.

2. Encrypting the message

a. PGP generates a session key and encrypts the message.

b. PGP retrieves the recipient's public key from the public-key ring using her_userid as an index.

c. The session key component of the message is constructed.

PGP Message Transmission

The receiving PGP entity performs the following steps (Figure 4.3.5):

1. Decrypting the message

PGP retrieves the receiver's private key from the private-key ring, using the Key ID field in the session key component of the message as an index.

a. PGP prompts the user for the passphrase to recover the unencrypted private key.

b. PGP then recovers the session key and decrypts the message.

2. Authenticating the message

a. PGP retrieves the sender's public key from the public-key ring, using the Key ID field in the signature key component of the message as an index.

b. PGP recovers the transmitted message digest.

c. PGP computes the message digest for the received message and compares it to the transmitted message digest to authenticate.

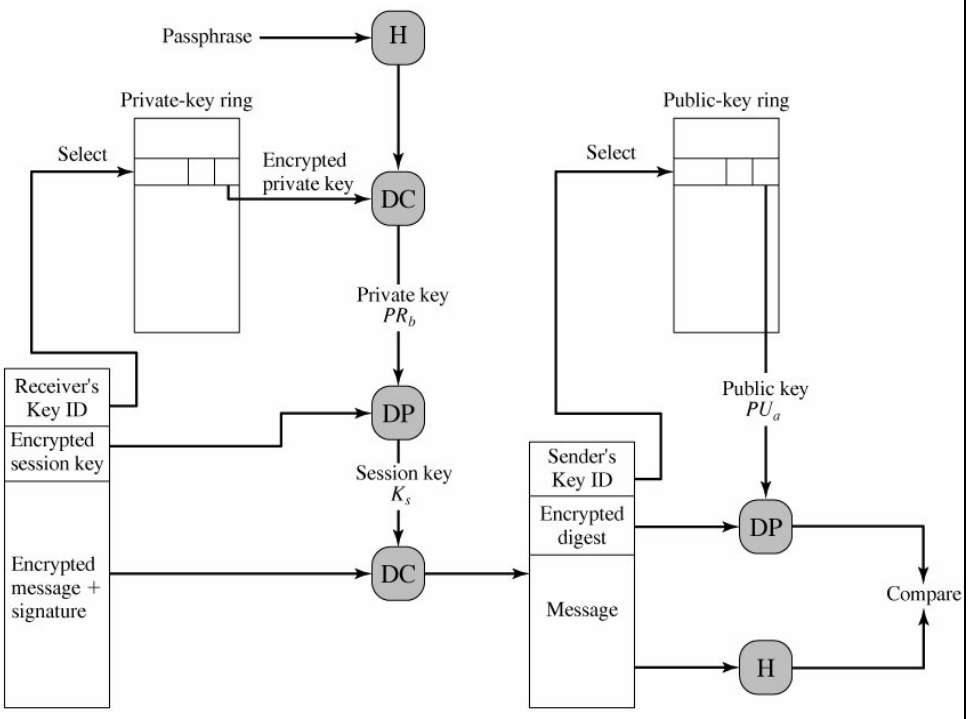


Fig 4.3.5 PGP Message Reception (from User A to User B; no compression or radix 64 conversion)

S/MIME

S/MIME (Secure / Multipurpose Internet Mail Extensions) is a set of specifications for securing

electronic mail. S/MIME is based upon the widely used MIME standard [MIME] and describes a

protocol for adding cryptographic security services through MIME encapsulation of digitally

signed and encrypted objects.

S/MIME provides the following cryptographic security services for electronic messaging applications:

- Authentication
- Message integrity
- Non-repudiation of origin (using digital signatures)
- Privacy and data security (using encryption)

S/MIME relies on four fundamental technologies to format and protect electronic mail messages.

These fundamental technologies are cryptographic algorithms, public key infrastructure (PKI),

the cryptographic message syntax (CMS) data format, and MIME. Correct implementation of

these mechanisms is essential to the security and interoperability of every S/MIME client. While

these technologies will not be tested in isolation, they will be tested indirectly.

S/MIME Functionality

In terms of general functionality, S/MIME is very similar to PGP. Both offer the ability to sign and/or encrypt messages.

Functions

S/MIME provides the following functions:

- **Enveloped data:** This consists of encrypted content of any type and encrypted-content encryption keys for one or more recipients.

- **Signed data:** A digital signature is formed by taking the message digest of the content to be signed and then encrypting that with the private key of the signer. The content plus signature are then encoded using base64 encoding. A signed data message can only be viewed by a recipient with S/MIME capability.

- **Clear-signed data:** As with signed data, a digital signature of the content is formed. However, in this case, only the digital signature is encoded using base64. As a result, recipients without S/MIME capability can view the message content, although they cannot verify the signature.

- **Signed and enveloped data:** Signed-only and encrypted-only entities may be nested, so that encrypted data may be signed and signed data or clear-signed data may be encrypted.

Cryptographic Algorithms

Table 4.3.2 summarizes the cryptographic algorithms used in S/MIME. S/MIME uses the following terminology, taken from RFC 2119 to specify the requirement level:

- **Must:** The definition is an absolute requirement of the specification. An implementation must include this feature or function to be in conformance with the specification.

- **Should:** There may exist valid reasons in particular circumstances to ignore this feature or function, but it is recommended that an implementation include the feature or function.

S/MIME incorporates three public-key algorithms. The Digital Signature Standard (DSS) is the preferred algorithm for digital signature. S/MIME lists Diffie-Hellman as the preferred algorithm for encrypting session keys. S/MIME uses a variant of Diffie-Hellman that does provide encryption/decryption, known as ElGamal. As an alternative, RSAcan be used for both signatures and session key encryption.

Function	Requirement
Create a message digest to be used in forming a digital signature. Encrypt message digest to form digital signature.	MUST support SHA-1. Receiver SHOULD support MD5 for backward compatibility. Sending and receiving agents MUST support DSS. Sending agents SHOULD support RSA encryption.

Function	Requirement
	Receiving agents SHOULD support verification of RSA signatures with key sizes 512 bits to 1024 bits.
Encrypt session key for transmission with message.	Sending and receiving agents SHOULD support Diffie-Hellman. Sending and receiving agents MUST support RSA encryption with key sizes 512 bits to 1024 bits.
Encrypt message for transmission with one-time session key.	Sending and receiving agents MUST support encryption with triple DES Sending agents SHOULD support encryption with AES. Sending agents SHOULD support encryption with RC2/40.
Create a message authentication code	Receiving agents MUST support HMAC with SHA-1. Receiving agents SHOULD support HMAC with SHA-1.

Table 4.3.2 Cryptographic Algorithms Used in S/MIME

For the hash function used to create the digital signature, the specification requires the 160-bit SHA-1 but recommends receiver support for the 128-bit MD5 for backward compatibility with older versions of S/MIME. For message encryption, three-key triple DES (tripleDES) is recommended. In essence, a sending agent has two decisions to make. First, the sending agent must determine if the receiving agent is capable of decrypting using a given encryption algorithm. Second, if the receiving agent is only capable of accepting weakly encrypted content, the sending agent must decide if it is acceptable to send using weak encryption. To support this decision process, a sending agent may announce its decrypting capabilities in order of preference any message that it sends out. A receiving agent may store that information for future use. The following rules, in the following order, should be followed by a sending agent:

1. If the sending agent has a list of preferred decrypting capabilities from an intended recipient, it SHOULD choose the first (highest preference) capability on the list that it is capable of using.

2. If the sending agent has no such list of capabilities from an intended recipient but has received one or more messages from the recipient, then the outgoing message SHOULD use the same encryption algorithm as was used on the last signed and encrypted message received from that intended recipient.

3. If the sending agent has no knowledge about the decryption capabilities of the intended recipient and is willing to risk that the recipient may not be able to decrypt the message, then the sending agent SHOULD use tripleDES.

4. If the sending agent has no knowledge about the decryption capabilities of the intended recipient and is not willing to risk that the recipient may not be able to decrypt the message, then the sending agent MUST use RC2/40.

If a message is to be sent to multiple recipients and a common encryption algorithm cannot be selected for all, then the sending agent will need to send two messages. However, in that case, it is important to note that the security of the message is made vulnerable by the transmission of one copy with lower security.

S/MIME Messages

S/MIME makes use of a number of new MIME content types, which are shown in Table 4.3.3. All of the new application types use the designation PKCS. This refers to a set of public-key cryptography specifications issued by RSA Laboratories and made available for the S/MIME effort.

Type	**Subtype**	**smime Parameter**	**Description**
Multipart	Signed		A clear-signed message in two parts: one is the message and the other is the signature.
Application	Pkcs 7-mime	signedData	A signed S/MIME entity.
	Pkcs 7-mime	envelopedData	An encrypted S/MIME entity.
	Pkcs 7-mime	degenerate signedData	An entity containing only public- key certificates.

Type	Subtype	smime Parameter	Description
	Pkcs 7-mime	CompressedData	A compressed S/MIME entity
	Pkcs7-signature	signedData	The content type of the signature subpart of a multipart/signed message.

Table 4.3.3 S/MIME Content Types

Securing a MIME Entity

S/MIME secures a MIME entity with a signature, encryption, or both. A MIME entity may be an entire message, or if the MIME content type is multipart, then a MIME entity is one or more of the subparts of the message. The MIME entity is prepared according to the normal rules for MIME message preparation. Then the MIME entity plus some security-related data, such as algorithm identifiers and certificates, are processed by S/MIME to produce what is known as a PKCS object. A PKCS object is then treated as message content and wrapped in MIME (provided with appropriate MIME headers).

S/MIME content types

EnvelopedData

The steps for preparing an envelopedData MIME entity are as follows:

1. Generate a pseudorandom session key for a particular symmetric encryption algorithm (RC2/40 or tripleDES).

2. For each recipient, encrypt the session key with the recipient's public RSA key.

3. For each recipient, prepare a block known as RecipientInfo that contains an identifier of the recipient's public-key certificate an identifier of the algorithm used to encrypt the session key, and the encrypted session key.

4. Encrypt the message content with the session key.

The RecipientInfo blocks followed by the encrypted content constitute the envelopedData. This information is then encoded into base64. A sample message is the following:

```
Content-Type:      application/pkcs7-mime;      smime-
type=enveloped-

       data; name=smime.p7m

Content-Transfer-Encoding: base64

Content-Disposition: attachment; filename=smime.p7m

rfvbnj75.6tbBghyHhHUujhJhjH77n8HHGT9HG4VQpfyF467GhIGfH
fYT6

7n8HHGghyHhHUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tb
B9H

f8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpf
yF4

0GhIGfHfQbnj756YT64V
```

To recover the encrypted message, the recipient first strips off the base64 encoding. Then the recipient's private key is used to recover the session key. Finally, the message content is decrypted with the session key.

SignedData

The signedData smime-type can actually be used with one or more signers. For clarity, we confine our description to the case of a single digital signature. The steps for preparing a signedData MIME entity are as follows:

1. Select a message digest algorithm (SHA or MD5).

2. Compute the message digest, or hash function, of the content to be signed.

3. Encrypt the message digest with the signer's private key.

4. Prepare a block known as SignerInfo that contains the signer's public-key certificate, an identifier of the message digest algorithm, an identifier of the algorithm used to encrypt the message digest, and the encrypted message digest.

The signedData entity consists of a series of blocks, including a message digest algorithm identifier, the message being signed, and SignerInfo. The signedData entity may also include a set of public-key certificates sufficient to constitute a chain from a recognized root or top-level certification authority to the signer. This information is then encoded into base64. A sample message is the following:

```
Content-Type:      application/pkcs7-mime;      smime-
type=signed-data;

        name=smime.p7m

Content-Transfer-Encoding: base64

Content-Disposition: attachment; filename=smime.p7m

567GhIGfHfYT6ghyHhHUujpfyF4f8HHGTrfvhJhjH776tbB9HG4VQb
nj7

77n8HHGT9HG4VQpfyF467GhIGfHfYT6rfvbnj756tbBghyHhHUujhJ
hjH

HUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H7n8HHGgh
yHh

6YT64V0GhIGfHfQbnj75
```

To recover the signed message and verify the signature, the recipient first strips off the base64 encoding. Then the signer's public key is used to decrypt the message digest. The recipient independently computes the message digest and compares it to the decrypted message digest to verify the signature.

Clear Signing

Clear signing is achieved using the multipart content type with a signed subtype. As was mentioned, this signing process does not involve transforming the message to be signed, so that the message is sent "in the clear." Thus, recipients with MIME capability but not S/MIME capability are able to read the incoming message.

A multipart/signed message has two parts. The first part can be any MIME type but must be prepared so that it will not be altered during transfer from source to destination. This means that if the first part is not 7bit, then it needs to be encoded using base64 or quoted-printable. Then this part is processed in the same manner as signedData, but in this case an object with signedData format is created that has an empty message content field. This object is a detached signature. It is then transfer encoded using base64 to become the second part of the multipart/signed message. This second part has a MIME content type of application and a subtype of pkcs7-signature. Here is a sample message:

```
Content-Type: multipart/signed;

   protocol="application/pkcs7-signature";

   micalg=sha1; boundary=boundary42

boundary42
```

```
Content-Type: text/plain

This is a clear-signed message.

boundary42

Content-Type:              application/pkcs7-signature;
name=smime.p7s

Content-Transfer-Encoding: base64

Content-Disposition: attachment; filename=smime.p7s

ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHf
YT6

4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfv
bnj

n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpf
yF4

7GhIGfHfYT64VQbnj756

boundary42
```

The protocol parameter indicates that this is a two-part clear-signed entity. The micalg parameter indicates the type of message digest used. The receiver can verify the signature by taking the message digest of the first part and comparing this to the message digest recovered from the signature in the second part.

Registration Request

Typically, an application or user will apply to a certification authority for a public-key certificate. The application/pkcs10 S/MIME entity is used to transfer a certification request. The certification request includes certificationRequestInfo block, followed by an identifier of the public-key encryption algorithm, followed by the signature of the certificationRequestInfo block, made using the sender's private key. The certificationRequestInfo block includes a name of the certificate subject (the entity whose public key is to be certified) and a bit-string representation of the user's public key.

Certificates-Only Message

A message containing only certificates or a certificate revocation list (CRL) can be sent in response to a registration request. The message is an application/pkcs7-mime type/subtype with an smime-type parameter of degenerate. The steps involved are the same as those for creating a signedData message, except that there is no message content and the signerInfo field is empty.

Enhanced Security Services

As of this writing, three enhanced security services have been proposed in an Internet draft. The details of these may change, and additional services may be added. The three services are as follows:

- **Signed receipts:** A signed receipt may be requested in a SignedData object. Returning a signed receipt provides proof of delivery to the originator of a message and allows the originator to demonstrate to a third party that the recipient received the message. In essence, the recipient signs the entire original message plus original (sender's) signature and appends the new signature to form a new S/MIME message.

- **Security labels:** A security label may be included in the authenticated attributes of a SignedData object. A security label is a set of security information regarding the sensitivity of the content that is protected by S/MIME encapsulation. The labels may be used for access control, by indicating which users are permitted access to an object. Other uses include priority (secret, confidential, restricted, and so on) or role based, describing which kind of people can see the information (e.g., patient's health-care team, medical billing agents, etc.).

- **Secure mailing lists:** When a user sends a message to multiple recipients, a certain amount of per-recipient processing is required, including the use of each recipient's public key. The user can be relieved of this work by employing the services of an S/MIME Mail List Agent (MLA). An MLA can take a single incoming message, perform the recipient-specific encryption for each recipient, and forward the message. The originator of a message need only send the message to the MLA, with encryption performed using the MLA's public key.

IP Security

Internet Protocol Security (IPsec) is a suite of protocols for securing Internet Protocol (IP) communications by authenticating and/or encrypting each IP packet in a data stream. IPsec also includes protocols for cryptographic key establishment.

IPsec protocols operate at the network layer, layer 3 of the OSI model. Other Internet security protocols in widespread use, such as SSL, TLS and SSH, operate from the transport layer up (OSI layers 4 - 7). This makes IPsec more flexible, as it can be used for protecting layer 4 protocols, including both TCP and UDP, the most commonly used transport layer protocols. IPsec has an advantage over SSL and other methods that operate at higher layers: an application doesn't need to be

designed to use IPsec, whereas the ability to use SSL or another higher-layer protocol must be incorporated into the design of an application.

IPsec is a framework of open standards that provides data confidentiality, data integrity, and data authentication between participating peers. IPsec provides these security services at the IP layer; it uses IKE to handle negotiation of protocols and algorithms based on local policy and to generate the encryption and authentication keys to be used by IPsec. IPSec can be used to protect one or more data flows between a pair of hosts, between a pair of security gateways, or between a security gateway and a host. The official term of "IPsec" as defined by the IETF is often being wrongly written as "IPSec".

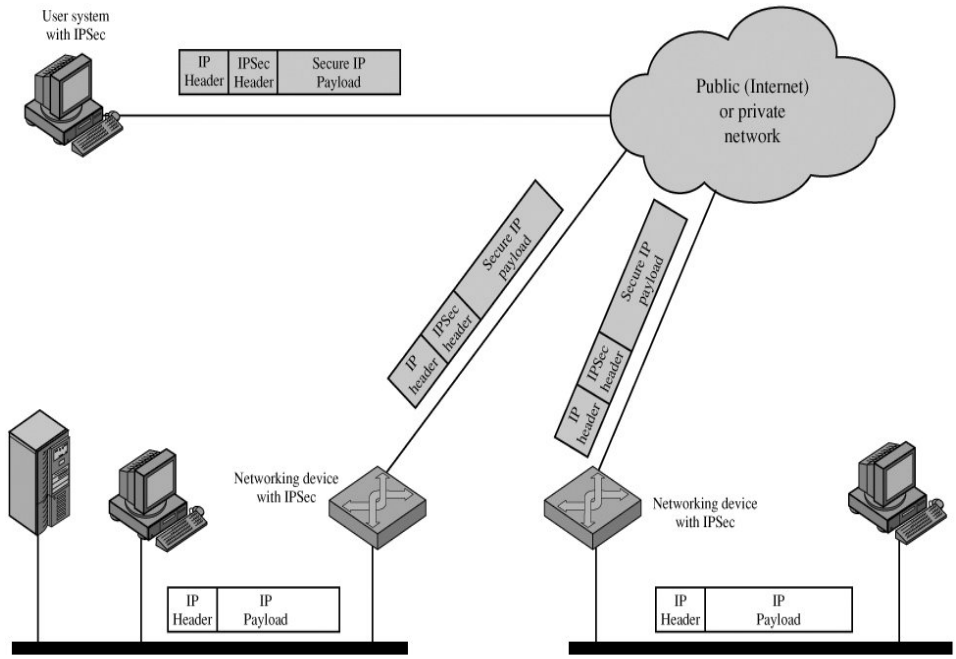


Fig 4.3.6 An IP Security Scenario

Figure 4.3.6 is a typical scenario of IPSec usage. An organization maintains LANs at dispersed locations. Nonsecure IP traffic is conducted on each LAN. For traffic offsite, through some sort of private or public WAN, IPSec protocols are used. These protocols operate in networking devices, such as a router or firewall, that connect each LAN to the outside world. The IPSec networking device will typically encrypt and compress all traffic going into the WAN, and decrypt and decompress traffic coming from the WAN; these operations are transparent to workstations and servers on the LAN. Secure transmission is also possible with individual users who dial into the WAN. Such user workstations must implement the IPSec protocols to provide security.

Benefits of IPSec

- When IPSec is implemented in a firewall or router, it provides strong security that can be applied to all traffic crossing the perimeter. Traffic within a company or workgroup does not incur the overhead of security-related processing.

- IPSec in a firewall is resistant to bypass if all traffic from the outside must use IP, and the firewall is the only means of entrance from the Internet into the organization.

- IPSec is below the transport layer (TCP, UDP) and so is transparent to applications. There is no need to change software on a user or server system when IPSec is implemented in the firewall or router. Even if IPSec is implemented in end systems, upper-layer software, including applications, is not affected.

- IPSec can be transparent to end users. There is no need to train users on security mechanisms, issue keying material on a per-user basis, or revoke keying material when users leave the organization.

- IPSec can provide security for individual users if needed. This is useful for offsite workers and for setting up a secure virtual subnetwork within an organization for sensitive applications.

IPSec Documents

The IPSec specification consists of numerous documents. The documents are divided into seven groups, as depicted in Fig 4.3.7 (RFC 2401).

- **Architecture:** Covers the general concepts, security requirements, definitions, and mechanisms defining IPSec technology.

- **Encapsulating Security Payload (ESP):** Covers the packet format and general issues related to the use of the ESP for packet encryption and, optionally, authentication.

- **Authentication Header (AH):** Covers the packet format and general issues related to the use of AH for packet authentication.

- **Encryption Algorithm:** A set of documents that describe how various encryption algorithms are used for ESP.

- **Authentication Algorithm:** A set of documents that describe how various authentication algorithms are used for AH and for the authentication option of ESP.

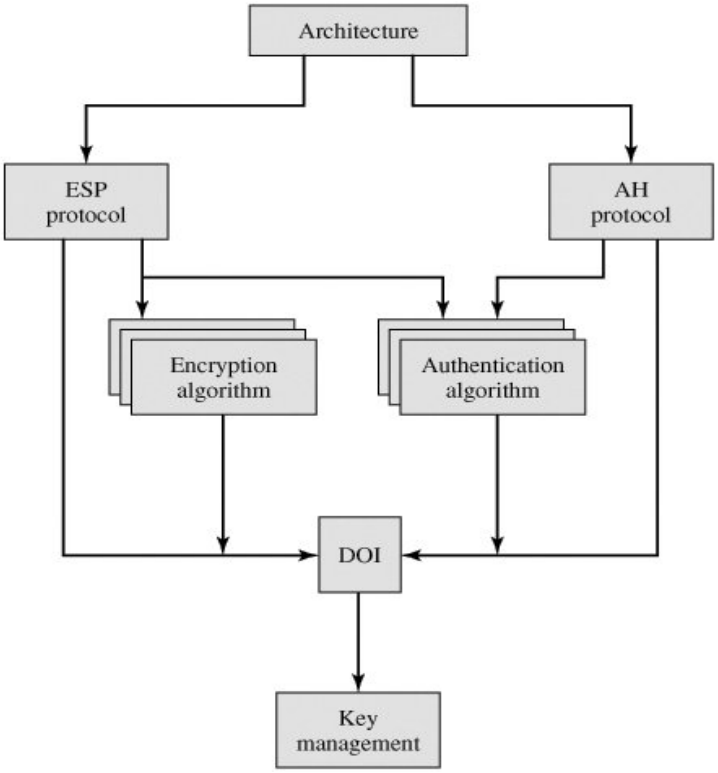


Fig 4.3.7 IPSec Document Overview

- **Key Management:** Documents that describe key management schemes.

- **Domain of Interpretation (DOI):** Contains values needed for the other documents to relate to each other. These include identifiers for approved encryption and authentication algorithms, as well as operational parameters such as key lifetime.

IPSec Services

IPSec provides security services at the IP layer by enabling a system to select required security protocols, determine the algorithm(s) to use for the service(s), and put in place any cryptographic keys required to provide the requested services. Two protocols are used to provide security: an authentication protocol designated by the header of the protocol, Authentication Header (AH); and a combined encryption/authentication protocol designated by the format of the packet for that protocol, Encapsulating Security Payload (ESP). The services are

- Access control
- Connectionless integrity
- Data origin authentication
- Rejection of replayed packets (a form of partial sequence integrity)
- Confidentiality (encryption)
- Limited traffic flow confidentiality

Table 4.3.4 shows which services are provided by the AH and ESP protocols. For ESP, there are two cases: with and without the authentication option. Both AH and ESP are vehicles for access control, based on the distribution of cryptographic keys and the management of traffic flows relative to these security protocols.

	AH	ESP (encryption only)	ESP (encryption plus authentication)
Access control	✔	✔	✔
Connectionless integrity	✔		✔
Data origin authentication	✔		✔
Rejection of replayed packets	✔	✔	✔
Confidentiality		✔	✔
Limited traffic flow confidentiality		✔	✔

Table 4.3.4 IPSec Services

Transport and Tunnel Modes

Both AH and ESP support two modes of use:

- Transport and
- Tunnel mode.

Transport Mode

Transport mode provides protection primarily for upper-layer protocols. That is, transport mode protection extends to the payload of an IP packet. Examples include a TCP or UDP segment or an ICMP packet, all of which operate directly above IP in a host protocol stack. Typically, transport mode is used for end-to-end communication between two hosts (e.g., a client and a server, or two workstations). When a host runs AH or ESP over IPv4, the payload is the data that normally follow the IP header. For IPv6, the payload is the data that normally follow both the IP header

and any IPv6 extensions headers that are present, with the possible exception of the destination options header, which may be included in the protection.

ESP in transport mode encrypts and optionally authenticates the IP payload but not the IP header. AH in transport mode authenticates the IP payload and selected portions of the IP header.

Tunnel Mode

Tunnel mode provides protection to the entire IP packet. To achieve this, after the AH or ESP fields are added to the IP packet, the entire packet plus security fields is treated as the payload of new "outer" IP packet with a new outer IP header. The entire original, or inner, packet travels through a "tunnel" from one point of an IP network to another; no routers along the way are able to examine the inner IP header. Because the original packet is encapsulated, the new, larger packet may have totally different source and destination addresses, adding to the security. Tunnel mode is used when one or both ends of an SA are a security gateway, such as a firewall or router that implements IPSec. With tunnel mode, a number of hosts on networks behind firewalls may engage in secure communications without implementing IPSec. The unprotected packets generated by such hosts are tunneled through external networks by tunnel mode SAs set up by the IPSec software in the firewall or secure router at the boundary of the local network.

Table 4.3.5 summarizes transport and tunnel mode functionality.

	Transport Mode SA	**Tunnel Mode SA**
AH	Authenticates IP payload and selected portions of IP header and IPv6 extension headers.	Authenticates entire inner IP packet (inner header plus IP payload) plus selected portions of outer IP header and outer IPv6 extension headers.
ESP	Encrypts IP payload and any IPv6 extension headers following the ESP header.	Encrypts entire inner IP packet.
ESP with Authentication	Encrypts IP payload and any IPv6 extension headers following the ESP header. Authenticates IP payload but not IP header.	Encrypts entire inner IP packet. Authenticates inner IP packet.

Table 4.3.5 Tunnel Mode and Transport Mode Functionality

Authentication Header

The Authentication Header provides support for data integrity and authentication of IP packets. The data integrity feature ensures that undetected modification to a packet's content in transit is not possible. The authentication feature enables an end system or network device to authenticate the user or application and filter traffic accordingly; it also prevents the address spoofing attacks observed in today's Internet.

Authentication is based on the use of a message authentication code (MAC). Hence the two parties must share a secret key.

The Authentication Header consists of the following fields (Figure 4.3.8):

- **Next Header (8 bits):** Identifies the type of header immediately following this header.
- **Payload Length (8 bits):** Length of Authentication Header in 32-bit words, minus 2. For example, the default length of the authentication data field is 96 bits, or three 32-bit words. With a three-word fixed header, there are a total of six words in the header, and the Payload Length field has a value of 4.
- **Reserved (16 bits):** For future use.
- **Security Parameters Index (32 bits):** Identifies a security association.
- **Sequence Number (32 bits):** A monotonically increasing counter value, discussed later.
- **Authentication Data (variable):** A variable-length field (must be an integral number of 32-bit words) that contains the Integrity Check Value (ICV), or MAC, for this packet, discussed later.

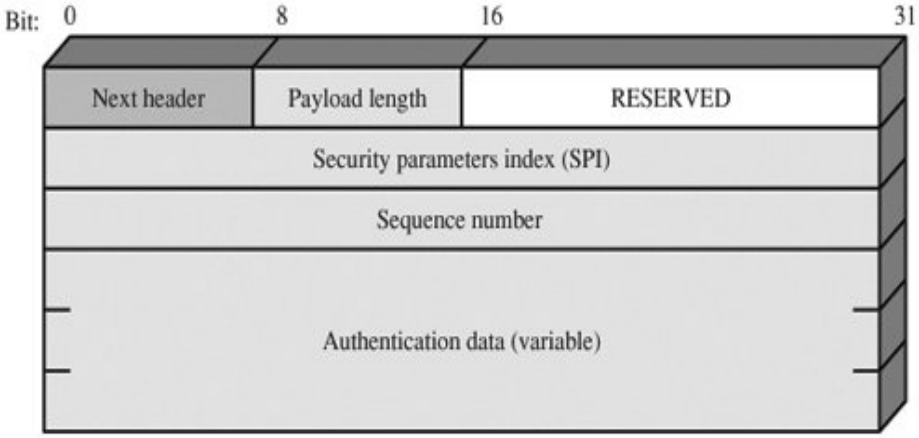


Fig 4.3.8 IPSec Authentication Header

Transport and Tunnel Modes

Figure 4.3.9 shows two ways in which the IPSec authentication service can be used. In one case, authentication is provided directly between a server and client workstations; the workstation can be either on the same network as the server or on an external network. As long as the workstation and the server share a protected secret key, the authentication process is secure. This case uses a transport mode SA. In the other case, a remote workstation authenticates itself to the corporate firewall, either for access to the entire internal network or because the requested server does not support the authentication feature. This case uses a tunnel mode SA.

Figure 4.3.10a shows typical IPv4 and IPv6 packets. In this case, the IP payload is a TCP segment; it could also be a data unit for any other protocol that uses IP, such as UDP or ICMP.

For transport mode AH using IPv4, the AH is inserted after the original IP header and before the IP payload (e.g., a TCP segment); this is shown in the upper part of Figure 4.3.10b. Authentication covers the entire packet, excluding mutable fields in the IPv4 header that are set to zero for MAC calculation.

In the context of IPv6, AH is viewed as an end-to-end payload; that is, it is not examined or processed by intermediate routers. Therefore, the AH appears after the IPv6 base header and the hop-by-hop, routing, and fragment extension headers. The destination options extension header could appear before or after the AH header, depending on the semantics desired. Again, authentication covers the entire packet, excluding mutable fields that are set to zero for MAC calculation

For tunnel mode AH, the entire original IP packet is authenticated, and the AH is inserted between the original IP header and a new outer IP header (Figure 4.3.10c). The inner IP header carries the ultimate source and destination addresses, while an outer IP header may contain different IP addresses (e.g., addresses of firewalls or other security gateways).

With tunnel mode, the entire inner IP packet, including the entire inner IP header is protected by AH. The outer IP header (and in the case of IPv6, the outer IP extension headers) is protected except for mutable and unpredictable fields.

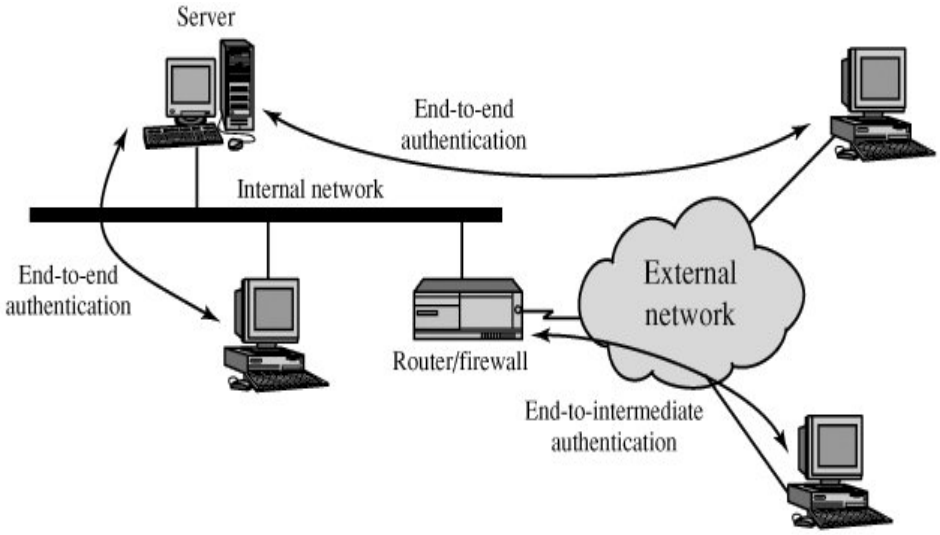


Fig 4.3.9 End-to-End versus End-to-Intermediate Authentication

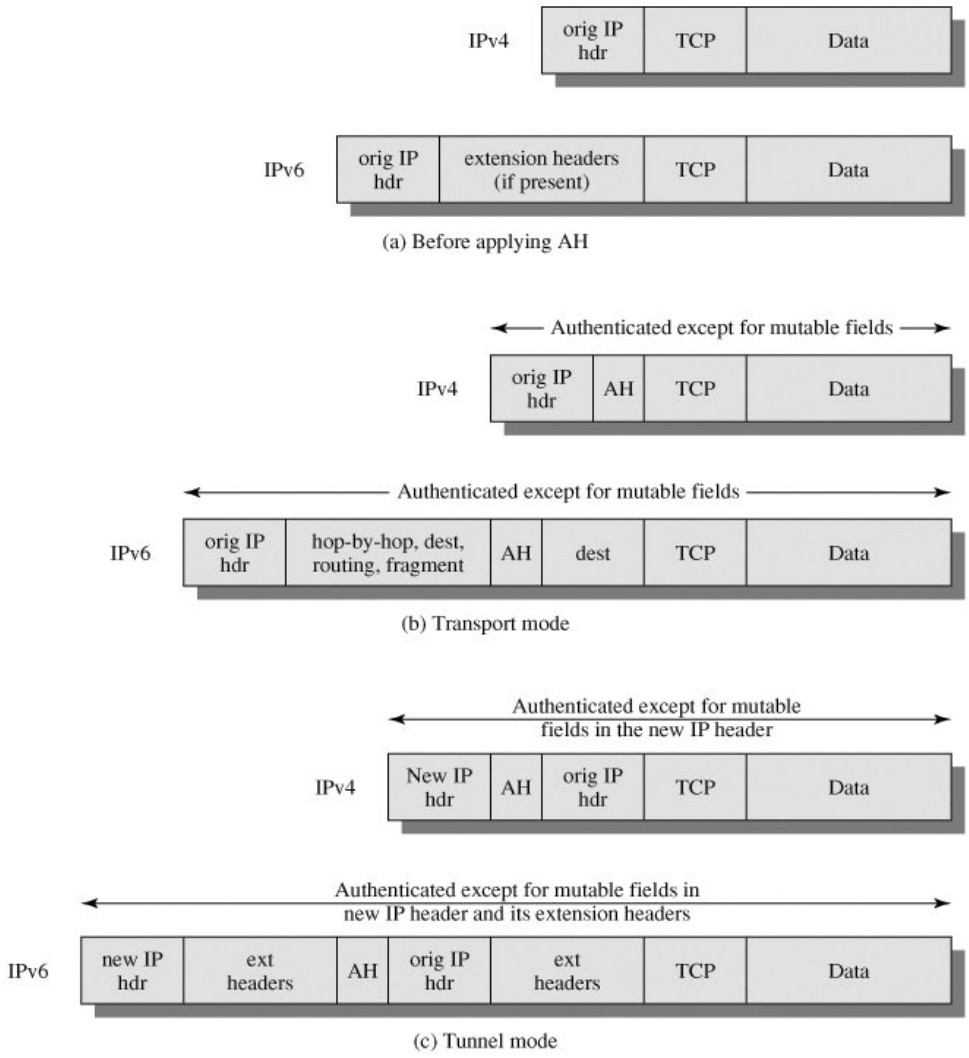


Fig 4.3.10 End-to-End versus End-to-Intermediate Authentication

Encapsulating Security Payload

The Encapsulating Security Payload provides confidentiality services, including confidentiality of message contents and limited traffic flow confidentiality. As an optional feature, ESP can also provide an authentication service.

ESP Format

Figure 4.3.11 shows the format of an ESP packet. It contains the following fields:

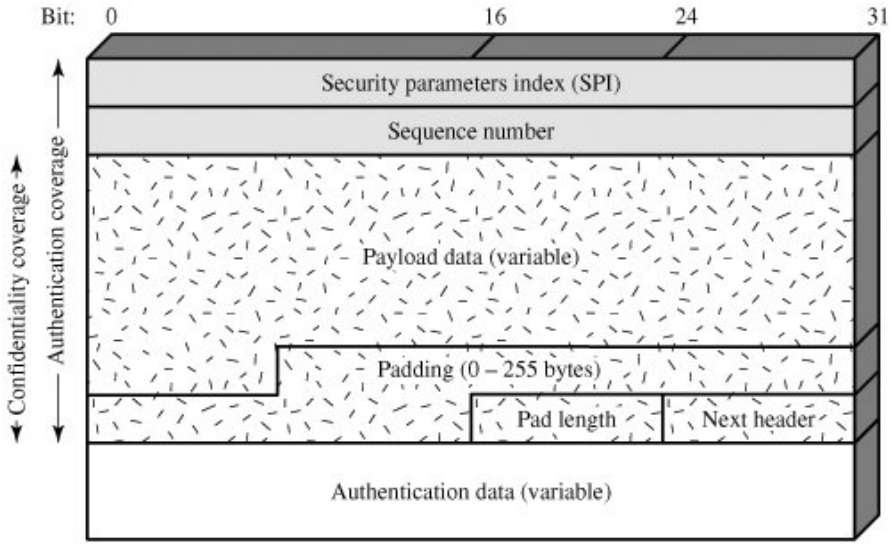


Fig 4.3.11 IPSec ESP format

- Security Parameters Index (32 bits): Identifies a security association.
- Sequence Number (32 bits): A monotonically increasing counter value; this provides an anti-replay function, as discussed for AH.
- Payload Data (variable): This is a transport-level segment (transport mode) or IP packet (tunnel mode) that is protected by encryption.
- Padding (0255 bytes): The purpose of this field is discussed later.
- Pad Length (8 bits): Indicates the number of pad bytes immediately preceding this field.
- Next Header (8 bits): Identifies the type of data contained in the payload data field by identifying the first header in that payload (for example, an extension header in IPv6, or an upper-layer protocol such as TCP).
- Authentication Data (variable): A variable-length field (must be an integral number of 32-bit words) that contains the Integrity Check Value computed over the ESP packet minus the Authentication Data field.

Encryption and Authentication Algorithms

The Payload Data, Padding, Pad Length, and Next Header fields are encrypted by the ESP service. If the algorithm used to encrypt the payload requires cryptographic synchronization data, such as an initialization vector (IV), then these data may be carried explicitly at the

beginning of the Payload Data field. If included, an IV is usually not encrypted, although it is often referred to as being part of the ciphertext.

The current specification dictates that a compliant implementation must support DES in cipher block chaining (CBC) mode. A number of other algorithms have been assigned identifiers in the DOI document and could therefore easily be used for encryption; these include

- Three-key triple DES
- RC5
- IDEA
- Three-key triple IDEA
- CAST
- Blowfish

Padding

The Padding field serves several purposes:

- If an encryption algorithm requires the plaintext to be a multiple of some number of bytes (e.g., the multiple of a single block for a block cipher), the Padding field is used to expand the plaintext (consisting of the Payload Data, Padding, Pad Length, and Next Header fields) to the required length.
- The ESP format requires that the Pad Length and Next Header fields be right aligned within a 32-bit word. Equivalently, the ciphertext must be an integer multiple of 32 bits. The Padding field is used to assure this alignment.
- Additional padding may be added to provide partial traffic flow confidentiality by concealing the actual length of the payload.

Transport and Tunnel Modes

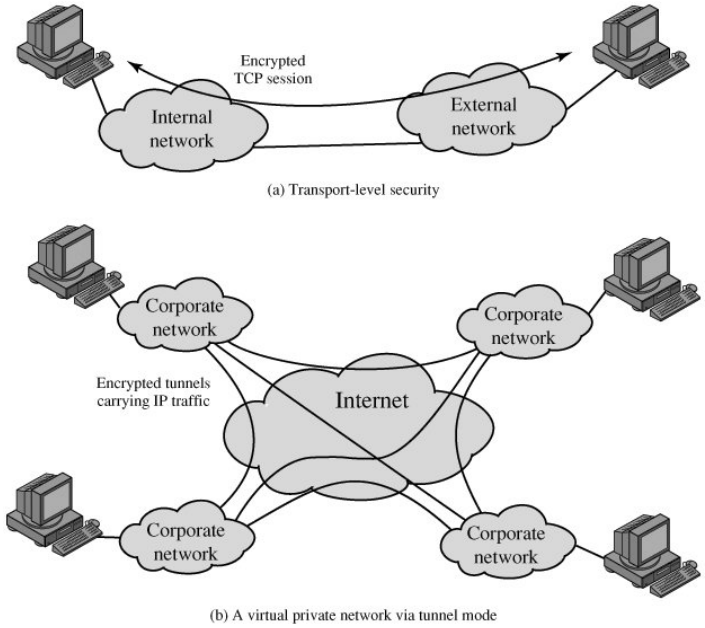


Fig 4.3.12 Transport-Mode vs. Tunnel-Mode Encryption

Figure 4.3.12 shows two ways in which the IPSec ESP service can be used. In the upper part of the figure, encryption (and optionally authentication) is provided directly between two hosts. Figure4.3.12b shows how tunnel mode operation can be used to set up a *virtual private network*. In this example, an organization has four private networks interconnected across the Internet. Hosts on the internal networks use the Internet for transport of data but do not interact with other Internet-based hosts. By terminating the tunnels at the security gateway to each internal network, the configuration allows the hosts to avoid implementing the security capability. The former technique is support by a transport mode SA, while the latter technique uses a tunnel mode SA.

Transport Mode ESP

Transport mode ESP is used to encrypt and optionally authenticate the data carried by IP (e.g., a TCP segment), as shown in Figure 4.3.13a. For this mode using IPv4, the ESP header is inserted into the IP packet immediately prior to the transport-layer header (e.g., TCP, UDP, ICMP) and an ESP trailer (Padding, Pad Length, and Next Header fields) is placed after the IP packet; if authentication is selected, the ESP Authentication Data field is added after the ESP trailer. The entire transport-level segment plus the ESP trailer are encrypted. Authentication covers all of the ciphertext plus the ESP header.

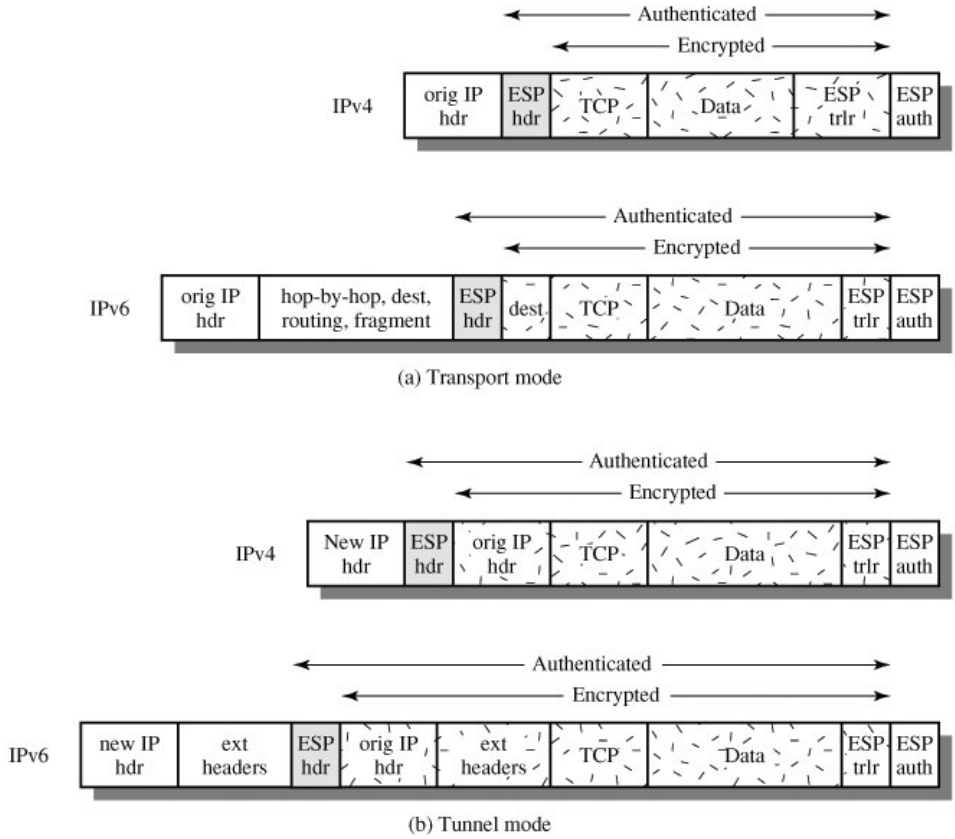


Fig 4.3.13 Scope of ESP Encryption and Authentication

In the context of IPv6, ESP is viewed as an end-to-end payload; that is, it is not examined or processed by intermediate routers. Therefore, the ESP header appears after the IPv6 base header and the hop-by-hop, routing, and fragment extension headers. The destination options extension header could appear before or after the ESP header, depending on the semantics desired. For IPv6, encryption covers the entire transport-level segment plus the ESP trailer plus the destination options extension header if it occurs after the ESP header. Again, authentication covers the ciphertext plus the ESP header.

Transport mode operation may be summarized as follows:

1. At the source, the block of data consisting of the ESP trailer plus the entire transport-layer segment is encrypted and the plaintext of this block is replaced with its ciphertext to form the IP packet for transmission. Authentication is added if this option is selected.

2. The packet is then routed to the destination. Each intermediate router needs to examine and process the IP header plus any plaintext IP extension headers but does not need to examine the ciphertext.

3. The destination node examines and processes the IP header plus any plaintext IP extension headers. Then, on the basis of the SPI in the ESP header, the destination node decrypts the remainder of the packet to recover the plaintext transport-layer segment.

Transport mode operation provides confidentiality for any application that uses it, thus avoiding the need to implement confidentiality in every individual application. This mode of operation is also reasonably efficient, adding little to the total length of the IP packet. One drawback to this mode is that it is possible to do traffic analysis on the transmitted packets.

Tunnel Mode ESP

Tunnel mode ESP is used to encrypt an entire IP packet (Figure 4.3.13b). For this mode, the ESP header is prefixed to the packet and then the packet plus the ESP trailer is encrypted. This method can be used to counter traffic analysis.

Because the IP header contains the destination address and possibly source routing directives and hop-by-hop option information, it is not possible simply to transmit the encrypted IP packet prefixed by the ESP header. Intermediate routers would be unable to process such a packet. Therefore, it is necessary to encapsulate the entire block (ESP header plus ciphertext plus Authentication Data, if present) with a new IP header that will contain sufficient information for routing but not for traffic analysis.

Whereas the transport mode is suitable for protecting connections between hosts that support the ESP feature, the tunnel mode is useful in a configuration that includes a firewall or other sort of security gateway that protects a trusted network from external networks. In this latter case, encryption occurs only between an external host and the security gateway or between two security gateways. This relieves hosts on the internal network of the processing burden of encryption and simplifies the key distribution task by reducing the number of needed keys. Further, it thwarts traffic analysis based on ultimate destination.

REFERENCES

1. William Stallings, Cryptography and Network Security, PHI Publishers

Review Questions:

1. Give an account of PGP message format
2. Explain in detail PGP message generation and reception
3. Write a short note on S/MIME
4. Explain S/MIME content types in detail
5. Explain about IPSEC in detail
6. Explain Transport and Tunnel modes of operation of IPSec in detail

4. WEB SECURITY

OBJECTIVE

This lesson looks at the IP security scheme that has been developed to operate both with the current IP and the emerging next-generation IP, known as IPv6 and issues of web-security including Secured Socket Layer (SSL) and Secure Electronic Transaction (SET).

SECURED SOCKET LAYER

Secured Socket Layer, abbreviated as SSL, is a protocol developed by Netscape for transmitting private documents via the Internet. SSL uses a cryptographic system that uses two keys to encrypt data − a public key known to everyone and a private or secret key known only to the recipient of the message. Both Netscape Navigator and Internet Explorer support SSL, and many Web sites use the protocol to obtain confidential user information, such as credit card numbers. By convention, URLs that require an SSL connection start with *https:* instead of *http*:.

Secure Sockets Layer (SSL) technology protects your Web site and makes it easy for your Web site visitors to trust you in three essential ways:

1. An SSL Certificate enables **encryption** of sensitive information during online transactions.

2. Each SSL Certificate contains unique, **authenticated** information about the certificate owner.

3. A Certificate Authority **verifies** the identity of the certificate owner when it is issued.

You need SSL if

- you have an online store or accept online orders and credit cards
- you offer a login or sign in on your site
- you process sensitive data such as address, birth date, license, or ID numbers
- you need to comply with privacy and security requirements
- you value privacy and expect others to trust you**.**

Another protocol for transmitting data securely over the World Wide Web is Secure HTTP (S-HTTP). Whereas SSL creates a secure connection between a client and a server, over which any amount of data can be sent securely, S-HTTP is designed to transmit individual messages securely. SSL and S-HTTP, therefore, can be seen as complementary rather than competing technologies. Both protocols have been approved by the Internet Engineering Task Force (IETF) as a standard.

SSL Architecture

SSL is designed to make use of TCP to provide a reliable end-to-end secure service. SSL is not a single protocol but rather two layers of protocols, as illustrated in Figure 4.3.14.

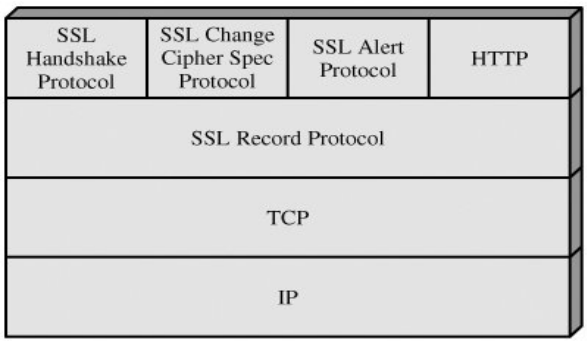


Fig 4.3.14 SSL Protocol Stack

The SSL Record Protocol provides basic security services to various higher-layer protocols. In particular, the Hypertext Transfer Protocol (HTTP), which provides the transfer service for Web client/server interaction, can operate on top of SSL. Three higher-layer protocols are defined as part of SSL: the Handshake Protocol, The Change Cipher Spec Protocol, and the Alert Protocol. These SSL-specific protocols are used in the management of SSL exchanges and are examined later in this section.

Two important SSL concepts are the SSL session and the SSL connection, which are defined in the specification as follows:

- **Connection:** A connection is a transport (in the OSI layering model definition) that provides a suitable type of service. For SSL, such connections are peer-to-peer relationships. The connections are transient. Every connection is associated with one session.

- **Session:** An SSL session is an association between a client and a server. Sessions are created by the Handshake Protocol. Sessions define a set of cryptographic security parameters, which can be shared among multiple connections. Sessions are used to avoid the expensive negotiation of new security parameters for each connection.

Between any pair of parties (applications such as HTTP on client and server), there may be multiple secure connections. There are actually a number of states associated with each session. Once a session is established, there is a current operating state for both read and write (i.e., receive and send). In addition, during the Handshake Protocol, pending read and write states are created. Upon successful conclusion of the Handshake Protocol, the pending states become the current states.

SSL Record Protocol

The SSL Record Protocol provides two services for SSL connections:

- **Confidentiality:** The Handshake Protocol defines a shared secret key that is used for conventional encryption of SSL payloads.
- **Message Integrity:** The Handshake Protocol also defines a shared secret key that is used to form a message authentication code (MAC).

Figure 4.3.15 indicates the overall operation of the SSL Record Protocol. The Record Protocol takes an application message to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts, adds a header, and transmits the resulting unit in a TCP segment. Received data are decrypted, verified, decompressed, and reassembled and then delivered to higher-level users.

The first step is fragmentation. Each upper-layer message is fragmented into blocks of 2^{14} bytes (16384 bytes) or less. Next, compression is optionally applied. Compression must be lossless and may not increase the content length by more than 1024 bytes.

The next step in processing is to compute a **message authentication code** over the compressed data. For this purpose, a shared secret key is used.

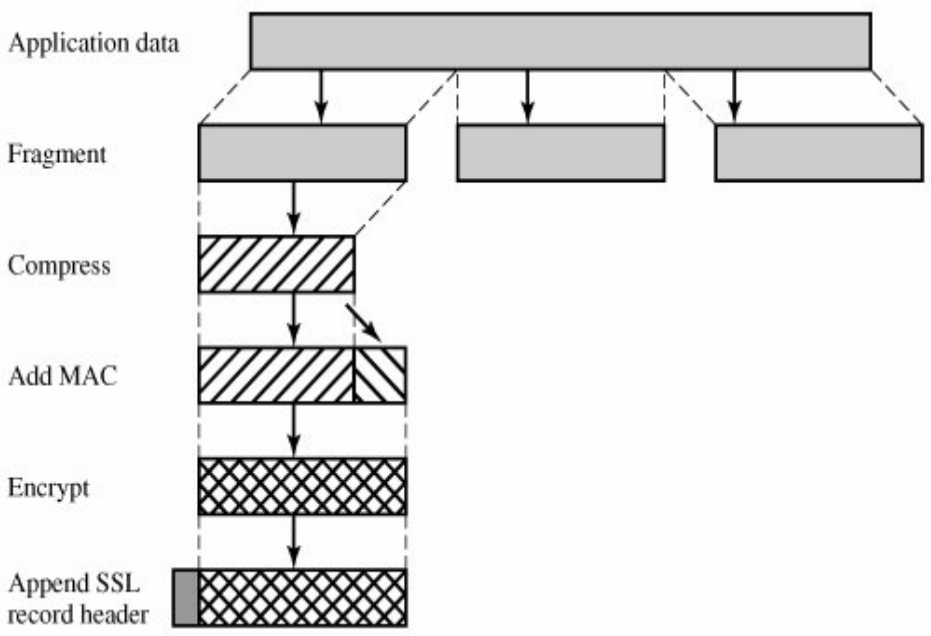


Fig 4.3.15 SSL Record Protocol Operation

The calculation is defined as:

```
hash(MAC_write_secret || pad_2 ||
      hash(MAC_write_secret || pad_1 || seq_num ||
      SSLCompressed.type ||
      SSLCompressed.length || SSLCompressed.fragment))
```

where

||	=	concatenation
MAC_write_secret	=	shared secret key
hash	=	cryptographic hash algorithm; either MD5 or SHA-1
pad_1	=	the byte 0x36 (0011 0110) repeated 48 times (384 bits) for MD5 and 40 times (320 bits) for SHA-1
pad_2	=	the byte 0x5C (0101 1100) repeated 48 times for MD5 and 40 times for SHA-1

||	=	concatenation
seq_num	=	the sequence number for this message
SSLCompressed.type	=	the higher-level protocol used to process this fragment
SSLCompressed.length	=	the length of the compressed fragment
SSLCompressed.fragment	=	the compressed fragment (if compression is not used, the plaintext fragment)

Next, the compressed message plus the MAC are encrypted using symmetric encryption. Encryption may not increase the content length by more than 1024 bytes, so that the total length may not exceed 2^{14} + 2048. The following encryption algorithms are permitted:

Block Cipher		**Stream Cipher**	
Algorithm	**Key Size**	**Algorithm**	**Key Size**
AES	128,256	RC4-40	40
IDEA	128	RC4-128	128
RC2-40	40		
DES-40	40		
DES	56		
3DES	168		
Fortezza	80		

For stream encryption, the compressed message plus the MAC are encrypted. Note that the MAC is computed before encryption takes place

and that the MAC is then encrypted along with the plaintext or compressed plaintext.

For block encryption, padding may be added after the MAC prior to encryption. The padding is in the form of a number of padding bytes followed by a one-byte indication of the length of the padding. The total amount of padding is the smallest amount such that the total size of the data to be encrypted (plaintext plus MAC plus padding) is a multiple of the cipher's block length. An example is a plaintext (or compressed text if compression is used) of 58 bytes, with a MAC of 20 bytes (using SHA-1), that is encrypted using a block length of 8 bytes (e.g., DES). With the padding.length byte, this yields a total of 79 bytes. To make the total an integer multiple of 8, one byte of padding is added.

The final step of SSL Record Protocol processing is to prepend a header, consisting of the following fields:

- **Content Type (8 bits):** The higher layer protocol used to process the enclosed fragment.
- **Major Version (8 bits):** Indicates major version of SSL in use. For SSLv3, the value is 3.
- **Minor Version (8 bits):** Indicates minor version in use. For SSLv3, the value is 0.
- **Compressed Length (16 bits):** The length in bytes of the plaintext fragment (or compressed fragment if compression is used). The maximum value is 2^{14} + 2048.

The content types that have been defined are change_cipher_spec, alert, handshake, and application_data. The first three are the SSL-specific protocols. Figure 4.3.16 illustrates the SSL record format.

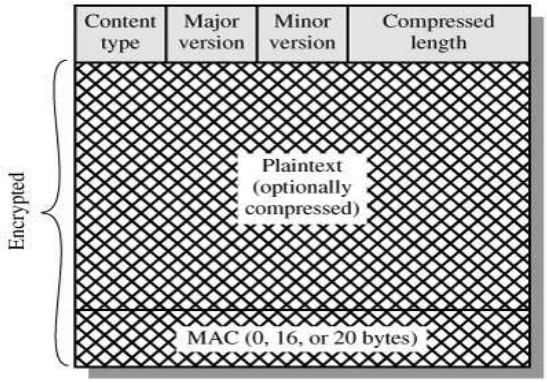


Fig 4.3.16 SSL Record Format

Change Cipher Spec Protocol

The Change Cipher Spec Protocol is one of the three SSL-specific protocols that use the SSL Record Protocol, and it is the simplest. This protocol consists of a single message (Figure 4.3.17a), which consists of a single byte with the value 1. The sole purpose of this message is to cause the pending state to be copied into the current state, which updates the cipher suite to be used on this connection.

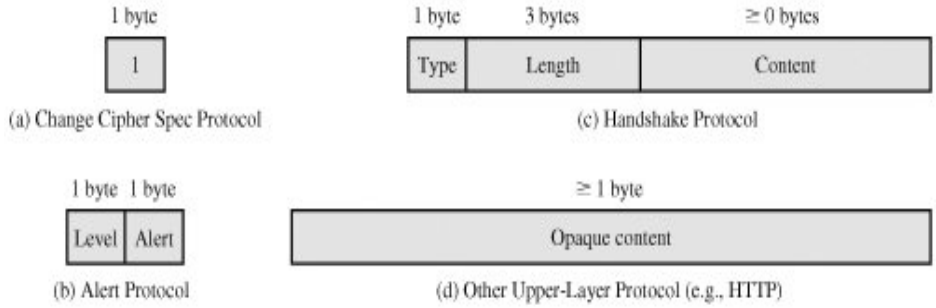


Fig 4.3.17 SSL Record Protocol Payload

Alert Protocol

The Alert Protocol is used to convey SSL-related alerts to the peer entity. As with other applications that use SSL, alert messages are compressed and encrypted, as specified by the current state.

Each message in this protocol consists of two bytes (Figure 4.3.17b). The first byte takes the value warning(1) or fatal(2) to convey the severity of the message. If the level is fatal, SSL immediately terminates the connection. Other connections on the same session may continue, but no new connections on this session may be established. The second byte contains a code that indicates the specific alert. First, we list those alerts that are always fatal (definitions from the SSL specification):

- **unexpected_message:** An inappropriate message was received.
- **bad_record_mac**: An incorrect MAC was received.
- **decompression_failure:** The decompression function received improper input (e.g., unable to decompress or decompress to greater than maximum allowable length).
- **handshake_failure:** Sender was unable to negotiate an acceptable set of security parameters given the options available.

- **illegal_parameter:** A field in a handshake message was out of range or inconsistent with other fields.
- **close_notify:** Notifies the recipient that the sender will not send any more messages on this connection. Each party is required to send a close_notify alert before closing the write side of a connection.
- **no_certificate:** May be sent in response to a certificate request if no appropriate certificate is available.
- **bad_certificate:** A received certificate was corrupt (e.g., contained a signature that did not verify).
- **unsupported_certificate:** The type of the received certificate is not supported.
- **certificate_revoked:** A certificate has been revoked by its signer.
- **certificate_expired:** A certificate has expired.
- **certificate_unknown:** Some other unspecified issue arose in processing the certificate, rendering it unacceptable.
-

Handshake Protocol

The most complex part of SSL is the Handshake Protocol. This protocol allows the server and client to authenticate each other and to negotiate an encryption and MAC algorithm and cryptographic keys to be used to protect data sent in an SSL record. The Handshake Protocol is used before any application data is transmitted.

The Handshake Protocol consists of a series of messages exchanged by client and server. All of these have the format shown in Figure 4.3.17c. Each message has three fields:

- Type (1 byte): Indicates one of 10 messages.
- Length (3 bytes): The length of the message in bytes.
- Content (≥0 bytes): The parameters associated with this message(Table 4.3.6).

Message Type	**Parameters**
hello_request	null

Message Type	Parameters
client_hello	version, random, session id, cipher suite, compression method
server_hello	version, random, session id, cipher suite, compression method
certificate	chain of X.509v3 certificates
server_key_exchange	parameters, signature
certificate_request	type, authorities
server_done	null
certificate_verify	signature
client_key_exchange	parameters, signature
finished	hash value

Table 4.3.6 SSL Handshake Protocol Message Types

Figure 4.3.18 shows the initial exchange needed to establish a logical connection between client and server. The exchange can be viewed as having four phases.

Phase1. Establish Security Capabilities

This phase is used to initiate a logical connection and to establish the security capabilities that will be associated with it. The exchange is initiated by the client, which sends a client_hello message with the following parameters:

- **Version:** The highest SSL version understood by the client.
- **Random:** A client-generated random structure, consisting of a 32-bit timestamp and 28 bytes generated by a secure random number generator. These values serve as nonces and are used during key exchange to prevent replay attacks.

- **Session ID:** A variable-length session identifier. A nonzero value indicates that the client wishes to update the parameters of an existing connection or create a new connection on this session. A zero value indicates that the client wishes to establish a new connection on a new session.

- **CipherSuite:** This is a list that contains the combinations of cryptographic algorithms supported by the client, in decreasing order of preference. Each element of the list (each cipher suite) defines both a key exchange algorithm and a CipherSpec.

- **Compression Method:** This is a list of the compression methods the client supports.

After sending the client_hello message, the client waits for the server_hello message, which contains the same parameters as the client_hello message. For the server_hello message, the following conventions apply. The Version field contains the lower of the version suggested by the client and the highest supported by the server. The Random field is generated by the server and is independent of the client's Random field. If the SessionID field of the client was nonzero, the same value is used by the server; otherwise the server's SessionID field contains the value for a new session. The CipherSuite field contains the single cipher suite selected by the server from those proposed by the client. The Compression field contains the compression method selected by the server from those proposed by the client.

The first element of the Cipher Suite parameter is the key exchange method (i.e., the means by which the cryptographic keys for conventional encryption and MAC are exchanged).

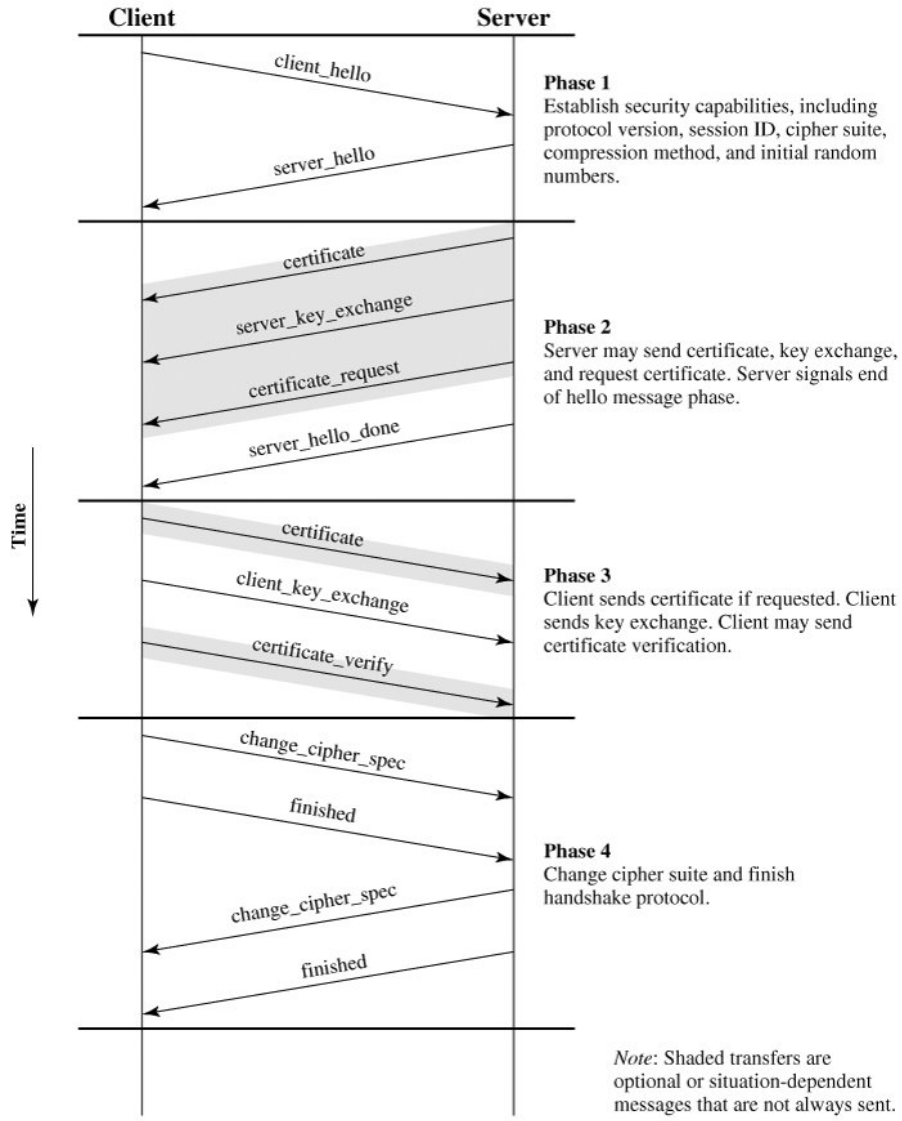


Fig 4.3.18 Handshake Protocol Action

The following key exchange methods are supported:

- **RSA**
- **Fixed Diffie-Hellman**.
- **Ephemeral Diffie**
- **Anonymous Diffie-Hellman**
- **Fortezza**

Following the definition of a key exchange method is the CipherSpec, which includes the following fields:

- **CipherAlgorithm**: Any of the algorithms mentioned earlier: RC4, RC2, DES, 3DES, DES40, IDEA, Fortezza
- **MACAlgorithm:** MD5 or SHA-1
- **CipherType:** Stream or Block
- **IsExportable:** True or False
- **HashSize:** 0, 16 (for MD5), or 20 (for SHA-1) bytes
- **Key Material:** A sequence of bytes that contain data used in generating the write keys
- **IV Size:** The size of the Initialization Value for Cipher Block Chaining (CBC) encryption

Phase2. Server Authentication and Key Exchange

The server begins this phase by sending its certificate, if it needs to be authenticated; the message contains one or a chain of X.509 certificates. The certificate message is required for any agreed-on key exchange method except anonymous Diffie-Hellman.

Next, a server_key_exchange message may be sent if it is required. It is not required in two instances:

(1) The server has sent a certificate with fixed Diffie-Hellman parameters

(2) RSA key exchange is to be used.

In this case the hash is defined as

hash(ClientHello.random || ServerHello.random || ServerParams)

So the hash covers not only the Diffie-Hellman or RSA parameters, but also the two nonces from the initial hello messages. This ensures against replay attacks and misrepresentation. In the case of a DSS signature, the hash is performed using the SHA-1 algorithm. In the case of an RSA signature, both an MD5 and an SHA-1 hash are calculated, and the concatenation of the two hashes (36 bytes) is encrypted with the server's private key.

Next, a nonanonymous server (server not using anonymous Diffie-Hellman) can request a certificate from the client. The certificate_request message includes two parameters: certificate_type and certificate_authorities. The certificate type indicates the public-key algorithm and its use:

- RSA, signature only
- DSS, signature only
- RSA for fixed Diffie-Hellman; in this case the signature is used only for authentication, by sending a certificate signed with RSA
- DSS for fixed Diffie-Hellman; again, used only for authentication
- RSA for ephemeral Diffie-Hellman
- DSS for ephemeral Diffie-Hellman
- Fortezza

The second parameter in the certificate_request message is a list of the distinguished names of acceptable certificate authorities.

The final and always required message in Phase 2 is the server_done message, which is sent by the server to indicate the end of the server hello and associated messages. After sending this message, the server will wait for a client response. This message has no parameters.

Phase3. Client Authentication and Key Exchange

Upon receipt of the server_done message, the client should verify that the server provided a valid certificate if required and check that the server_hello parameters are acceptable. If all is satisfactory, the client sends one or more messages back to the server.

If the server has requested a certificate, the client begins this phase by sending a certificate message. If no suitable certificate is available, the client sends a no_certificate alert instead.

Next is the client_key_exchange message, which must be sent in this phase. The content of the message depends on the type of key exchange, as follows:

- **RSA:** The client generates a 48-byte pre-master secret and encrypts with the public key from the server's certificate or temporary RSA key from a server_key_exchange message. Its use to compute a master secret is explained later.
- **Ephemeral or Anonymous Diffie-Hellman:** The client's public Diffie-Hellman parameters are sent.

- **Fixed Diffie-Hellman:** The client's public Diffie-Hellman parameters were sent in a certificate message, so the content of this message is null.

- **Fortezza:** The client's Fortezza parameters are sent.

Finally, in this phase, the client may send a certificate_verify message to provide explicit verification of a client certificate. This message is only sent following any client certificate that has signing capability (i.e., all certificates except those containing fixed Diffie-Hellman parameters). This message signs a hash code based on the preceding messages, defined as follows:

```
CertificateVerify.signature.md5_hash

     MD5(master_secret||pad_2||

              MD5(handshake_messages||master_secret  ||
     pad_1));

Certificate.signature.sha_hash

      SHA(master_secret||pad_2||  HA(handshake_messages
|| master_secret || pad_1));
```

where pad_1 and pad_2 are the values defined earlier for the MAC, handshake_messages refers to all Handshake Protocol messages sent or received starting at client_hello but not including this message, and master_secret is the calculated secret whose construction is explained later in this section. If the user's private key is DSS, then it is used to encrypt the SHA-1 hash. If the user's private key is RSA, it is used to encrypt the concatenation of the MD5 and SHA-1 hashes. In either case, the purpose is to verify the client's ownership of the private key for the client certificate. Even if someone is misusing the client's certificate, he or she would be unable to send this message.

Phase4. Finish

This phase completes the setting up of a secure connection. The client sends a change_cipher_spec message and copies the pending CipherSpec into the current CipherSpec. Note that this message is not considered part of the Handshake Protocol but is sent using the Change Cipher Spec Protocol. The client then immediately sends the finished message under the new algorithms, keys, and secrets. The finished message verifies that the key exchange and authentication processes were successful. The content of the finished message is the concatenation of two hash values:

```
MD5(master_secret || pad2 || MD5(handshake_messages ||

    Sender || master_secret || pad1))
```

```
SHA(master_secret || pad2 || SHA(handshake_messages ||
    Sender || master_secret || pad1))
```

where Sender is a code that identifies that the sender is the client and handshake_messages is all of the data from all handshake messages up to but not including this message.

In response to these two messages, the server sends its own change_cipher_spec message, transfers the pending to the current CipherSpec, and sends its finished message. At this point the handshake is complete and the client and server may begin to exchange application layer data.

SECURE ELECTRONIC TRANSACTION

SET is an open encryption and security specification designed to protect credit card transactions on the Internet. SET is a set of security protocols and formats that enables users to employ the existing credit card payment infrastructure on an open network, such as the Internet, in a secure fashion. In essence, SET provides three services:

- Provides a secure communications channel among all parties involved in a transaction
- Provides trust by the use of X.509v3 digital certificates
- Ensures privacy because the information is only available to parties in a transaction when and where necessary

Key Features of SET

- **Confidentiality of information:** Cardholder account and payment information is secured as it travels across the network. An interesting and important feature of SET is that it prevents the merchant from learning the cardholder's credit card number; this is only provided to the issuing bank. Conventional encryption by DES is used to provide confidentiality.
- **Integrity of data:** Payment information sent from cardholders to merchants includes order information, personal data, and payment instructions. SET guarantees that these message contents are not altered in transit. RSA digital signatures, using SHA-1 hash codes, provide message integrity. Certain messages are also protected by HMAC using SHA-1.
- **Cardholder account authentication:** SET enables merchants to verify that a cardholder is a legitimate user of a valid card account

number. SET uses X.509v3 digital certificates with RSA signatures for this purpose.

- **Merchant authentication:** SET enables cardholders to verify that a merchant has a relationship with a financial institution allowing it to accept payment cards. SET uses X.509v3 digital certificates with RSA signatures for this purpose.

Note that unlike IPSec and SSL/TLS, SET provides only one choice for each cryptographic algorithm. This makes sense, because SET is a single application with a single set of requirements, whereas IPSec and SSL/TLS are intended to support a range of applications.

SET Participants

Figure 4.3.19 indicates the participants in the SET system, which include the following:

- **Cardholder:** In the electronic environment, consumers and corporate purchasers interact with merchants from personal computers over the Internet. A cardholder is an authorized holder of a payment card (e.g., MasterCard, Visa) that has been issued by an issuer.

- **Merchant:** A merchant is a person or organization that has goods or services to sell to the cardholder. Typically, these goods and services are offered via a Web site or by electronic mail. A merchant that accepts payment cards must have a relationship with an acquirer.

- **Issuer**: This is a financial institution, such as a bank, that provides the cardholder with the payment card. Typically, accounts are applied for and opened by mail or in person. Ultimately, it is the issuer that is responsible for the payment of the debt of the cardholder.

- **Acquirer**: This is a financial institution that establishes an account with a merchant and processes payment card authorizations and payments. Merchants will usually accept more than one credit card brand but do not want to deal with multiple bankcard associations or with multiple individual issuers. The acquirer provides authorization to the merchant that a given card account is active and that the proposed purchase does not exceed the credit limit. The acquirer also provides electronic transfer of payments to the merchant's account. Subsequently, the acquirer is reimbursed by the issuer over some sort of payment network for electronic funds transfer.

- **Payment gateway**: This is a function operated by the acquirer or a designated third party that processes merchant payment messages. The payment gateway interfaces between SET and

the existing bankcard payment networks for authorization and payment functions. The merchant exchanges SET messages with the payment gateway over the Internet, while the payment gateway has some direct or network connection to the acquirer's financial processing system.

- **Certification authority (CA):** This is an entity that is trusted to issue X.509v3 public-key certificates for cardholders, merchants, and payment gateways. The success of SET will depend on the existence of a CA infrastructure available for this purpose. As was discussed in previous chapters, a hierarchy of CAs is used, so that participants need not be directly certified by a root authority.

The following are the events that are required for a transaction. The customer opens an account. The customer obtains a credit card account, such as MasterCard or Visa, with a bank that supports electronic payment and SET.

1. The customer receives a certificate. After suitable verification of identity, the customer receives an X.509v3 digital certificate, which is signed by the bank. The certificate verifies the customer's RSA public key and its expiration date. It also establishes a relationship, guaranteed by the bank, between the customer's key pair and his or her credit card.

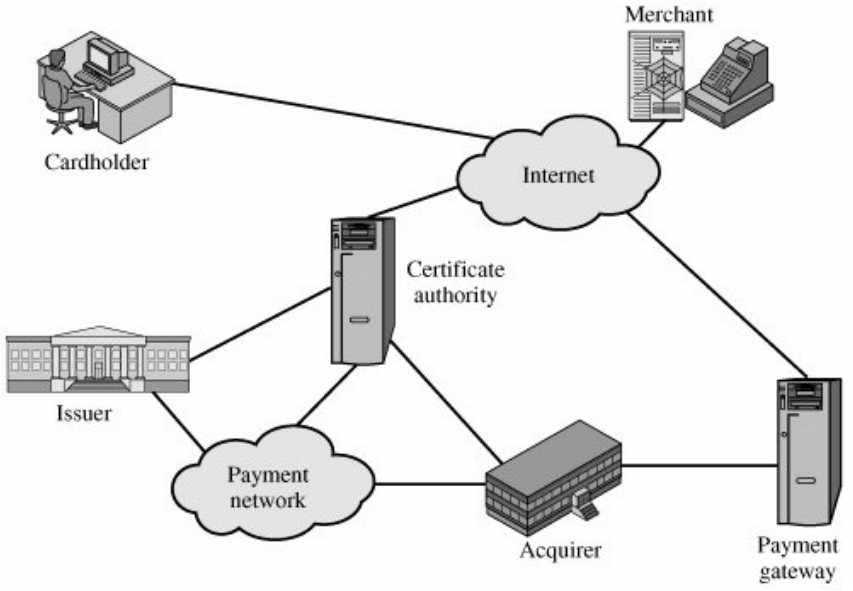


Fig 4.3.19 Secure Electronic Commerce Components

2. Merchants have their own certificates. A merchant who accepts a certain brand of card must be in possession of two certificates for two public keys owned by the merchant: one for signing messages, and one for key exchange. The merchant also needs a copy of the payment gateway's public-key certificate.

3. The customer places an order. This is a process that may involve the customer first browsing through the merchant's Web site to select items and determine the price. The customer then sends a list of the items to be purchased to the merchant, who returns an order form containing the list of items, their price, a total price, and an order number.

4. The merchant is verified. In addition to the order form, the merchant sends a copy of its certificate, so that the customer can verify that he or she is dealing with a valid store.

5. The order and payment are sent. The customer sends both order and payment information to the merchant, along with the customer's certificate. The order confirms the purchase of the items in the order form. The payment contains credit card details. The payment information is encrypted in such a way that it cannot be read by the merchant. The customer's certificate enables the merchant to verify the customer.

6. The merchant requests payment authorization. The merchant sends the payment information to the payment gateway, requesting authorization that the customer's available credit is sufficient for this purchase.

7. The merchant confirms the order. The merchant sends confirmation of the order to the customer.

8. The merchant provides the goods or service. The merchant ships the goods or provides the service to the customer.

9. The merchant requests payment. This request is sent to the payment gateway, which handles all of the payment processing.

Dual Signature

An important innovation introduced in SET is the dual signature. The purpose of the dual signature is to link two messages that are intended for two different recipients. In this case, the customer wants to send the order information (OI) to the merchant and the payment information (PI) to the bank. The merchant does not need to know the customer's credit card number, and the bank does not need to know the details of the customer's order. The customer is afforded extra protection in terms of privacy by keeping these two items separate. However, the two items must be linked in a way that can be used to resolve disputes if necessary. The link is needed so that the customer can prove that this payment is intended for this order and not for some other goods or service.

To see the need for the link, suppose that the customers send the merchant two messages: a signed OI and a signed PI, and the merchant passes the PI on to the bank. If the merchant can capture another OI

from this customer, the merchant could claim that this OI goes with the PI rather than the original OI. The linkage prevents this.

Figure 4.3.20 shows the use of a dual signature to meet the requirement of the preceding paragraph. The customer takes the hash (using SHA-1) of the PI and the hash of the OI. These two hashes are then concatenated and the hash of the result is taken. Finally, the customer encrypts the final hash with his or her private signature key, creating the dual signature. The operation can be summarized as

$$\mathbf{DS = E(PR_c, [H(H(PI)||H(OI)])}$$

where PR_c is the customer's private signature key. Now suppose that the merchant is in possession of the dual signature (DS), the OI, and the message digest for the PI (PIMD). The merchant also has the public key of the customer, taken from the customer's certificate. Then the merchant can compute the quantities

$$\mathbf{H(PIMS||H[OI]);\ D(PU_c, DS)}$$

where PU_c is the customer's public signature key. If these two quantities are equal, then the merchant has verified the signature. Similarly, if the bank is in possession of DS, PI, the message digest for OI (OIMD), and the customer's public key, then the bank can compute

$$\mathbf{H(H[OI]||OIMD);\ D(PU_c, DS)}$$

In summary,

1. The merchant has received OI and verified the signature.
2. The bank has received PI and verified the signature.
3. The customer has linked the OI and PI and can prove the linkage.

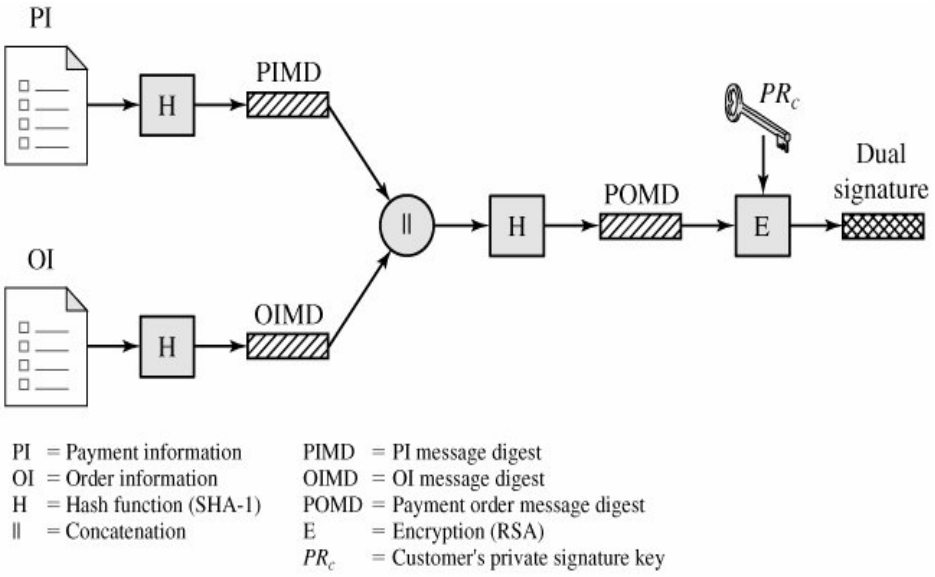


Fig 4.3.20 Construction of Dual Signature

For example, suppose the merchant wishes to substitute another OI in this transaction, to its advantage. It would then have to find another OI whose hash matches the existing OIMD. With SHA-1, this is deemed not to be feasible. Thus, the merchant cannot link another OI with this PI.

Payment Processing

Table 4.3.7 lists the transaction types supported by SET. We look in some detail at the following transactions:

- Purchase request
- Payment authorization
- Payment capture

Cardholder registration	Cardholders must register with a CA before they can send SET messages to merchants.
Merchant registration	Merchants must register with a CA before they can exchange SET messages with customers and payment gateways.
Purchase request	Message from customer to merchant containing OI for merchant and PI for bank.
Payment authorization	Exchange between merchant and payment gateway to authorize a given amount for a purchase on a given credit

Cardholder registration	Cardholders must register with a CA before they can send SET messages to merchants.
	card account.
Payment capture	Allows the merchant to request payment from the payment gateway.
Certificate inquiry and status	If the CA is unable to complete the processing of a certificate request quickly, it will send a reply to the cardholder or merchant indicating that the requester should check back later. The cardholder or merchant sends the Certificate Inquiry message to determine the status of the certificate request and to receive the certificate if the request has been approved.
Purchase inquiry	Allows the cardholder to check the status of the processing of an order after the purchase response has been received. Note that this message does not include information such as the status of back ordered goods, but does indicate the status of authorization, capture and credit processing.
Authorization reversal	Allows a merchant to correct previous authorization requests. If the order will not be completed, the merchant reverses the entire authorization. If part of the order will not be completed (such as when goods are back ordered), the merchant reverses part of the amount of the authorization.
Capture reversal	Allows a merchant to correct errors in capture requests such as transaction amounts that were entered incorrectly by a clerk.
Credit	Allows a merchant to issue a credit to a cardholder's account such as when goods are returned or were damaged during shipping. Note that the SET Credit message is always initiated by the merchant, not the cardholder. All communications between the cardholder and merchant that result in a credit being processed happen outside of SET.
Credit reversal	Allows a merchant to correct a previously request credit.
Payment gateway certificate request	Allows a merchant to query the payment gateway and receive a copy of the gateway's current key-exchange and signature certificates.
Batch administration	Allows a merchant to communicate information to the payment gateway regarding merchant batches.
Error message	Indicates that a responder rejects a message because it

Cardholder registration	Cardholders must register with a CA before they can send SET messages to merchants.
	fails format or content verification tests.

Table 4.3.7 SET Transaction Types

Purchase Request

Before the Purchase Request exchange begins, the cardholder has completed browsing, selecting, and ordering. The end of this preliminary phase occurs when the merchant sends a completed order form to the customer. All of the preceding occurs without the use of SET.

The purchase request exchange consists of four messages: Initiate Request, Initiate Response, Purchase Request, and Purchase Response.

In order to send SET messages to the merchant, the cardholder must have a copy of the certificates of the merchant and the payment gateway. The customer requests the certificates in the Initiate Request message, sent to the merchant. This message includes the brand of the credit card that the customer is using. The message also includes an ID assigned to this request/response pair by the customer and a nonce used to ensure timeliness.

The merchant generates a response and signs it with its private signature key. The response includes the nonce from the customer, another nonce for the customer to return in the next message, and a transaction ID for this purchase transaction. In addition to the signed response, the Initiate Response message includes the merchant's signature certificate and the payment gateway's key exchange certificate.

The cardholder verifies the merchant and gateway certificates by means of their respective CA signatures and then creates the OI and PI. The transaction ID assigned by the merchant is placed in both the OI and PI. The OI does not contain explicit order data such as the number and price of items. Rather, it contains an order reference generated in the exchange between merchant and customer during the shopping phase before the first SET message. Next, the cardholder prepares the Purchase Request message (Figure 4.3.21). For this purpose, the cardholder generates a one-time symmetric encryption key, K_s. The message includes the following:

1. **Purchase-related information.** This information will be forwarded to the payment gateway by the merchant and consists of
    - The PI

- The dual signature, calculated over the PI and OI, signed with the customer's private signature key
- The OI message digest (OIMD)

  The OIMD is needed for the payment gateway to verify the dual signature, as explained previously. All of these items are encrypted with K_s. The final item is
- The digital envelope. This is formed by encrypting K_s with the payment gateway's public key-exchange key. It is called a digital envelope because this envelope must be opened (decrypted) before the other items listed previously can be read.

The value of K_s is not made available to the merchant. Therefore, the merchant cannot read any of this payment-related information.

2. **Order-related information.** This information is needed by the merchant and consists of
   - The OI
   - The dual signature, calculated over the PI and OI, signed with the customer's private signature key
   - The PI message digest (PIMD)

The PIMD is needed for the merchant to verify the dual signature. Note that the OI is sent in the clear.

3. **Cardholder certificate.** This contains the cardholder's public signature key. It is needed by the merchant and by the payment gateway.

When the merchant receives the Purchase Request message, it performs the following actions (Figure 4.3.22):

1. Verifies the cardholder certificates by means of its CA signatures.
2. Verifies the dual signature using the customer's public signature key. This ensures that the order has not been tampered with in transit and that it was signed using the cardholder's private signature key.
3. Processes the order and forwards the payment information to the payment gateway for authorization (described later).
4. Sends a purchase response to the cardholder.

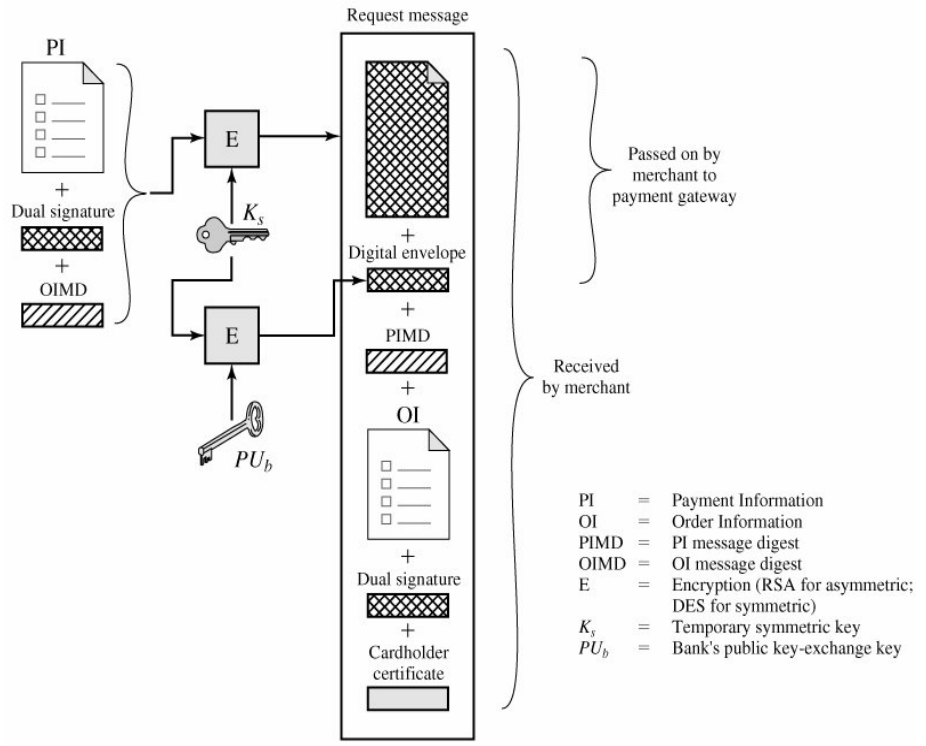


Fig 4.3.21 Cardholder Sends Purchase Request

The Purchase Response message includes a response block that acknowledges the order and references the corresponding transaction number. This block is signed by the merchant using its private signature key. The block and its signature are sent to the customer, along with the merchant's signature certificate.

When the cardholder software receives the purchase response message, it verifies the merchant's certificate and then verifies the signature on the response block. Finally, it takes some action based on the response, such as displaying a message to the user or updating a database with the status of the order.

Payment Authorization

During the processing of an order from a cardholder, the merchant authorizes the transaction with the payment gateway. The payment authorization ensures that the transaction was approved by the issuer. This authorization guarantees that the merchant will receive payment; the merchant can therefore provide the services or goods to the customer. The payment authorization exchange consists of two messages: Authorization Request and Authorization response.

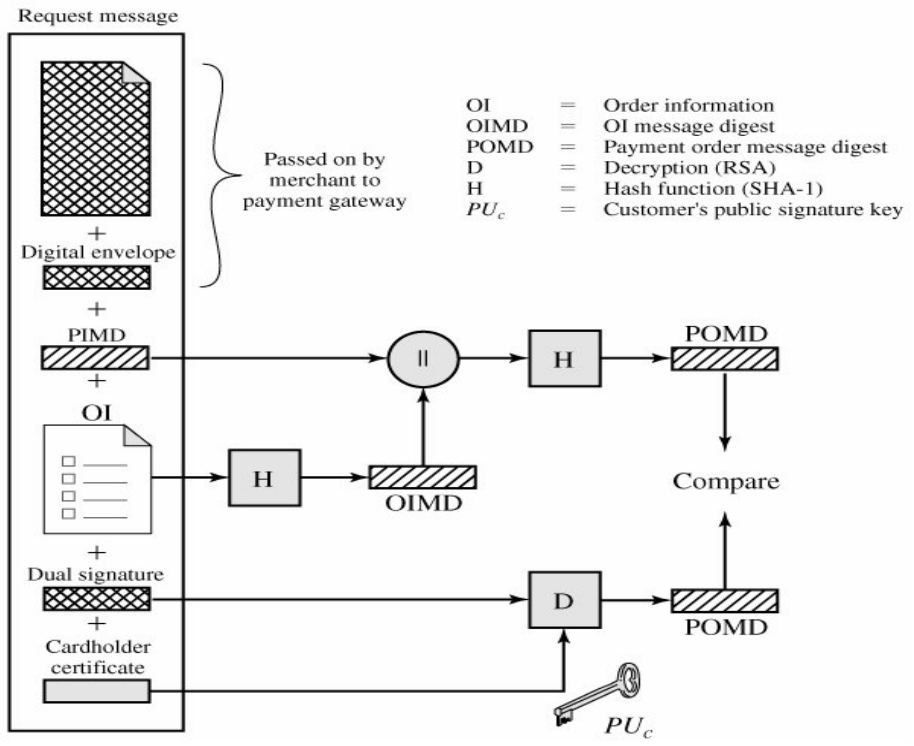


Fig 4.3.22. Merchant Verifies Customer Purchase Request

The merchant sends an Authorization Request message to the payment gateway consisting of the following:

1. **Purchase-related information.** This information was obtained from the customer and consists of
   - The PI
   - -The dual signature, calculated over the PI and OI, signed with the customer's private signature key
   - The OI message digest (OIMD)
   - The digital envelope
2. **Authorization-related information.** This information is generated by the merchant and consists of
   - An authorization block that includes the transaction ID, signed with the merchant's private signature key and encrypted with a one-time symmetric key generated by the merchant
   - A digital envelope. This is formed by encrypting the one-time key with the payment gateway's public key-exchange key.

3. **Certifcates.** The merchant includes the cardholder's signature key certificate (used to verify the dual signature), the merchant's signature key certificate (used to verify the merchant's signature), and the merchant's key-exchange certificate (needed in the payment gateway's response).

The payment gateway performs the following tasks:

1. Verifies all certificates
2. Decrypts the digital envelope of the authorization block to obtain the symmetric key and then decrypts the authorization block
3. Verifies the merchant's signature on the authorization block
4. Decrypts the digital envelope of the payment block to obtain the symmetric key and then decrypts the payment block
5. Verifies the dual signature on the payment block
6. Verifies that the transaction ID received from the merchant matches that in the PI received (indirectly) from the customer
7. Requests and receives an authorization from the issuer

Having obtained authorization from the issuer, the payment gateway returns an Authorization Response message to the merchant. It includes the following elements:

1. **Authorization-related information.** Includes an authorization block, signed with the gateway's private signature key and encrypted with a one-time symmetric key generated by the gateway. Also includes a digital envelope that contains the one-time key encrypted with the merchants public key-exchange key.
2. **Capture token information.** This information will be used to effect payment later. This block is of the same form as (1), namely, a signed, encrypted capture token together with a digital envelope. This token is not processed by the merchant. Rather, it must be returned, as is, with a payment request.
3. **Certificate.** The gateway's signature key certificate.

With the authorization from the gateway, the merchant can provide the goods or service to the customer.

Payment Capture

To obtain payment, the merchant engages the payment gateway in a payment capture transaction, consisting of a capture request and a capture response message.

For the Capture Request message, the merchant generates, signs, and encrypts a capture request block, which includes the payment amount and the transaction ID. The message also includes the encrypted capture token received earlier (in the Authorization Response) for this transaction, as well as the merchant's signature key and key-exchange key certificates.

When the payment gateway receives the capture request message, it decrypts and verifies the capture request block and decrypts and verifies the capture token block. It then checks for consistency between the capture request and capture token. It then creates a clearing request that is sent to the issuer over the private payment network. This request causes funds to be transferred to the merchant's account.

The gateway then notifies the merchant of payment in a Capture Response message. The message includes a capture response block that the gateway signs and encrypts. The message also includes the gateway's signature key certificate. The merchant software stores the capture response to be used for reconciliation with payment received from the acquirer.

REFERENCES

1. William Stallings, Cryptography and Network Security, PHI Publishers

2. www.wikipedia.org

Review Questions:

7. Write a short note on SSL Protocol stack
8. Explain SSL Handshake protocol in detail
9. Expalin SSL Record Protocol in detail
10. Write a short note on SET.
11. Explain the payment processing in SET

5. SYSTEM SECURITY

OBJECTIVE

This lesson examines Virtual Private Networks and a variety of information access and service threats presented by hackers that exploit vulnerabilities in network-based computing systems. It also focuses on the software threats to systems, with a special emphasis on viruses and worms and then looks at countermeasures.

VIRTUAL PRIVATE NETWORKS

A virtual private network (VPN) is a computer network in which some of the links between nodes are carried by open connections or virtual circuits in some larger network (e.g., the Internet) instead of by physical wires. The link-layer protocols of the virtual network are said to be tunneled through the larger network when this is the case. One common application is secure communications through the public Internet, but a VPN need not have explicit security features, such as authentication or content encryption. VPNs, for example, can be used to separate the traffic of different user communities over an underlying network with strong security features.

VPNs need to provide the following four critical functions to ensure security for

data:

- **authentication**—ensuring that the data originates at the source that it claims
- **access control**—restricting unauthorized users from gaining admission to the network
- **confidentiality**—preventing anyone from reading or copying data as it travels across the Internet
- **data integrity**—ensuring that no one tampers with data as it travels across the Internet

A virtual private network (VPN) is the extension of a private network that encompasses links across shared or public networks like the Internet. A VPN enables you to send data between two computers across a shared or public internetwork in a manner that emulates the properties of a point-

to-point private link. The act of configuring and creating a virtual private network is known as virtual private networking.

To emulate a point-to-point link, data is encapsulated, or wrapped, with a header that provides routing information allowing it to traverse the shared or public transit internetwork to reach its endpoint. To emulate a private link, the data being sent is encrypted for confidentiality. Packets that are intercepted on the shared or public network are indecipherable without the encryption keys. The portion of the connection in which the private data is encapsulated is known as the tunnel. The portion of the connection in which the private data is encrypted is known as the virtual private network (VPN) connection.

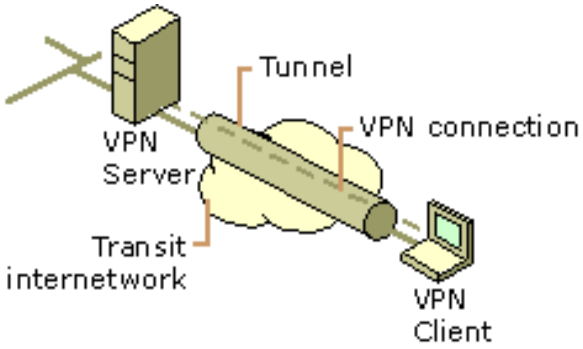


Figure 4.4.1: Virtual private network connection

VPN connections allow users working at home or on the road to connect in a secure fashion to a remote corporate server using the routing infrastructure provided by a public internetwork (such as the Internet). From the user's perspective, the VPN connection is a point-to-point connection between the user's computer and a corporate server. The nature of the intermediate internetwork is irrelevant to the user because it appears as if the data is being sent over a dedicated private link.

VPN technology also allows a corporation to connect to branch offices or to other companies over a public internetwork (such as the Internet), while maintaining secure communications. The VPN connection across the Internet logically operates as a wide area network (WAN) link between the sites.

In both of these cases, the secure connection across the internetwork appears to the user as a private network communication—despite the fact that this communication occurs over a public internetwork—hence the name virtual private network.

VPN technology is designed to address issues surrounding the current business trend toward increased telecommuting and widely distributed global operations, where workers must be able to connect to central resources and must be able to communicate with each other.

To provide employees with the ability to connect to corporate computing resources, regardless of their location, a corporation must deploy a scalable remote access solution. Typically, corporations choose either an MIS department solution, where an internal information systems department is charged with buying, installing, and maintaining corporate modem pools and a private network infrastructure; or they choose a value-added network (VAN) solution, where they pay an outsourced company to buy, install, and maintain modem pools and a telecommunication infrastructure.

Neither of these solutions provides the necessary scalability, in terms of cost, flexible administration, and demand for connections. Therefore, it makes sense to replace the modem pools and private network infrastructure with a less expensive solution based on Internet technology so that the business can focus on its core competencies. With an Internet solution, a few Internet connections through Internet service providers (ISPs) and VPN server computers can serve the remote networking needs of hundreds or thousands of remote clients and branch offices.

Common Uses of VPNs

The next few subsections describe the more common VPN configurations in more detail.

Remote Access Over the Internet

VPNs provide remote access to corporate resources over the public Internet, while maintaining privacy of information. Figure 4.4.2 shows a VPN connection used to connect a remote user to a corporate intranet.

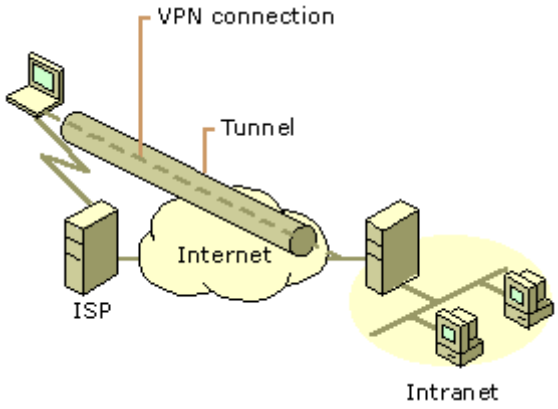


Figure 4.4.2: Using a VPN connection to connect a remote client to a private intranet

Rather than making a long distance (or 1-800) call to a corporate or outsourced network access server (NAS), the user calls a local ISP. Using the connection to the local ISP, the VPN software creates a virtual private network between the dial-up user and the corporate VPN server across the Internet.

Connecting Networks Over the Internet

There are two methods for using VPNs to connect local area networks at remote sites:

- **Using dedicated lines to connect a branch office to a corporate LAN**. Rather than using an expensive long-haul dedicated circuit between the branch office and the corporate hub, both the branch office and the corporate hub routers can use a local dedicated circuit and local ISP to connect to the Internet. The VPN software uses the local ISP connections and the Internet to create a virtual private network between the branch office router and corporate hub router.

- **Using a dial-up line to connect a branch office to a corporate LAN**. Rather than having a router at the branch office make a long distance (or 1-800) call to a corporate or outsourced NAS, the router at the branch office can call the local ISP. The VPN software uses the connection to the local ISP to create a VPN between the branch office router and the corporate hub router across the Internet.

In both cases, the facilities that connect the branch office and corporate offices to the Internet are local. The corporate hub router that acts as a VPN server must be connected to a local ISP with a dedicated line. This VPN server must be listening 24 hours a day for incoming VPN traffic.

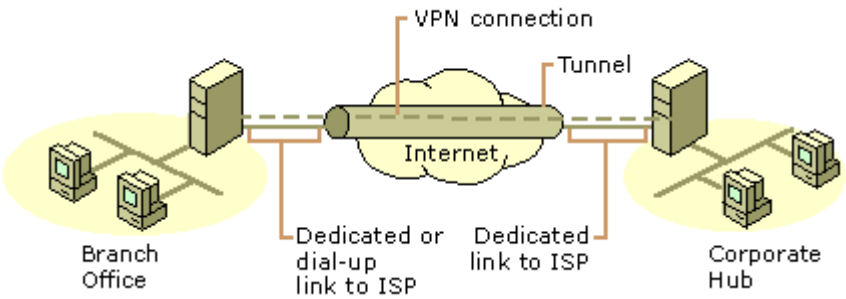


Figure 4.4.3: Using a VPN connection to connect two remote sites

Connecting Computers over an Intranet

In some corporate internetworks, the departmental data is so sensitive that the department's LAN is physically disconnected from the rest of the corporate internetwork. Although this protects the department's confidential information, it creates information accessibility problems for those users not physically connected to the separate LAN.

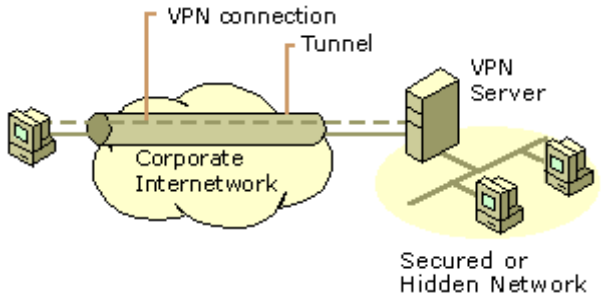


Figure 4.4.4: Using a VPN connection to connect to a secured or hidden network

VPNs allow the department's LAN to be physically connected to the corporate internetwork but separated by a VPN server. The VPN server is not acting as a router between the corporate internetwork and the department LAN. A router would connect the two networks, allowing everyone access to the sensitive LAN. By using a VPN, the network administrator can ensure that only those users on the corporate internetwork who have appropriate credentials (based on a need-to-know policy within the company) can establish a VPN with the VPN server and gain access to the protected resources of the department. Additionally, all communication across the VPN can be encrypted for data confidentiality. Those users who do not have the proper credentials cannot view the department LAN.

Basic VPN Requirements

Typically, when deploying a remote networking solution, an enterprise needs to facilitate controlled access to corporate resources and information. The solution must allow roaming or remote clients to connect to LAN resources, and the solution must allow remote offices to connect to each other to share resources and information (router-to-router connections). In addition, the solution must ensure the privacy and integrity of data as it traverses the Internet. The same concerns apply in the case of sensitive data traversing a corporate internetwork. Therefore, a VPN solution should provide at least all of the following:

- **User Authentication**. The solution must verify the VPN client's identity and restrict VPN access to authorized users only. It must also provide audit and accounting records to show who accessed what information and when.

- **Address Management**. The solution must assign a VPN client's address on the intranet and ensure that private addresses are kept private.

- **Data Encryption**. Data carried on the public network must be rendered unreadable to unauthorized clients on the network.

- **Key Management**. The solution must generate and refresh encryption keys for the client and the server.

- **Multiprotocol Support**. The solution must handle common protocols used in the public network. These include IP, Internetwork Packet Exchange (IPX), and so on.

An Internet VPN solution based on the Point-to-Point Tunneling Protocol (PPTP) or Layer Two Tunneling Protocol (L2TP) meets all of these basic requirements and takes advantage of the broad availability of the Internet. Other solutions, including Internet Protocol Security (IPSec), meet only some of these requirements, but remain useful for specific situations.

Tunneling Basics

Tunneling is a method of using an internetwork infrastructure to transfer data for one network over another network. The data to be transferred (or payload) can be the frames (or packets) of another protocol. Instead of sending a frame as it is produced by the originating node, the tunneling protocol encapsulates the frame in an additional header. The additional header provides routing information so that the encapsulated payload can traverse the intermediate internetwork.

The encapsulated packets are then routed between tunnel endpoints over the internetwork. The logical path through which the encapsulated packets travel through the internetwork is called a tunnel. Once the encapsulated frames reach their destination on the internetwork, the frame is decapsulated and forwarded to its final destination. Tunneling includes this entire process (encapsulation, transmission, and decapsulation of packets).

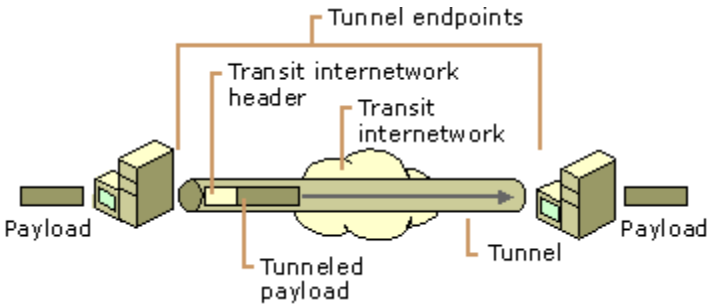


Figure 4.4.5: Tunneling

The transit internetwork can be any internetwork-the Internet is a public internetwork and is the most widely known real world example. There are many examples of tunnels that are carried over corporate internetworks. And while the Internet provides one of the most pervasive and cost-effective internetworks, references to the Internet in this paper can be

replaced by any other public or private internetwork that acts as a transit internetwork.

Tunneling technologies have been in existence for some time. Some examples of mature technologies include:

- **SNA tunneling over IP internetworks**. When System Network Architecture (SNA) traffic is sent across a corporate IP internetwork, the SNA frame is encapsulated in a UDP and IP header.

- **IPX tunneling for Novell NetWare over IP internetworks**. When an IPX packet is sent to a NetWare server or IPX router, the server or the router wraps the IPX packet in a UDP and IP header, and then sends it across an IP internetwork. The destination IP-to-IPX router removes the UDP and IP header and forwards the packet to the IPX destination.

New tunneling technologies have been introduced in recent years. These newer technologies-which are the primary focus of this paper-include:

- **Point-to-Point Tunneling Protocol (PPTP)**. PPTP allows IP, IPX, or NetBEUI traffic to be encrypted, and then encapsulated in an IP header to be sent across a corporate IP internetwork or a public IP internetwork such as the Internet.

- **Layer Two Tunneling Protocol (L2TP)**. L2TP allows IP, IPX, or NetBEUI traffic to be encrypted, and then sent over any medium that supports point-to-point datagram delivery, such as IP, X.25, Frame Relay, or ATM.

- **IPSec tunnel mode**. IPSec tunnel mode allows IP packets to be encrypted, and then encapsulated in an IP header to be sent across a corporate IP internetwork or a public IP internetwork such as the Internet.

Tunneling Protocols

For a tunnel to be established, both the tunnel client and the tunnel server must be using the same *tunneling protocol.*

Tunneling technology can be based on either a Layer 2 or a Layer 3 tunneling protocol. These layers correspond to the Open Systems Interconnection (OSI) Reference Model. Layer 2 protocols correspond to the data-link layer and use frames as their unit of exchange. PPTP and L2TP are Layer 2 tunneling protocols; both encapsulate the payload in a PPP frame to be sent across an internetwork. Layer 3 protocols correspond to the Network layer, and use packets. IPSec tunnel mode is an example of a Layer 3 tunneling protocol and encapsulate IP packets in an additional IP header before sending them across an IP internetwork.
How Tunneling Works

For Layer 2 tunneling technologies, such as PPTP and L2TP, a tunnel is similar to a session; both of the tunnel endpoints must agree to the tunnel and must negotiate configuration variables, such as address assignment or encryption or compression parameters. In most cases, data transferred across the tunnel is sent using a datagram-based protocol. A tunnel maintenance protocol is used as the mechanism to manage the tunnel.

Layer 3 tunneling technologies generally assume that all of the configuration issues are preconfigured, often by manual processes. For these protocols, there may be no tunnel maintenance phase. For Layer 2 protocols (PPTP and L2TP), however, a tunnel must be created, maintained, and then terminated.

Once the tunnel is established, tunneled data can be sent. The tunnel client or server uses a tunnel data transfer protocol to prepare the data for transfer. For example, when the tunnel client sends a payload to the tunnel server, the tunnel client first appends a tunnel data transfer protocol header to the payload. The client then sends the resulting encapsulated payload across the internetwork, which routes it to the tunnel server. The tunnel server accepts the packets, removes the tunnel data transfer protocol header, and forwards the payload to the target network. Information sent between the tunnel server and the tunnel client behaves similarly.

Tunneling Protocols and the Basic Tunneling Requirements

Because they are based on the well-defined PPP protocol, Layer 2 protocols (such as PPTP and L2TP) inherit a suite of useful features. These features, and their Layer 3 counterparts address the basic VPN requirements, as outlined below.

- **User Authentication**. Layer 2 tunneling protocols inherit the user authentication schemes of PPP, including the EAP methods discussed below. Many Layer 3 tunneling schemes assume that the endpoints were well known (and authenticated) before the tunnel was established. An exception to this is IPSec Internet Key Exchange (IKE) negotiation, which provides mutual authentication of the tunnel endpoints. Most IPSec implementations including Windows 2000 support computer-based certificates only, rather than user certificates. As a result, any user with access to one of the endpoint computers can use the tunnel. This potential security weakness can be eliminated when IPSec is paired with a Layer 2 protocol such as L2TP.

- **Token card support**. Using the Extensible Authentication Protocol (EAP), Layer 2 tunneling protocols can support a wide variety of authentication methods, including one-time passwords, cryptographic calculators, and smart cards. Layer 3 tunneling protocols can use similar methods; for example, IPSec defines public key certificate authentication in its IKE negotiation.

- **Dynamic address assignment**. Layer 2 tunneling supports dynamic assignment of client addresses based on the Network Control Protocol (NCP) negotiation mechanism. Generally, Layer 3 tunneling schemes assume that an address has already been assigned prior to initiation of the tunnel. Schemes for assignment of addresses in IPSec tunnel mode are currently under development and are not yet available.

- **Data compression**. Layer 2 tunneling protocols support PPP-based compression schemes. For example, the Microsoft implementations of both PPTP and L2TP use Microsoft Point-to-Point Compression (MPPC). The IETF is investigating similar mechanisms (such as IP Compression) for the Layer 3 tunneling protocols.

- **Data encryption**. Layer 2 tunneling protocols support PPP-based data encryption mechanisms. The Microsoft implementation of PPTP supports optional use of Microsoft Point-to-Point Encryption (MPPE), based on the RSA/RC4 algorithm. Layer 3 tunneling protocols can use similar methods; for example, IPSec defines several optional data encryption methods, which are negotiated during the IKE exchange. The Microsoft implementation of the L2TP protocol uses IPSec encryption to protect the data stream from the VPN client to the VPN server.

- **Key Management**. MPPE, a Layer 2 encryption mechanism, relies on the initial key generated during user authentication, and then refreshes it periodically. IPSec explicitly negotiates a common key during the IKE exchange, and also refreshes it periodically.

- **Multiprotocol support**. Layer 2 tunneling supports multiple payload protocols, which makes it easy for tunneling clients to access their corporate networks using IP, IPX, NetBEUI, and so on. In contrast, Layer 3 tunneling protocols, such as IPSec tunnel mode, typically support only target networks that use the IP protocol.

Point-to-Point Protocol (PPP)

Because the Layer 2 protocols depend heavily on the features originally specified for PPP, it is worth examining this protocol more closely. PPP was designed to send data across dial-up or dedicated point-to-point connections. PPP encapsulates IP, IPX, and NetBEUI packets within PPP frames, and then transmits the PPP-encapsulated packets across a point-to-point link. PPP is used between a dial-up client and an NAS.

There are four distinct phases of negotiation in a PPP dial-up session. Each of these four phases must complete successfully before the PPP connection is ready to transfer user data.

Phase 1: PPP Link Establishment

PPP uses Link Control Protocol (LCP) to establish, maintain, and end the physical connection. During the initial LCP phase, basic communication options are selected. During the link establishment phase (Phase 1), authentication protocols are selected, but they are not actually implemented until the connection authentication phase (Phase 2). Similarly, during LCP a decision is made as to whether the two peers will negotiate the use of compression and/or encryption. The actual choice of compression and encryption algorithms and other details occurs during Phase 4.

Phase 2: User Authentication

In the second phase, the client PC presents the user's credentials to the remote access server. A secure authentication scheme provides protection against replay attacks and remote client impersonation. A replay attack occurs when a third party monitors a successful connection and uses captured packets to play back the remote client's response so that it can gain an authenticated connection. Remote client impersonation occurs when a third party takes over an authenticated connection. The intruder waits until the connection has been authenticated, and then traps the conversation parameters, disconnects the authenticated user, and takes control of the authenticated connection.

Most implementations of PPP provide limited authentication methods, typically Password Authentication Protocol (PAP), Challenge Handshake Authentication Protocol (CHAP), and Microsoft Challenge Handshake Authentication Protocol (MS-CHAP).

- **Password Authentication Protocol (PAP)**. PAP is a simple, clear-text authentication scheme. The NAS requests the user name and password, and PAP returns them in clear text (unencrypted). Obviously, this authentication scheme is not secure because a third party could capture the user's name and password and use it to get subsequent access to the NAS and all of the resources provided by the NAS. PAP provides no protection against replay attacks or remote client impersonation once the user's password is compromised.

- **Challenge-Handshake Authentication Protocol (CHAP)**. CHAP is an encrypted authentication mechanism that avoids transmission of the actual password on the connection. The NAS sends a challenge, which consists of a session ID and an arbitrary challenge string, to the remote client. The remote client must use the MD5 one-way hashing algorithm to return the user name and an encryption of the challenge, session ID, and the client's password. The user name is sent unhashed. CHAP is an improvement over PAP because the clear-text password is not sent over the link. Instead, the password is used

to create an encrypted hash from the original challenge. The server knows the client's clear-text password and can, therefore, replicate the operation and compare the result to the password sent in the client's response. CHAP protects against replay attacks by using an arbitrary challenge string for each authentication attempt. CHAP protects against remote client impersonation by unpredictably sending repeated challenges to the remote client throughout the duration of the connection.

During phase 2 of PPP link configuration, the NAS collects the authentication data, and then validates the data against its own user database or a central authentication database server, such as one maintained by a Windows domain controller, or the authentication data is sent to a Remote Authentication Dial-in User Service (RADIUS) server.

Phase 3: PPP Callback Control

The Microsoft implementation of PPP includes an optional callback control phase. This phase uses the Callback Control Protocol (CBCP) immediately after the authentication phase. If configured for callback, both the remote client and NAS disconnect after authentication. The NAS then calls the remote client back at a specified phone number. This provides an additional level of security to dial-up networking. The NAS allows connections from remote clients physically residing at specific phone numbers only.

Phase 4: Invoking Network Layer Protocol(s)

Once the previous phases have been completed, PPP invokes the various network control protocols (NCPs) that were selected during the link establishment phase (Phase 1) to configure protocols used by the remote client. For example, during this phase the IP control protocol (IPCP) can assign a dynamic address to the dial-in user. In the Microsoft implementation of PPP, the compression control protocol is used to negotiate both data compression (using MPPC) and data encryption (using MPPE) for because both are implemented in the same routine.
Data-Transfer Phase

Once the four phases of negotiation have been completed, PPP begins to forward data to and from the two peers. Each transmitted data packet is wrapped in a PPP header which is removed by the receiving system. If data compression was selected in phase 1 and negotiated in phase 4, data is compressed before transmission. If data encryption is selected and negotiated, data is encrypted before transmission.

Point-to-Point Tunneling Protocol (PPTP)

PPTP is a Layer 2 protocol that encapsulates PPP frames in IP datagrams for transmission over an IP internetwork, such as the Internet. PPTP can be used for remote access and router-to-router VPN connections. PPTP is documented in RFC 2637.

The Point-to-Point Tunneling Protocol (PPTP) uses a TCP connection for tunnel maintenance and a modified version of Generic Routing Encapsulation (GRE) to encapsulate PPP frames for tunneled data. The payloads of the encapsulated PPP frames can be encrypted and/or compressed. Figure 4.4.6 shows the structure of a PPTP packet containing user data.

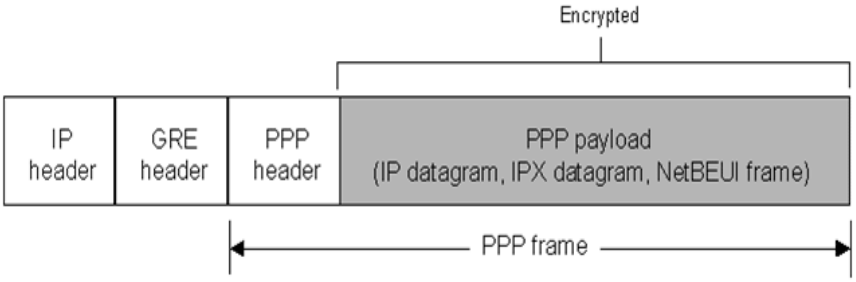


Figure 4.4.6: Structure of a PPTP packet containing user data

Layer Two Tunneling Protocol (L2TP)

L2TP is a combination of PPTP and Layer 2 Forwarding (L2F), a technology proposed by Cisco Systems, Inc. L2TP represents the best features of PPTP and L2F. L2TP encapsulates PPP frames to be sent over IP, X.25, Frame Relay, or Asynchronous Transfer Mode (ATM) networks. When configured to use IP as its datagram transport, L2TP can be used as a tunneling protocol over the Internet. L2TP is documented in RFC 2661.

L2TP over IP internetworks uses UDP and a series of L2TP messages for tunnel maintenance. L2TP also uses UDP to send L2TP-encapsulated PPP frames as the tunneled data. The payloads of encapsulated PPP frames can be encrypted and/or compressed. Figure 4.4.7 shows the structure of an L2TP packet containing user data.

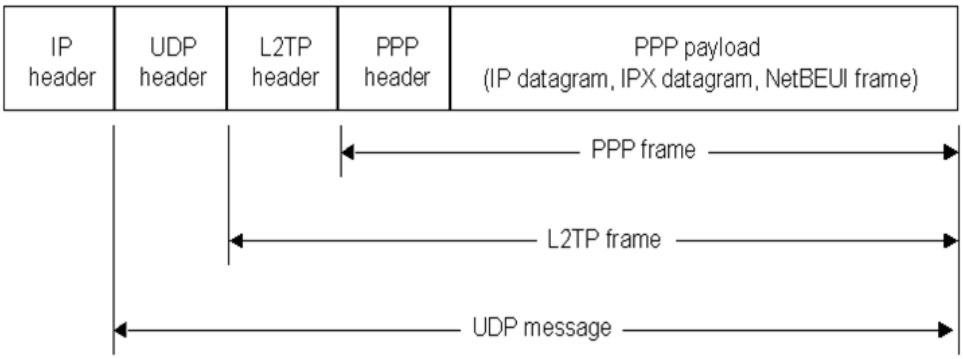


Figure 4.4.7: Structure of an L2TP packet containing user data

In Windows 2000, IPSec Encapsulating Security Payload (ESP) is used to encrypt the L2TP packet. This is known as L2TP/IPSec. The result after applying ESP is shown in Figure 4.4.8.

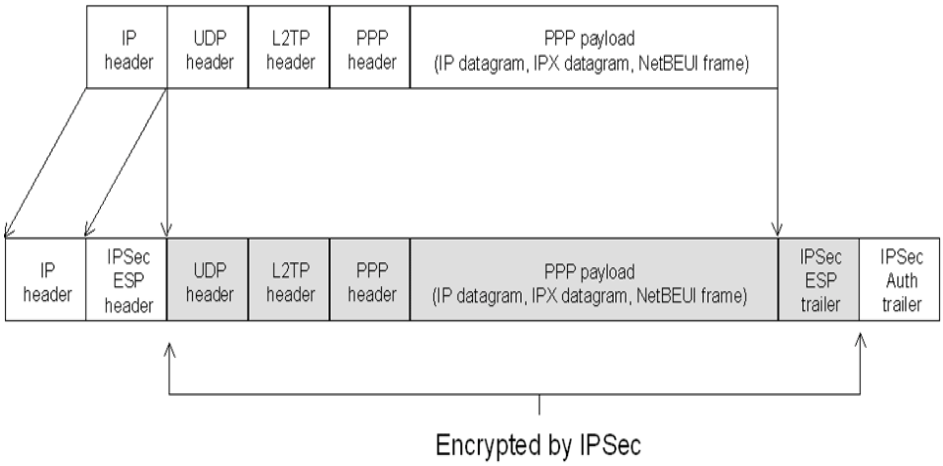


Figure 4.4.8: Encryption of an L2TP packet with IPSec ESP

PPTP Compared to L2TP/IPSec

Both PPTP and L2TP/IPSec use PPP to provide an initial envelope for the data, and then append additional headers for transport through the internetwork. However, there are the following differences:

- With PPTP, data encryption begins after the PPP connection process (and, therefore, PPP authentication) is completed. With L2TP/IPSec, data encryption begins before the PPP connection process by negotiating an IPSec security association.

- PPTP connections use MPPE, a stream cipher that is based on the Rivest-Shamir-Aldeman (RSA) RC-4 encryption algorithm and uses 40, 56, or 128-bit encryption keys. Stream ciphers encrypt data as a bit stream. L2TP/IPSec connections use the Data Encryption Standard (DES), which is a block cipher that uses either a 56-bit key for DES or three 56-bit keys for 3-DES. Block ciphers encrypt data in discrete blocks (64-bit blocks, in the case of DES).

- PPTP connections require only user-level authentication through a PPP-based authentication protocol. L2TP/IPSec connections require the same user-level authentication and, in addition, computer-level authentication using computer certificates.

Advantages of L2TP/IPSec over PPTP

The following are the advantages of using L2TP/IPSec over PPTP in Windows 2000:

- IPSec provides per packet data authentication (proof that the data was sent by the authorized user), data integrity (proof that the data was not modified in transit), replay protection (prevention from resending a stream of captured packets), and data confidentiality (prevention from interpreting captured packets without the encryption key). By contrast, PPTP provides only per-packet data confidentiality.

- L2TP/IPSec connections provide stronger authentication by requiring both computer-level authentication through certificates and user-level authentication through a PPP authentication protocol.

- PPP packets exchanged during user-level authentication are never sent in an unencrypted form because the PPP connection process for L2TP/IPSec occurs after the IPSec security associations (SAs) are established. If intercepted, the PPP authentication exchange for some types of PPP authentication protocols can be used to perform offline dictionary attacks and determine user passwords. By encrypting the PPP authentication exchange, offline dictionary attacks are only possible after the encrypted packets have been successfully decrypted.

Advantages of PPTP over L2TP/IPSec

The following are advantages of PPTP over L2TP/IPSec in Windows 2000:

- PPTP does not require a certificate infrastructure. L2TP/IPSec requires a certificate infrastructure for issuing computer certificates to the VPN server computer (or other authenticating server) and all VPN client computers.

- PPTP can be used by computers running Windows XP, Windows 2000, Windows NT version 4.0, Windows Millennium Edition (ME), Windows 98, and Windows 95 with the Windows Dial-Up Networking 1.3 Performance & Security Update. L2TP/IPSec can only be used with Windows XP and Windows 2000 VPN clients. Only these clients support the L2TP protocol, IPSec, and the use of certificates.

- PPTP clients and server can be placed behind a network address translator (NAT) if the NAT has the appropriate editors for PPTP traffic. L2TP/IPSec-based VPN clients or servers cannot be

placed behind a NATunless both support IPSec NAT Traversal (NAT-T). IPSec NAT-T is supported by Windows Server 2003, Microsoft L2TP/IPSec VPN Client, and for VPN clients with L2TP/IPSec NAT-T Update for Windows XP and Windows 2000.

Accounting, Auditing, and Alarming

To properly administer a VPN system, network administrators should be able to track who uses the system, how many connections are made, unusual activity, error conditions, and situations that may indicate equipment failure. This information can be used for billing, auditing, and alarm or error-notification purposes.

For example, an administrator may need to know who connected to the system and for how long in order to construct billing data. Unusual activity may indicate a misuse of the system or inadequate system resources. Real-time monitoring of equipment (for example, unusually high activity on one modem and inactivity on another) may generate alerts to notify the administrator of a modem failure. The tunnel server should provide all of this information, and the system should provide event logs, reports, and a data storage facility to handle the data appropriately.

The RADIUS protocol defines a suite of call-accounting requests that are independent from the authentication requests discussed above. These messages from the NAS to the RADIUS server request the latter to generate accounting records at the start of a call, the end of a call, and at predetermined intervals during a call. The Routing and Remote Access service can be configured to generate these RADIUS accounting requests separately from connection requests (which could go to the domain controller or to a RADIUS server). This allows an administrator to configure an accounting RADIUS server, whether RADIUS is used for authentication or not. An accounting server can then collect records for every VPN connection for later analysis. A number of third-parties have already written billing and audit packages that read these RADIUS accounting records and produce various useful reports.

VPNs allow users or corporations to connect to remote servers, branch offices, or to other companies over a public internetwork, while maintaining secure communications. In all of these cases, the secure connection appears to the user as a private network communication-despite the fact that this communication occurs over a public internetwork. VPN technology is designed to address issues surrounding the current business trend toward increased telecommuting and widely distributed global operations, where workers must be able to connect to central resources and communicate with each other.

MACLICIOUS SOFTWARES

Malware is a general term for a piece of software inserted into an information system to cause harm to that system or other systems, or to subvert them for use other than that intended by their owners.[6] Malware

can gain remote access to an information system, record and send data from that system to a third party without the user‟s permission or knowledge, conceal that the information system has been compromised, disable security measures, damage the information system, or otherwise affect the data and system integrity.

Infectious malware: viruses and worms

The best-known types of malware, *viruses* and *worms*, are known for the manner in which they spread, rather than any other particular behavior. The rest of the malwares are tabulated in table 4.4.1.The term *computer virus* is used for a program which has infected some executable software and which causes that software, *when run*, to spread the virus to other executable software. Viruses may also contain a payload which performs other actions, often malicious. A *worm*, on the other hand, is a program which actively transmits itself over a network to infect other computers. It too may carry a payload.

These definitions lead to the observation that a virus requires user intervention to spread, whereas a worm spreads automatically. Using this distinction, infections transmitted by email or Microsoft Word documents, which rely on the recipient opening a file or email to infect the system, would be classified as viruses rather than worms.

Name	Description
Virus	Attaches itself to a program and propagates copies of itself to other programs
Worm	Program that propagates copies of itself to other computers
Logic bomb	Triggers action when condition occurs
Trojan horse	Program that contains unexpected additional functionality
Backdoor (trapdoor)	Program modification that allows unauthorized access to functionality
Exploits	Code specific to a single vulnerability or set of vulnerabilities
Downloaders	Program that installs other items on a machine that is under attack. Usually, a downloader is sent in an e-mail.
Auto-rooter	Malicious hacker tools used to break into new machines remotely

Name	Description
Kit (virus generator)	Set of tools for generating new viruses automatically
Spammer programs	Used to send large volumes of unwanted e-mail
Flooders	Used to attack networked computer systems with a large volume of traffic to carry out a denial of service (DoS) attack
Keyloggers	Captures keystrokes on a compromised system
Rootkit	Set of hacker tools used after attacker has broken into a computer system and gained root-level access
Zombie	Program activated on an infected machine that is activated to launch attacks on other machines

Table 4.4.1 Terminology of Malicious Programs

Malicious software can be divided into two categories: those that need a host program, and those that are independent. The former are essentially fragments of programs that cannot exist independently of some actual application program, utility, or system program. Viruses, logic bombs, and backdoors are examples. The latter are self-contained programs that can be scheduled and run by the operating system. Worms and zombie programs are examples.

Backdoor

A backdoor, also known as a trapdoor, is a secret entry point into a program that allows someone that is aware of the backdoor to gain access without going through the usual security access procedures. Programmers have used backdoors legitimately for many years to debug and test programs. This usually is done when the programmer is developing an application that has an authentication procedure, or a long setup, requiring the user to enter many different values to run the application. To debug the program, the developer may wish to gain special privileges or to avoid all the necessary setup and authentication. The programmer may also want to ensure that there is a method of activating the program should something be wrong with the authentication procedure that is being built into the application. The backdoor is code that recognizes some special sequence of input or is triggered by being run from a certain user ID or by an unlikely sequence of events.

Backdoors become threats when unscrupulous programmers use them to gain unauthorized access. The backdoor was the basic idea for the vulnerability portrayed in the movie War Games. Another example is that during the development of Multics, penetration tests were conducted by an Air Force "tiger team" (simulating adversaries). One tactic employed was to send a bogus operating system update to a site running Multics. The update contained a Trojan horse (described later) that could be activated by a backdoor and that allowed the tiger team to gain access. The threat was so well implemented that the Multics developers could not find it, even after they were informed of its presence.

It is difficult to implement operating system controls for backdoors. Security measures must focus on the program development and software update activities.

Logic Bomb

One of the oldest types of program threat, predating viruses and worms, is the logic bomb. The logic bomb is code embedded in some legitimate program that is set to "explode" when certain conditions are met. Examples of conditions that can be used as triggers for a logic bomb are the presence or absence of certain files, a particular day of the week or date, or a particular user running the application. Once triggered, a bomb may alter or delete data or entire files, cause a machine halt, or do some other damage. A striking example of how logic bombs can be employed was the case of Tim Lloyd, who was convicted of setting a logic bomb that cost his employer, Omega Engineering, more than $10 million, derailed its corporate growth strategy, and eventually led to the layoff of 80 workers. Ultimately, Lloyd was sentenced to 41 months in prison and ordered to pay $2 million in restitution.

Trojan Horses

A Trojan horse is a useful, or apparently useful, program or command procedure containing hidden code that, when invoked, performs some unwanted or harmful function.

Trojan horse programs can be used to accomplish functions indirectly that an unauthorized user could not accomplish directly. For example, to gain access to the files of another user on a shared system, a user could create a Trojan horse program that, when executed, changed the invoking user's file permissions so that the files are readable by any user. The author could then induce users to run the program by placing it in a common directory and naming it such that it appears to be a useful utility. An example is a program that ostensibly produces a listing of the user's files in a desirable format. After another user has run the program, the author can then access the information in the user's files. An example of a Trojan horse program that would be difficult to detect is a compiler that has been modified to insert additional code into certain programs as they are compiled, such as a system login program. The code creates a backdoor in the login program that permits the author to log on to the

system using a special password. This Trojan horse can never be discovered by reading the source code of the login program.

Another common motivation for the Trojan horse is data destruction. The program appears to be performing a useful function (e.g., a calculator program), but it may also be quietly deleting the user's files. For example, a CBS executive was victimized by a Trojan horse that destroyed all information contained in his computer's memory. The Trojan horse was implanted in a graphics routine offered on an electronic bulletin board system.

Zombie

A zombie is a program that secretly takes over another Internet-attached computer and then uses that computer to launch attacks that are difficult to trace to the zombie's creator. Zombies are used in denial-of-service attacks, typically against targeted Web sites. The zombie is planted on hundreds of computers belonging to unsuspecting third parties, and then used to overwhelm the target Web site by launching an overwhelming onslaught of Internet traffic.

The Nature of Viruses

A virus is a piece of software that can "infect" other programs by modifying them; the modification includes a copy of the virus program, which can then go on to infect other programs.

Biological viruses are tiny scraps of genetic codeDNA or RNAthat can take over the machinery of a living cell and trick it into making thousands of flawless replicas of the original virus. Like its biological counterpart, a computer virus carries in its instructional code the recipe for making perfect copies of itself. The typical virus becomes embedded in a program on a computer. Then, whenever the infected computer comes into contact with an uninfected piece of software, a fresh copy of the virus passes into the new program. Thus, the infection can be spread from computer to computer by unsuspecting users who either swap disks or send programs to one another over a network. In a network environment, the ability to access applications and system services on other computers provides a perfect culture for the spread of a virus.

A virus can do anything that other programs do. The only difference is that it attaches itself to another program and executes secretly when the host program is run. Once a virus is executing, it can perform any function, such as erasing files and programs.

During its lifetime, a typical virus goes through the following four phases:

- **Dormant phase:** The virus is idle. The virus will eventually be activated by some event, such as a date, the presence of another program or file, or the capacity of the disk exceeding some limit. Not all viruses have this stage.

- **Propagation phase:** The virus places an identical copy of itself into other programs or into certain system areas on the disk. Each infected program will now contain a clone of the virus, which will itself enter a propagation phase.

- **Triggering phase:** The virus is activated to perform the function for which it was intended. As with the dormant phase, the triggering phase can be caused by a variety of system events, including a count of the number of times that this copy of the virus has made copies of itself.

- **Execution phase:** The function is performed. The function may be harmless, such as a message on the screen, or damaging, such as the destruction of programs and data files.

Most viruses carry out their work in a manner that is specific to a particular operating system and, in some cases, specific to a particular hardware platform. Thus, they are designed to take advantage of the details and weaknesses of particular systems.

Virus Structure

A virus can be prepended or postpended to an executable program, or it can be embedded in some other fashion. The key to its operation is that the infected program, when invoked, will first execute the virus code and then execute the original code of the program.

A very general depiction of virus structure is shown in Fig 4.4.9. In this case, the virus code, V, is prepended to infected programs, and it is assumed that the entry point to the program, when invoked, is the first line of the program.

```
     program V :=

{goto main;
     1234567;

     subroutine infect-executable :=
          {loop:
          file := get-random-executable-file;
          if (first-line-of-file = 1234567)
             then goto loop
             else prepend V to file; }

     subroutine do-damage :=
          {whatever damage is to be done}

     subroutine trigger-pulled :=
          {return true if some condition holds}

main:    main-program :=
          {infect-executable;
          if trigger-pulled then do-damage;
          goto next;}
next:

}
```

Fig 4.4.9 A Simple Virus

An infected program begins with the virus code and works as follows. The first line of code is a jump to the main virus program. The second line is a special marker that is used by the virus to determine whether or not a potential victim program has already been infected with this virus. When the program is invoked, control is immediately transferred to the main virus program. The virus program first seeks out uninfected executable files and infects them. Next, the virus may perform some action, usually detrimental to the system. This action could be performed every time the program is invoked, or it could be a logic bomb that triggers only under certain conditions. Finally, the virus transfers control to the original program. If the infection phase of the program is reasonably rapid, a user is unlikely to notice any difference between the execution of an infected and uninfected program.

A virus such as the one just described is easily detected because an infected version of a program is longer than the corresponding uninfected one. A way to thwart such a simple means of detecting a virus is to compress the executable file so that both the infected and uninfected versions are of identical length. Fig 4.4.10 shows in general terms the logic required. The key lines in this virus are numbered, and Fig 4.4.11 illustrates the operation. We assume that program P_1 is infected with the virus CV. When this program is invoked, control passes to its virus, which performs the following steps:

```
    program CV :=

{goto main;
    01234567;

    subroutine infect-executable :=
            {loop:
                file := get-random-executable-file;
            if (first-line-of-file = 01234567) then goto loop;
        (1)  compress file;
        (2)  prepend CV to file;
        }

main:   main-program :=
            {if ask-permission then infect-executable;
        (3)  uncompress rest-of-file;
        (4)  run uncompressed file;}
        }
```

Fig 4.4.10 Logic for a Compression Virus

In this example, the virus does nothing other than propagate. As in the previous example, the virus may include a logic bomb.

Initial Infection

Once a virus has gained entry to a system by infecting a single program, it is in a position to infect some or all other executable files on that system when the infected program executes. Thus, viral infection can be completely prevented by preventing the virus from gaining entry in the first place. Unfortunately, prevention is extraordinarily difficult because a virus can be part of any program outside a system. Thus, unless one is content to take an absolutely bare piece of iron and write all one's own system and application programs, one is vulnerable.

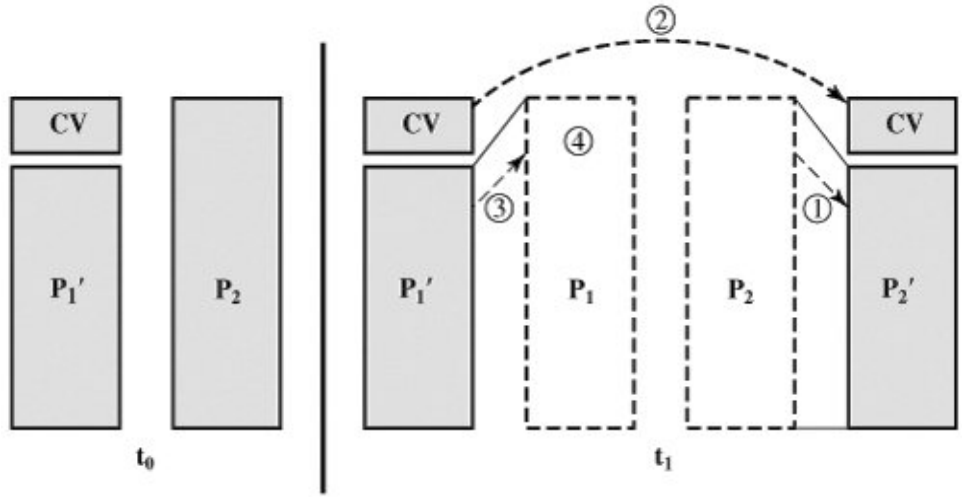


Fig 4.4.11 A Compression Virus

Types of Viruses

There has been a continuous arms race between virus writers and writers of antivirus software since viruses first appeared. As effective countermeasures have been developed for existing types of viruses, new types have been developed. The following categories are among the most significant types of viruses:

- **Parasitic virus:** The traditional and still most common form of virus. A parasitic virus attaches itself to executable files and replicates, when the infected program is executed, by finding other executable files to infect.

- **Memory-resident virus:** Lodges in main memory as part of a resident system program. From that point on, the virus infects every program that executes.

- **Boot sector virus:** Infects a master boot record or boot record and spreads when a system is booted from the disk containing the virus.

- **Stealth virus:** A form of virus explicitly designed to hide itself from detection by antivirus software.

- **Polymorphic virus:** A virus that mutates with every infection, making detection by the "signature" of the virus impossible.

- **Metamorphic virus:** As with a polymorphic virus, a metamorphic virus mutates with every infection. The difference is that a metamorphic virus rewrites itself completely at each iteration, increasing the difficulty of detection. Metamorphic viruses my change their behavior as well as their appearance.

One example of a **stealth virus** was discussed earlier: a virus that uses compression so that the infected program is exactly the same length as an uninfected version. Far more sophisticated techniques are possible. For example, a virus can place intercept logic in disk I/O routines, so that when there is an attempt to read suspected portions of the disk using these routines, the virus will present back the original, uninfected program. Thus, stealth is not a term that applies to a virus as such but, rather, is a technique used by a virus to evade detection.

A **polymorphic virus** creates copies during replication that are functionally equivalent but have distinctly different bit patterns. As with a stealth virus, the purpose is to defeat programs that scan for viruses. In this case, the "signature" of the virus will vary with each copy. To achieve this variation, the virus may randomly insert superfluous instructions or interchange the order of independent instructions. A more effective approach is to use encryption. A portion of the virus, generally called a mutation engine, creates a random encryption key to encrypt the remainder of the virus. The key is stored with the virus, and the mutation engine itself is altered. When an infected program is invoked, the virus uses the stored random key to decrypt the virus. When the virus replicates, a different random key is selected.

Another weapon in the virus writers' armory is the virus-creation toolkit. Such a toolkit enables a relative novice to create quickly a number of different viruses. Although viruses created with toolkits tend to be less sophisticated than viruses designed from scratch, the sheer number of new viruses that can be generated creates a problem for antivirus schemes.

Macro Viruses

In the mid-1990s, macro viruses became by far the most prevalent type of virus. Macro viruses are particularly threatening for a number of reasons:

1. A macro virus is platform independent. Virtually all of the macro viruses infect Microsoft Word documents. Any hardware platform and operating system that supports Word can be infected.

2. Macro viruses infect documents, not executable portions of code. Most of the information introduced onto a computer system is in the form of a document rather than a program.

3. Macro viruses are easily spread. A very common method is by electronic mail.

Macro viruses take advantage of a feature found in Word and other office applications such as Microsoft Excel, namely the macro. In essence, a macro is an executable program embedded in a word processing document or other type of file. Typically, users employ macros to automate repetitive tasks and thereby save keystrokes. The macro language is usually some form of the Basic programming language. A

user might define a sequence of keystrokes in a macro and set it up so that the macro is invoked when a function key or special short combination of keys is input.

Successive releases of Word provide increased protection against macro viruses. For example, Microsoft offers an optional Macro Virus Protection tool that detects suspicious Word files and alerts the customer to the potential risk of opening a file with macros. Various antivirus product vendors have also developed tools to detect and correct macro viruses. As in other types of viruses, the arms race continues in the field of macro viruses, but they no longer are the predominant virus threat.

E-mail Viruses

A more recent development in malicious software is the e-mail virus. The first rapidly spreading e-mail viruses, such as Melissa, made use of a Microsoft Word macro embedded in an attachment. If the recipient opens the e-mail attachment, the Word macro is activated. Then

1. The e-mail virus sends itself to everyone on the mailing list in the user's e-mail package.

2. The virus does local damage.

At the end of 1999, a more powerful version of the e-mail virus appeared. This newer version can be activated merely by opening an e-mail that contains the virus rather than opening an attachment. The virus uses the Visual Basic scripting language supported by the e-mail package.

Thus we see a new generation of malware that arrives via e-mail and uses e-mail software features to replicate itself across the Internet. The virus propagates itself as soon as activated (either by opening an e-mail attachment of by opening the e-mail) to all of the e-mail addresses known to the infected host. As a result, whereas viruses used to take months or years to propagate, they now do so in hours. This makes it very difficult for antivirus software to respond before much damage is done. Ultimately, a greater degree of security must be built into Internet utility and application software on PCs to counter the growing threat.

Worms

A worm is a program that can replicate itself and send copies from computer to computer across network connections. Upon arrival, the worm may be activated to replicate and propagate again. In addition to propagation, the worm usually performs some unwanted function. An e-mail virus has some of the characteristics of a worm, because it propagates itself from system to system. However, we can still classify it as a virus because it requires a human to move it forward. A worm actively seeks out more machines to infect and each machine that is infected serves as an automated launching pad for attacks on other machines.

Network worm programs use network connections to spread from system to system. Once active within a system, a network worm can behave as a computer virus or bacteria, or it could implant Trojan horse programs or perform any number of disruptive or destructive actions.

To replicate itself, a network worm uses some sort of network vehicle. Examples include the following:

- Electronic mail facility: A worm mails a copy of itself to other systems.
- Remote execution capability: A worm executes a copy of itself on another system.
- Remote login capability: A worm logs onto a remote system as a user and then uses commands to copy itself from one system to the other.

The new copy of the worm program is then run on the remote system where, in addition to any functions that it performs at that system, it continues to spread in the same fashion.

A network worm exhibits the same characteristics as a computer virus: a dormant phase, a propagation phase, a triggering phase, and an execution phase. The propagation phase generally performs the following functions:

1. Search for other systems to infect by examining host tables or similar repositories of remote system addresses.
2. Establish a connection with a remote system.
3. Copy itself to the remote system and cause the copy to be run.

The network worm may also attempt to determine whether a system has previously been infected before copying itself to the system. In a multiprogramming system, it may also disguise its presence by naming itself as a system process or using some other name that may not be noticed by a system operator.

As with viruses, network worms are difficult to counter.

The Morris Worm

Until the current generation of worms, the best known was the worm released onto the Internet by Robert Morris in 1998. The Morris worm was designed to spread on UNIX systems and used a number of different techniques for propagation. When a copy began execution, its first task

was to discover other hosts known to this host that would allow entry from this host. The worm performed this task by examining a variety of lists and tables, including system tables that declared which other machines were trusted by this host, users' mail forwarding files, tables by which users gave themselves permission for access to remote accounts, and from a program that reported the status of network connections. For each discovered host, the worm tried a number of methods for gaining access:

1. It attempted to log on to a remote host as a legitimate user. In this method, the worm first attempted to crack the local password file, and then used the discovered passwords and corresponding user IDs. The assumption was that many users would use the same password on different systems. To obtain the passwords, the worm ran a password-cracking program that tried

    a. Each user's account name and simple permutations of it

    b. A list of 432 built-in passwords that Morris thought to be likely candidates

    c. All the words in the local system directory

2. It exploited a bug in the finger protocol, which reports the whereabouts of a remote user.

3. It exploited a trapdoor in the debug option of the remote process that receives and sends mail.

If any of these attacks succeeded, the worm achieved communication with the operating system command interpreter. It then sent this interpreter a short bootstrap program, issued a command to execute that program, and then logged off. The bootstrap program then called back the parent program and downloaded the remainder of the worm. The new worm was then executed.

Recent Worm Attacks

The contemporary era of worm threats began with the release of the Code Red worm in July of 2001. Code Red exploits a security hole in the Microsoft Internet Information Server (IIS) to penetrate and spread. It also disables the system file checker in Windows. The worm probes random IP addresses to spread to other hosts. During a certain period of time, it only spreads. It then initiates a denial-of-service attack against a government Web site by flooding the site with packets from numerous hosts. The worm then suspends activities and reactivates periodically. In the second wave of attack, Code Red infected nearly 360,000 servers in 14 hours. In addition to the havoc it causes at the targeted server, Code Red can consume enormous amounts of Internet capacity, disrupting service.

Code Red II is a variant that targets Microsoft IISs. In addition, this newer worm installs a backdoor allowing a hacker to direct activities of victim computers.

In late 2001, a more versatile worm appeared, known as Nimda. Nimda spreads by multiple mechanisms:

- from client to client via e-mail
- from client to client via open network shares
- from Web server to client via browsing of compromised Web sites
- from client to Web server via active scanning for and exploitation of various Microsoft IIS 4.0 / 5.0 directory traversal vulnerabilities
- from client to Web server via scanning for the back doors left behind by the "Code Red II" worms

The worm modifies Web documents (e.g., .htm, .html, and .asp files) and certain executable files found on the systems it infects and creates numerous copies of itself under various filenames.

In early 2003, the SQL Slammer worm appeared. This worm exploited a buffer overflow vulnerability in Microsoft SQL server. The Slammer was extremely compact and spread rapidly, infecting 90% of vulnerable hosts within 10 minutes. Late 2003 saw the arrival of the Sobig.f worm, which exploited open proxy servers to turn infected machines into spam engines. At its peak, Sobig.f reportedly accounted for one in every 17 messages and produced more than one million copies of itself within the first 24 hours.

Mydoom is a mass-mailing e-mail worm that appeared in 2004. It followed a growing trend of installing a backdoor in infected computers, thereby enabling hackers to gain remote access to data such as passwords and credit card numbers. Mydoom replicated up to 1000 times per minute and reportedly flooded the Internet with 100 million infected messages in 36 hours.

State of Worm Technology

The state of the art in worm technology includes the following:

- Multiplatform: Newer worms are not limited to Windows machines but can attack a variety of platforms, especially the popular varieties of UNIX.
- Multiexploit: New worms penetrate systems in a variety of ways, using exploits against Web servers, browsers, e-mail, file sharing, and other network-based applications.

- Ultrafast spreading: One technique to accelerate the spread of a worm is to conduct a prior Internet scan to accumulate Internet addresses of vulnerable machines.

- Polymorphic: To evade detection, skip past filters, and foil real-time analysis, worms adopt the virus polymorphic technique. Each copy of the worm has new code generated on the fly using functionally equivalent instructions and encryption techniques.

- Metamorphic: In addition to changing their appearance, metamorphic worms have a repertoire of behavior patterns that are unleashed at different stages of propagation.

- Transport vehicles: Because worms can rapidly compromise a large number of systems, they are ideal for spreading other distributed attack tools, such as distributed denial of service zombies.

- Zero-day exploit: To achieve maximum surprise and distribution, a worm should exploit an unknown vulnerability that is only discovered by the general network community when the worm is launched.

Virus Countermeasures

Antivirus Approaches

The ideal solution to the threat of viruses is prevention: Do not allow a virus to get into the system in the first place. This goal is, in general, impossible to achieve, although prevention can reduce the number of successful viral attacks. The next best approach is to be able to do the following:

- **Detection:** Once the infection has occurred, determine that it has occurred and locate the virus.

- **Identification:** Once detection has been achieved, identify the specific virus that has infected a program.

- **Removal:** Once the specific virus has been identified, remove all traces of the virus from the infected program and restore it to its original state. Remove the virus from all infected systems so that the disease cannot spread further.

If detection succeeds but either identification or removal is not possible, then the alternative is to discard the infected program and reload a clean backup version.

Advances in virus and antivirus technology go hand in hand. Early viruses were relatively simple code fragments and could be identified and purged with relatively simple antivirus software packages. As the virus arms race

has evolved, both viruses and, necessarily, antivirus software have grown more complex and sophisticated.

Four generations of antivirus software:

- **First generation:** simple scanners
- **Second generation:** heuristic scanners
- **Third generation:** activity traps
- **Fourth generation:** full-featured protection

A first-generation scanner requires a virus signature to identify a virus. The virus may contain "wildcards" but has essentially the same structure and bit pattern in all copies. Such signature-specific scanners are limited to the detection of known viruses. Another type of first-generation scanner maintains a record of the length of programs and looks for changes in length.

A second-generation scanner does not rely on a specific signature. Rather, the scanner uses heuristic rules to search for probable virus infection. One class of such scanners looks for fragments of code that are often associated with viruses. For example, a scanner may look for the beginning of an encryption loop used in a polymorphic virus and discover the encryption key. Once the key is discovered, the scanner can decrypt the virus to identify it, then remove the infection and return the program to service.

Another second-generation approach is integrity checking. A checksum can be appended to each program. If a virus infects the program without changing the checksum, then an integrity check will catch the change. To counter a virus that is sophisticated enough to change the checksum when it infects a program, an encrypted hash function can be used. The encryption key is stored separately from the program so that the virus cannot generate a new hash code and encrypt that. By using a hash function rather than a simpler checksum, the virus is prevented from adjusting the program to produce the same hash code as before.

Third-generation programs are memory-resident programs that identify a virus by its actions rather than its structure in an infected program. Such programs have the advantage that it is not necessary to develop signatures and heuristics for a wide array of viruses. Rather, it is necessary only to identify the small set of actions that indicate an infection is being attempted and then to intervene.

Fourth-generation products are packages consisting of a variety of antivirus techniques used in conjunction. These include scanning and activity trap components. In addition, such a package includes access control capability, which limits the ability of viruses to penetrate a system

and then limits the ability of a virus to update files in order to pass on the infection.

The arms race continues. With fourth-generation packages, a more comprehensive defense strategy is employed, broadening the scope of defense to more general-purpose computer security measures.

Advanced Antivirus Techniques

More sophisticated antivirus approaches and products continue to appear. In this subsection, we highlight two of the most important.

Generic Decryption

Generic decryption (GD) technology enables the antivirus program to easily detect even the most complex polymorphic viruses, while maintaining fast scanning speeds. Recall that when a file containing a polymorphic virus is executed, the virus must decrypt itself to activate. In order to detect such a structure, executable files are run through a GD scanner, which contains the following elements:

- **CPU emulator:** A software-based virtual computer. Instructions in an executable file are interpreted by the emulator rather than executed on the underlying processor. The emulator includes software versions of all registers and other processor hardware, so that the underlying processor is unaffected by programs interpreted on the emulator.

- **Virus signature scanner:** A module that scans the target code looking for known virus signatures.

- **Emulation control module:** Controls the execution of the target code.

At the start of each simulation, the emulator begins interpreting instructions in the target code, one at a time. Thus, if the code includes a decryption routine that decrypts and hence exposes the virus, that code is interpreted. In effect, the virus does the work for the antivirus program by exposing the virus. Periodically, the control module interrupts interpretation to scan the target code for virus signatures.

During interpretation, the target code can cause no damage to the actual personal computer environment, because it is being interpreted in a completely controlled environment.

The most difficult design issue with a GD scanner is to determine how long to run each interpretation. Typically, virus elements are activated soon after a program begins executing, but this need not be the case. The longer the scanner emulates a particular program, the more likely it is to catch any hidden viruses. However, the antivirus program can take up only a limited amount of time and resources before users complain.

Digital Immune System

The digital immune system is a comprehensive approach to virus protection developed by IBM. The motivation for this development has been the rising threat of Internet-based virus propagation. We first say a few words about this threat and then summarize IBM's approach.

Traditionally, the virus threat was characterized by the relatively slow spread of new viruses and new mutations. Antivirus software was typically updated on a monthly basis, and this has been sufficient to control the problem. Also traditionally, the Internet played a comparatively small role in the spread of viruses. But, two major trends in Internet technology have had an increasing impact on the rate of virus propagation in recent years:

- **Integrated mail systems:** Systems such as Lotus Notes and Microsoft Outlook make it very simple to send anything to anyone and to work with objects that are received.

- **Mobile-program systems:** Capabilities such as Java and ActiveX allow programs to move on their own from one system to another.

In response to the threat posed by these Internet-based capabilities, IBM has developed a prototype digital immune system. This system expands on the use of program emulation discussed in the preceding subsection and provides a general-purpose emulation and virus-detection system. The objective of this system is to provide rapid response time so that viruses can be stamped out almost as soon as they are introduced. When a new virus enters an organization, the immune system automatically captures it, analyzes it, adds detection and shielding for it, removes it, and passes information about that virus to systems running IBM AntiVirus so that it can be detected before it is allowed to run elsewhere.

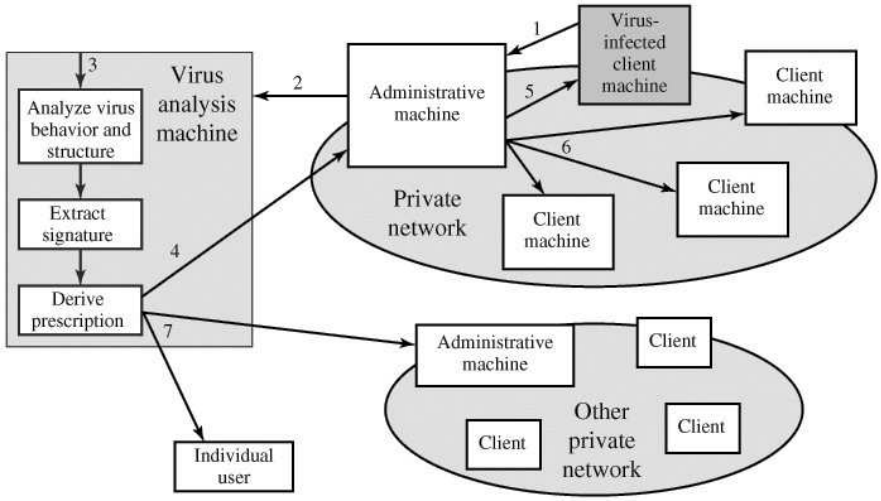


Fig 4.4.12 Digital Immune System

Figure 4.4.12 illustrates the typical steps in digital immune system operation:

1. A monitoring program on each PC uses a variety of heuristics based on system behavior, suspicious changes to programs, or family signature to infer that a virus may be present. The monitoring program forwards a copy of any program thought to be infected to an administrative machine within the organization.

2. The administrative machine encrypts the sample and sends it to a central virus analysis machine.

3. This machine creates an environment in which the infected program can be safely run for analysis. Techniques used for this purpose include emulation, or the creation of a protected environment within which the suspect program can be executed and monitored. The virus analysis machine then produces a prescription for identifying and removing the virus.

4. The resulting prescription is sent back to the administrative machine.

5. The administrative machine forwards the prescription to the infected client.

6. The prescription is also forwarded to other clients in the organization.

7. Subscribers around the world receive regular antivirus updates that protect them from the new virus.

The success of the digital immune system depends on the ability of the virus analysis machine to detect new and innovative virus strains. By constantly analyzing and monitoring the viruses found in the wild, it should be possible to continually update the digital immune software to keep up with the threat.

Behavior-Blocking Software

Unlike heuristics or fingerprint-based scanners, behavior-blocking software integrates with the operating system of a host computer and monitors program behavior in real-time for malicious actions. The behavior blocking software then blocks potentially malicious actions before they have a chance to affect the system. Monitored behaviors can include the following:

- Attempts to open, view, delete, and/or modify files;

- Attempts to format disk drives and other unrecoverable disk operations;
- Modifications to the logic of executable files or macros;
- Modification of critical system settings, such as start-up settings;
- Scripting of e-mail and instant messaging clients to send executable content; and
- Initiation of network communications.

If the behavior blocker detects that a program is initiating would-be malicious behaviors as it runs, it can block these behaviors in real-time and/or terminate the offending software. This gives it a fundamental advantage over such established antivirus detection techniques as fingerprinting or heuristics. While there are literally trillions of different ways to obfuscate and rearrange the instructions of a virus or worm, many of which will evade detection by a fingerprint scanner or heuristic, eventually malicious code must make a well-defined request to the operating system. Given that the behavior blocker can intercept all such requests, it can identify and block malicious actions regardless of how obfuscated the program logic appears to be.

The ability to watch software as it runs in real time clearly confers a huge benefit to the behavior blocker; however, it also has drawbacks. Since the malicious code must actually run on the target machine before all its behaviors can be identified, it can cause a great deal of harm to the system before it has been detected and blocked by the behavior blocking system. For instance, a new virus might shuffle a number of seemingly unimportant files around the hard drive before infecting a single file and being blocked. Even though the actual infection was blocked, the user may be unable to locate their files, causing a loss to productivity or possibly worse.

Distributed Denial of Service Attacks

Distributed denial of service (DDoS) attacks present a significant security threat to corporations, and the threat appears to be growing. A denial of service (DoS) attack is an attempt to prevent legitimate users of a service from using that service. When this attack comes from a single host or network node, then it is simply referred to as a DoS attack. A more serious threat is posed by a DDoS attack. In a DDoS attack, an attacker is able to recruit a number of hosts throughout the Internet to simultaneously or in a coordinated fashion launch an attack upon the target. This section is concerned with DDoS attacks. First, we look at the nature and types of attacks. Next, we examine means by which an attacker is able to recruit a network of hosts for attack launch. Finally, this section looks at countermeasures.

DDoS Attack Description

A DDoS attack attempts to consume the target's resources so that it cannot provide service. One way to classify DDoS attacks is in terms of the type of resource that is consumed. Broadly speaking, the resource consumed is either an internal host resource on the target system or data transmission capacity in the local network to which the target is attacked.

A simple example of an internal resource attack is the SYN flood attack. Figure 4.4.13a shows the steps involved:

1. The attacker takes control of multiple hosts over the Internet, instructing them to contact the target Web server.

2. The slave hosts begin sending TCP/IP SYN (synchronize/initialization) packets, with erroneous return IP address information, to the target.

3. Each SYN packet is a request to open a TCP connection. For each such packet, the Web server responds with a SYN/ACK (synchronize/acknowledge) packet, trying to establish a TCP connection with a TCP entity at a spurious IP address. The Web server maintains a data structure for each SYN request waiting for a response back and becomes bogged down as more traffic floods in. The result is that legitimate connections are denied while the victim machine is waiting to complete bogus “half-open” connections.

The TCP state data structure is a popular internal resource target but by no means the only one. gives the following examples:

1. In many systems, a limited number of data structures are available to hold process information (process identifiers, process table entries, process slots, etc.). An intruder may be able to consume these data structures by writing a simple program or script that does nothing but repeatedly create copies of itself.

2. An intruder may also attempt to consume disk space in other ways, including

    - generating excessive numbers of mail messages
    - intentionally generating errors that must be logged
    - placing files in anonymous ftp areas or network-shared areas

Figure 4.4.13b illustrates an example of an attack that consumes data transmission resources. The following steps are involved:

1. The attacker takes control of multiple hosts over the Internet, instructing them to send ICMP ECHO packets with the target's spoofed IP address to a group of hosts that act as reflectors, as described subsequently.

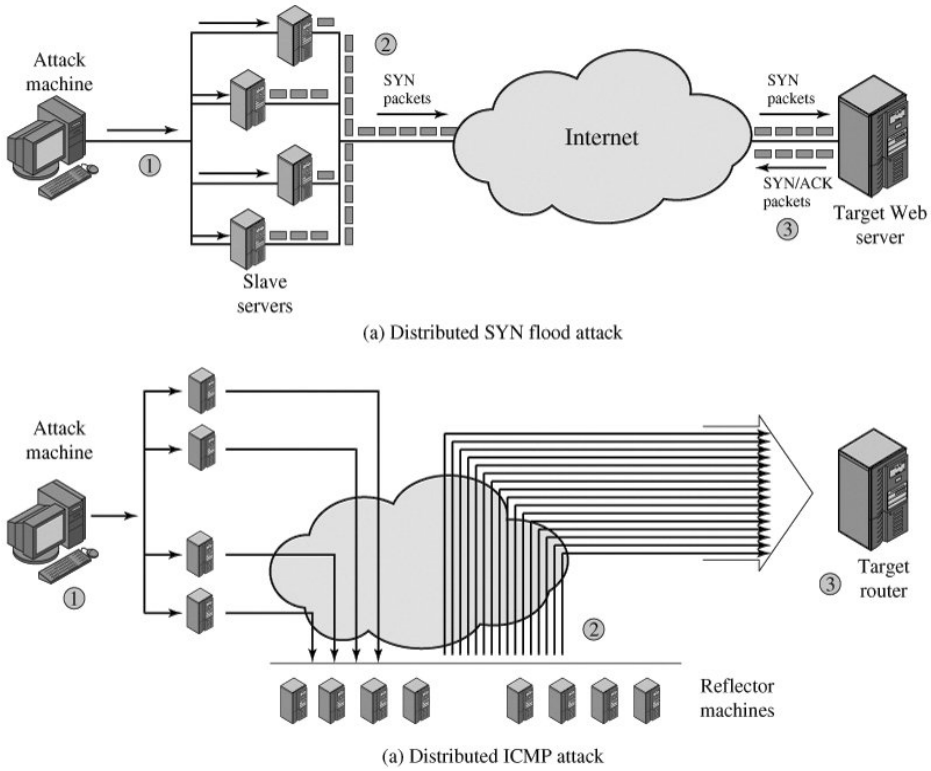


Fig 4.4.13 Examples of Simple DDoS Attacks

2. Nodes at the bounce site receive multiple spoofed requests and respond by sending echo reply packets to the target site.

3. The target's router is flooded with packets from the bounce site, leaving no data transmission capacity for legitimate traffic.

Another way to classify DDoS attacks is as either direct or reflector DDoS attacks. In a direct DDoS attack (Fig 4.4.14a), the attacker is able to implant zombie software on a number of sites distributed throughout the Internet. Often, the DDoS attack involves two levels of zombie machines: master zombies and slave zombies. The hosts of both machines have been infected with malicious code. The attacker coordinates and triggers the master zombies, which in turn coordinate and trigger the slave zombies. The use of two levels of zombies makes it more difficult to trace

the attack back to its source and provides for a more resilient network of attackers.

DDoS Countermeasures

In general, there are three lines of defense against DDoS attacks:

- Attack prevention and preemption (before the attack): These mechanisms enable the victim to endure attack attempts without denying service to legitimate clients. Techniques include enforcing policies for resource consumption and providing backup resources available on demand. In addition, prevention mechanisms modify systems and protocols on the Internet to reduce the possibility of DDoS attacks.

- Attack detection and filtering (during the attack): These mechanisms attempt to detect the attack as it begins and respond immediately. This minimizes the impact of the attack on the target. Detection involves looking for suspicious patterns of behavior. Response involves filtering out packets likely to be part of the attack.

- Attack source traceback and identification (during and after the attack): This is an attempt to identify the source of the attack as a first step in preventing future attacks. However, this method typically does not yield results fast enough, if at all, to mitigate an ongoing attack.

The challenge in coping with DDoS attacks is the sheer number of ways in which they can operate. Thus DDoS countermeasures must evolve with the threat.

REFERENCES

1. William Stallings, Cryptography and Network Security, PHI Publishers

2. www.wikipedia.org

3. Microsoft TechNet (VPN's)

Review Questions:

1. Write a short note on virus and worm.
2. Explain the nature of viruses
3. Give an account of virus counter measures
4. write a short note on Digital Immune System

6. FIREWALLS

OBJECTIVE

In this lesson, we discuss the principles of firewall design and looks at specific techniques. This lesson also covers the related issue of trusted systems.

INTRODUCTION

Firewalls can be an effective means of protecting a local system or network of systems from network-based security threats while at the same time affording access to the outside world via wide area networks and the Internet.

Firewall Design Principles

Information systems in corporations, government agencies, and other organizations have undergone a steady evolution:

- Centralized data processing system, with a central mainframe supporting a number of directly connected terminals
- Local area networks (LANs) interconnecting PCs and terminals to each other and the mainframe
- Premises network, consisting of a number of LANs, interconnecting PCs, servers, and perhaps a mainframe or two
- Enterprise-wide network, consisting of multiple, geographically distributed premises networks interconnected by a private wide area network (WAN)
- Internet connectivity, in which the various premises networks all hook into the Internet and may or may not also be connected by a private WAN

Internet connectivity is no longer optional for organizations. The information and services available are essential to the organization. Moreover, individual users within the organization want and need Internet access, and if this is not provided via their LAN, they will use dial-up capability from their PC to an Internet service provider (ISP). However, while Internet access provides benefits to the organization, it enables the outside world to reach and interact with local network assets. This creates a threat to the organization. While it is possible to equip each workstation

and server on the premises network with strong security features, such as intrusion protection, this is not a practical approach. Consider a network with hundreds or even thousands of systems, running a mix of various versions of UNIX, plus Windows. When a security flaw is discovered, each potentially affected system must be upgraded to fix that flaw. The alternative, increasingly accepted, is the firewall. The firewall is inserted between the premises network and the Internet to establish a controlled link and to erect an outer security wall or perimeter. The aim of this perimeter is to protect the premises network from Internet-based attacks and to provide a single choke point where security and audit can be imposed. The firewall may be a single computer system or a set of two or more systems that cooperate to perform the firewall function.

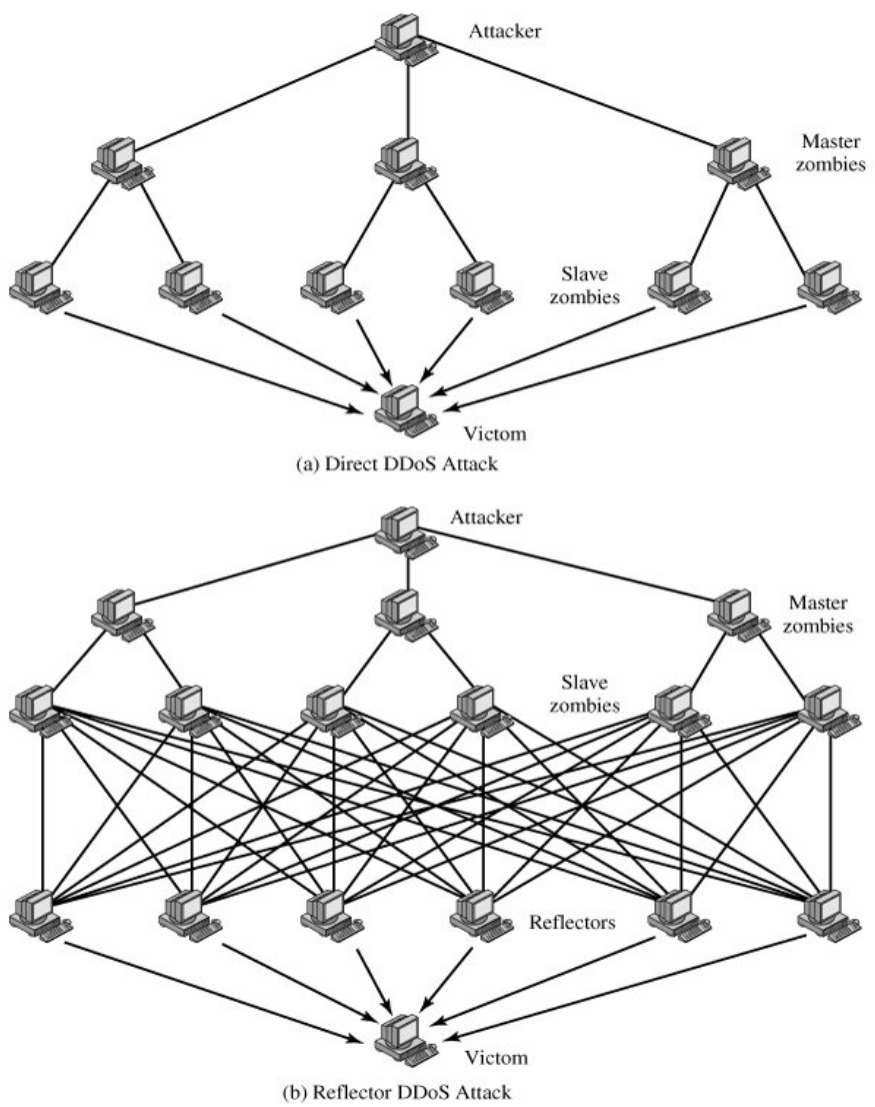


Fig 4.4.14 Types of Flooding-Based DDoS Attacks

Firewall Characteristics

The following are the design goals for a firewall:

1. All traffic from inside to outside, and vice versa, must pass through the firewall. This is achieved by physically blocking all access to the local network except via the firewall. Various configurations are possible, as explained later in this section.

2. Only authorized traffic, as defined by the local security policy, will be allowed to pass. Various types of firewalls are used, which implement various types of security policies, as explained later in this section.

3. The firewall itself is immune to penetration. This implies that use of a trusted system with a secure operating system.

Four general techniques that firewalls use to control access and enforce the site's security policy.

- Service control: Determines the types of Internet services that can be accessed, inbound or outbound. The firewall may filter traffic on the basis of IP address and TCP port number; may provide proxy software that receives and interprets each service request before passing it on; or may host the server software itself, such as a Web or mail service.

- Direction control: Determines the direction in which particular service requests may be initiated and allowed to flow through the firewall.

- User control: Controls access to a service according to which user is attempting to access it. This feature is typically applied to users inside the firewall perimeter (local users). It may also be applied to incoming traffic from external users; the latter requires some form of secure authentication technology, such as is provided in IPSec.

- Behavior control: Controls how particular services are used. For example, the firewall may filter e-mail to eliminate spam, or it may enable external access to only a portion of the information on a local Web server.

Before proceeding to the details of firewall types and configurations, it is best to summarize what one can expect from a firewall. The following capabilities are within the scope of a firewall:

1. A firewall defines a single choke point that keeps unauthorized users out of the protected network, prohibits potentially vulnerable services from entering or leaving the network, and provides protection from various kinds of IP spoofing and routing attacks. The use of a single choke point simplifies security management

because security capabilities are consolidated on a single system or set of systems.

2. A firewall provides a location for monitoring security-related events. Audits and alarms can be implemented on the firewall system.

3. A firewall is a convenient platform for several Internet functions that are not security related. These include a network address translator, which maps local addresses to Internet addresses, and a network management function that audits or logs Internet usage.

4. A firewall can serve as the platform for IPSec. Using the tunnel mode capability, the firewall can be used to implement virtual private networks.

Firewalls have their limitations, including the following:

1. The firewall cannot protect against attacks that bypass the firewall. Internal systems may have dial-out capability to connect to an ISP. An internal LAN may support a modem pool that provides dial-in capability for traveling employees and telecommuters.

2. The firewall does not protect against internal threats, such as a disgruntled employee or an employee who unwittingly cooperates with an external attacker.

3. The firewall cannot protect against the transfer of virus-infected programs or files. Because of the variety of operating systems and applications supported inside the perimeter, it would be impractical and perhaps impossible for the firewall to scan all incoming files, e-mail, and messages for viruses.

Types of Firewalls

Figure 4.4.15 illustrates the three common types of firewalls: packet filters, application-level gateways, and circuit-level gateways. We examine each of these in turn.

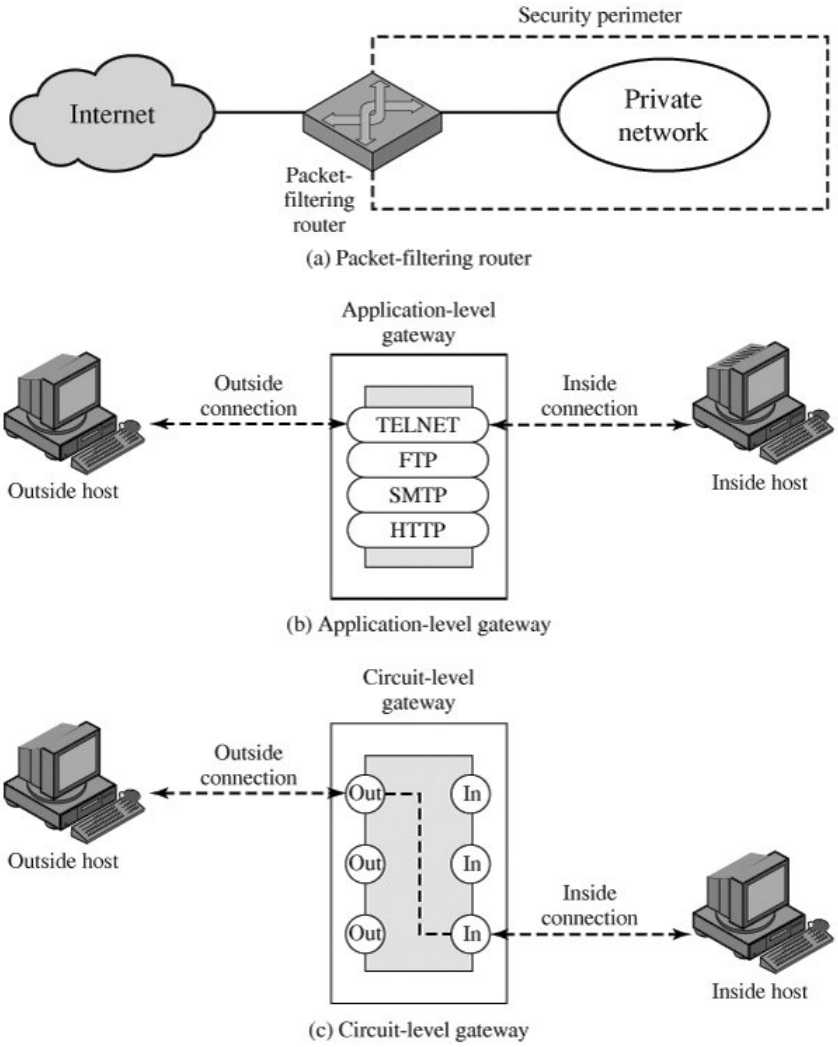


Fig 4.4.15 Firewall Types

Packet-Filtering Router

A packet-filtering router applies a set of rules to each incoming and outgoing IP packet and then forwards or discards the packet. The router is typically configured to filter packets going in both directions (from and to the internal network). Filtering rules are based on information contained in a network packet:

- Source IP address: The IP address of the system that originated the IP packet (e.g., 192.178.1.1)
- Destination IP address: The IP address of the system the IP packet is trying to reach (e.g., 192.168.1.2)

- Source and destination transport-level address: The transport level (e.g., TCP or UDP) port number, which defines applications such as SNMP or TELNET

- IP protocol field: Defines the transport protocol

- Interface: For a router with three or more ports, which interface of the router the packet came from or which interface of the router the packet is destined for

The packet filter is typically set up as a list of rules based on matches to fields in the IP or TCP header. If there is a match to one of the rules, that rule is invoked to determine whether to forward or discard the packet. If there is no match to any rule, then a default action is taken. Two default policies are possible:

- Default = discard: That which is not expressly permitted is prohibited.

- Default = forward: That which is not expressly prohibited is permitted.

The default discard policy is more conservative. Initially, everything is blocked, and services must be added on a case-by-case basis. This policy is more visible to users, who are more likely to see the firewall as a hindrance. The default forward policy increases ease of use for end users but provides reduced security; the security administrator must, in essence, react to each new security threat as it becomes known.

One advantage of a packet-filtering router is its simplicity. Also, packet filters typically are transparent to users and are very fast. The following are the weaknesses of packet filter firewalls:

- Because packet filter firewalls do not examine upper-layer data, they cannot prevent attacks that employ application-specific vulnerabilities or functions. For example, a packet filter firewall cannot block specific application commands; if a packet filter firewall allows a given application, all functions available within that application will be permitted.

- Because of the limited information available to the firewall, the logging functionality present in packet filter firewalls is limited. Packet filter logs normally contain the same information used to make access control decisions (source address, destination address, and traffic type).

- Most packet filter firewalls do not support advanced user authentication schemes. Once again, this limitation is mostly due to the lack of upper-layer functionality by the firewall.

- They are generally vulnerable to attacks and exploits that take advantage of problems within the TCP/IP specification and protocol stack, such as network layer address spoofing. Many packet filter firewalls cannot detect a network packet in which the OSI Layer 3 addressing information has been altered. Spoofing attacks are generally employed by intruders to bypass the security controls implemented in a firewall platform.

- Finally, due to the small number of variables used in access control decisions, packet filter firewalls are susceptible to security breaches caused by improper configurations. In other words, it is easy to accidentally configure a packet filter firewall to allow traffic types, sources, and destinations that should be denied based on an organization's information security policy.

  Some of the attacks that can be made on packet-filtering routers and the appropriate countermeasures are the following:

- IP address spoofing: The intruder transmits packets from the outside with a source IP address field containing an address of an internal host. The attacker hopes that the use of a spoofed address will allow penetration of systems that employ simple source address security, in which packets from specific trusted internal hosts are accepted. The countermeasure is to discard packets with an inside source address if the packet arrives on an external interface.

- Source routing attacks: The source station specifies the route that a packet should take as it crosses the Internet, in the hopes that this will bypass security measures that do not analyze the source routing information. The countermeasure is to discard all packets that use this option.

- Tiny fragment attacks: The intruder uses the IP fragmentation option to create extremely small fragments and force the TCP header information into a separate packet fragment. This attack is designed to circumvent filtering rules that depend on TCP header information. Typically, a packet filter will make a filtering decision on the first fragment of a packet. All subsequent fragments of that packet are filtered out solely on the basis that they are part of the packet whose first fragment was rejected. The attacker hopes that the filtering router examines only the first fragment and that the remaining fragments are passed through. A tiny fragment attack can be defeated by enforcing a rule that the first fragment of a packet must contain a predefined minimum amount of the transport header. If the first fragment is rejected, the filter can remember the packet and discard all subsequent fragments.

Stateful Inspection Firewalls

A traditional packet filter makes filtering decisions on an individual packet basis and does not take into consideration any higher layer context. To understand what is meant by context and why a traditional packet filter is limited with regard to context, a little background is needed. Most standardized applications that run on top of TCP follow a client/server model. For example, for the Simple Mail Transfer Protocol (SMTP), e-mail is transmitted from a client system to a server system. The client system generates new e-mail messages, typically from user input. The server system accepts incoming e-mail messages and places them in the appropriate user mailboxes. SMTP operates by setting up a TCP connection between client and server, in which the TCP server port number, which identifies the SMTP server application, is 25. The TCP port number for the SMTP client is a number between 1024 and 65535 that is generated by the SMTP client.

In general, when an application that uses TCP creates a session with a remote host, it creates a TCP connection in which the TCP port number for the remote (server) application is a number less than 1024 and the TCP port number for the local (client) application is a number between 1024 and 65535. The numbers less than 1024 are the "well-known" port numbers and are assigned permanently to particular applications (e.g., 25 for server SMTP). The numbers between 1024 and 65535 are generated dynamically and have temporary significance only for the lifetime of a TCP connection.

A simple packet-filtering firewall must permit inbound network traffic on all these high-numbered ports for TCP-based traffic to occur. This creates a vulnerability that can be exploited by unauthorized users.

Application-Level Gateway

An application-level gateway, also called a proxy server, acts as a relay of application-level traffic (Fig 4.4.15b). The user contacts the gateway using a TCP/IP application, such as Telnet or FTP, and the gateway asks the user for the name of the remote host to be accessed. When the user responds and provides a valid user ID and authentication information, the gateway contacts the application on the remote host and relays TCP segments containing the application data between the two endpoints. If the gateway does not implement the proxy code for a specific application, the service is not supported and cannot be forwarded across the firewall. Further, the gateway can be configured to support only specific features of an application that the network administrator considers acceptable while denying all other features.

Application-level gateways tend to be more secure than packet filters. Rather than trying to deal with the numerous possible combinations that are to be allowed and forbidden at the TCP and IP level, the application-level gateway need only scrutinize a few allowable applications. In

addition, it is easy to log and audit all incoming traffic at the application level.

A prime disadvantage of this type of gateway is the additional processing overhead on each connection. In effect, there are two spliced connections between the end users, with the gateway at the splice point, and the gateway must examine and forward all traffic in both directions.

Circuit-Level Gateway

A third type of firewall is the circuit-level gateway (Fig 4.4.15c). This can be a stand-alone system or it can be a specialized function performed by an application-level gateway for certain applications. A circuit-level gateway does not permit an end-to-end TCP connection; rather, the gateway sets up two TCP connections, one between itself and a TCP user on an inner host and one between itself and a TCP user on an outside host. Once the two connections are established, the gateway typically relays TCP segments from one connection to the other without examining the contents. The security function consists of determining which connections will be allowed.

A typical use of circuit-level gateways is a situation in which the system administrator trusts the internal users. The gateway can be configured to support application-level or proxy service on inbound connections and circuit-level functions for outbound connections. In this configuration, the gateway can incur the processing overhead of examining incoming application data for forbidden functions but does not incur that overhead on outgoing data.

Bastion Host

A bastion host is a system identified by the firewall administrator as a critical strong point in the network's security. Typically, the bastion host serves as a platform for an application-level or circuit-level gateway. Common characteristics of a bastion host include the following:

- The bastion host hardware platform executes a secure version of its operating system, making it a trusted system.
- Only the services that the network administrator considers essential are installed on the bastion host. These include proxy applications such as Telnet, DNS, FTP, SMTP, and user authentication.
- The bastion host may require additional authentication before a user is allowed access to the proxy services. In addition, each proxy service may require its own authentication before granting user access.

- Each proxy is configured to support only a subset of the standard application's command set.

- Each proxy is configured to allow access only to specific host systems. This means that the limited command/feature set may be applied only to a subset of systems on the protected network.

- Each proxy maintains detailed audit information by logging all traffic, each connection, and the duration of each connection. The audit log is an essential tool for discovering and terminating intruder attacks.

- Each proxy module is a very small software package specifically designed for network security. Because of its relative simplicity, it is easier to check such modules for security flaws. For example, a typical UNIX mail application may contain over 20,000 lines of code, while a mail proxy may contain fewer than 1000.

- Each proxy is independent of other proxies on the bastion host. If there is a problem with the operation of any proxy, or if a future vulnerability is discovered, it can be uninstalled without affecting the operation of the other proxy applications. Also, if the user population requires support for a new service, the network administrator can easily install the required proxy on the bastion host.

- A proxy generally performs no disk access other than to read its initial configuration file. This makes it difficult for an intruder to install Trojan horse sniffers or other dangerous files on the bastion host.

- Each proxy runs as a nonprivileged user in a private and secured directory on the bastion host.

Firewall Configurations

In addition to the use of a simple configuration consisting of a single system, such as a single packet-filtering router or a single gateway (Fig 4.4.15), more complex configurations are possible and indeed more common. Fig 4.4.16 illustrates three common firewall configurations. We examine each of these in turn.

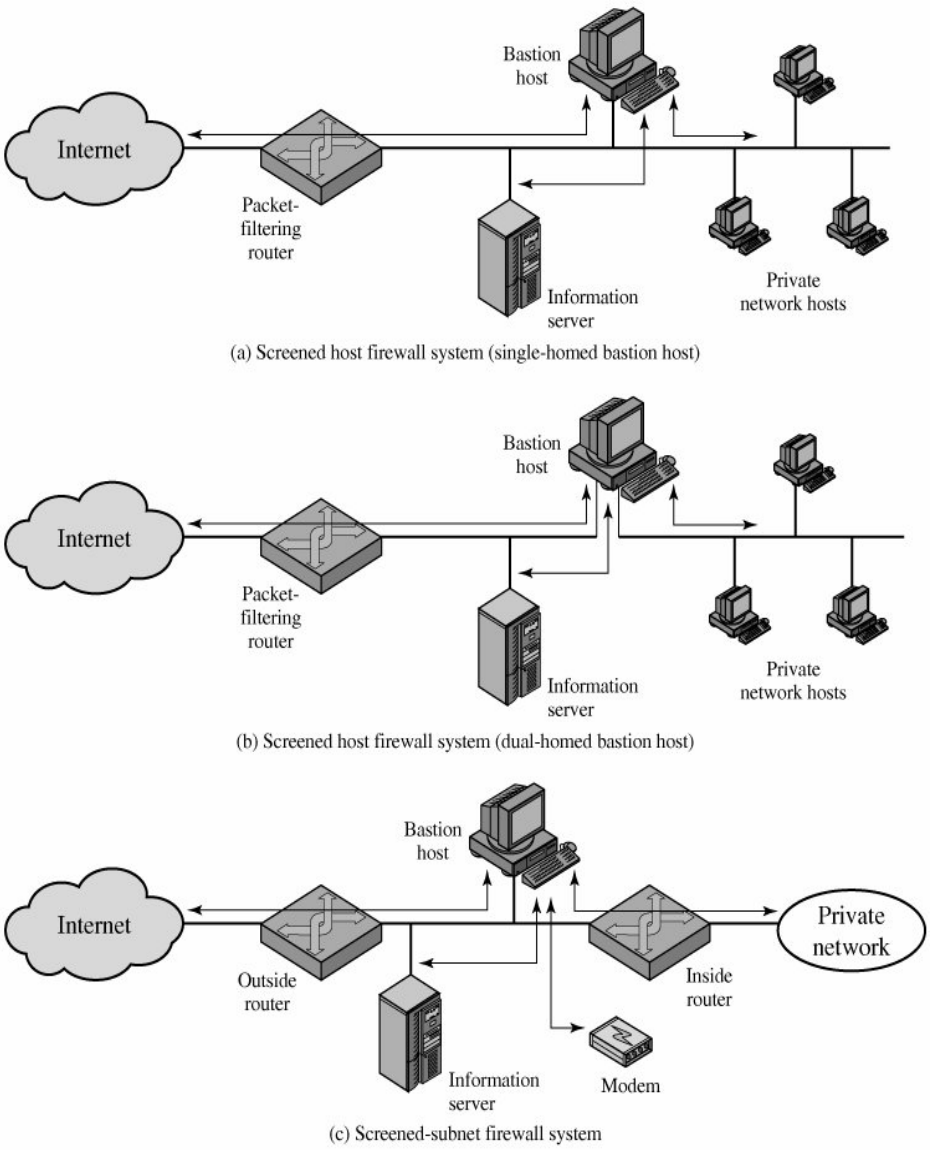


Fig 4.4.16 Firewall Configurations

In the screened host firewall, single-homed bastion configuration (Fig 4.4.16a), the firewall consists of two systems: a packet-filtering router and a bastion host. Typically, the router is configured so that

1. For traffic from the Internet, only IP packets destined for the bastion host are allowed in.
2. For traffic from the internal network, only IP packets from the bastion host are allowed out.

The bastion host performs authentication and proxy functions. This configuration has greater security than simply a packet-filtering router or an application-level gateway alone, for two reasons. First, this configuration implements both packet-level and application-level filtering, allowing for considerable flexibility in defining security policy. Second, an intruder must generally penetrate two separate systems before the security of the internal network is compromised.

This configuration also affords flexibility in providing direct Internet access. For example, the internal network may include a public information server, such as a Web server, for which a high level of security is not required. In that case, the router can be configured to allow direct traffic between the information server and the Internet.

In the single-homed configuration just described, if the packet-filtering router is completely compromised, traffic could flow directly through the router between the Internet and other hosts on the private network. The screened host firewall, dual-homed bastion configuration physically prevents such a security breach (Fig 4.4.16b). The advantages of dual layers of security that were present in the previous configuration are present here as well. Again, an information server or other hosts can be allowed direct communication with the router if this is in accord with the security policy.

The screened subnet firewall configuration of Fig 4.4.16c is the most secure of those we have considered. In this configuration, two packet-filtering routers are used, one between the bastion host and the Internet and one between the bastion host and the internal network. This configuration creates an isolated subnetwork, which may consist of simply the bastion host but may also include one or more information servers and modems for dial-in capability. Typically, both the Internet and the internal network have access to hosts on the screened subnet, but traffic across the screened subnet is blocked. This configuration offers several advantages:

- There are now three levels of defense to thwart intruders.
- The outside router advertises only the existence of the screened subnet to the Internet; therefore, the internal network is invisible to the Internet.
- Similarly, the inside router advertises only the existence of the screened subnet to the internal network; therefore, the systems on the inside network cannot construct direct routes to the Internet.

REFERENCES

1. William Stallings, Cryptography and Network Security, PHI Publishers
2. www.wikipedia.org
3. Microsoft TechNet (VPN's)

Review Questions:

1. Explain firewall design principles and characteristics in detail
2. Explain the configuration of firewalls in detail.
3. Write a short note on Bastion Host