CHAPTER 1

Introduction to Security

1.1. INTRODUCTION

Inside This Chapter

Our society is maintained by information: information about who we were, and in some cases what we will become. We live in the Information age, a time where movement of information is faster than physical movement. Some say that we live in a new type of society called an Information Society, in which the creation, distribution and manipulation of information has become a significant economic and cultural activity. Matthew Lesko, a columnist, made this point clear when he wrote, "Information is the currency of today's world". Sir Francis Bacon an English statesman from the 1500s proclaimed wisely that, "Knowledge is power". His words echo today in the familiar truism, "Information is power".

The 1990's can be characterized as the decade of Information Technology. Advancements in computer and telecommunication technologies helped to drive the U.S. and world economies to staggering growth through the year 2000. The technological advancements and their impact on the economy and society had a revolutionary effect around the world. This growth was also driven by the acceptance of the Internet as a medium for communication among consumers, businesses, and governments. The Internet provided a cost-effective and efficient medium to share information that the world had not seen prior to the 1990's. In 1995, the U.S. had approximately 18 million users on the Internet, and as of April 2002, there were more than 165 million users.

The advancement of technology, the Internet, and information sharing has had both positive and negative impacts. One of the negative impacts was the large increase in new "information" threats. The number of threats and reported computer related incidents increased at a tremendous rate by the end of the 1990's, and into the 2000's. Many of the computer incidents exploited confidential information being stored by companies in a variety of different industries. The ability to carry out threats against information systems has been made easier due to the sharp increase in system vulnerabilities. Unauthorized access to confidential information was also the result of weak

or non-existent information security practices. Not identifying and mitigating risks is a leading cause of unauthorized access and the exploitation of vulnerabilities.

These computer incidents have raised a number of concerns about how information is secured and maintained. With such a critical dependency on information, a threat to the security and trustworthiness of information was also a threat to the world economies. The cost to protect against information threats has increased as the number of threats and vulnerabilities also increase.

1.2. NEED OF SECURITY

As all of us know Internet has completely transformed our lives as everything is now available globally and our world has become a global village now. With wide spectrum of Internet in our lives Internet Security is something that has grown to be a main concern among society. As ever more people use the internet for shopping, business transactions, online banking, etc., the incidence of internet fraud and scams has shot up in an alarming fashion. Not only has the level of internet crime increased but the scammers and fraudsters grow cleverer and more sophisticated every day. What can you do to fight back? With computers being a critical component, it is more valuable than ever to ensure the security of your networks particularly where there is sensitive data. The objective of computer security varies and can include protection of information from theft or corruption, or the preservation of availability, as defined in the security policy.

Security is often viewed as the need to protect one or the more aspects of network's operation and permitted use (access, behaviour, performance, privacy and confidentiality included). Security requirements may be Global or Local in their scope, depending upon the network's or internet work's purpose of design and deployment.

Primary elements of security of any computer network include security provisioning at the sending node, Intermediate forwarding node, receiving node, interconnection links and mechanism of transmission or reception at physical and logical levels. Extraneous factors that these elements may be influenced by may include various kinds of external and internal attacks, unintentional leakages and location of devices include in communication. Apart from the obvious networking elements, network security is also influenced by the System and Application software security provisioning or lack of it on individual nodes.

1.3. SECURITY PRINCIPLES OR SECURITY GOALS

For over twenty years, information security has held confidentiality, integrity and availability (known as the CIA triad) to be the core principles of information security.

1.3.1. Confidentiality (Protect Information Value)

Confidentiality is the most common aspect of information security. Confidentiality is the term used to prevent the disclosure of information to unauthorized individuals or systems. It specifies that only sender and the recipient should be able to access the contents of a message. Confidentiality gets compromised if an unauthorized access takes place.

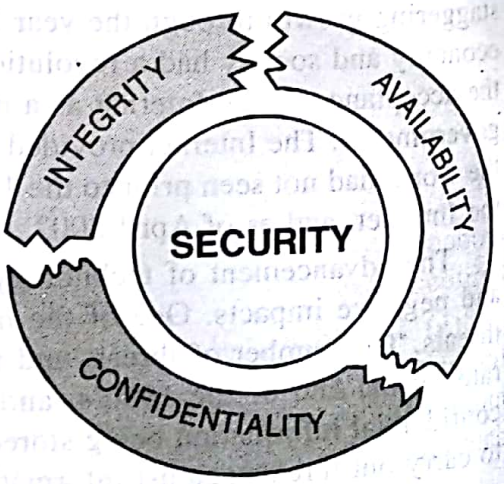


Fig 1.1. *Security Principles*

1.3.2. Integrity (Protect Information Accuracy)

Integrity as a concept means that there is resistance to alteration or substitution of data, and/or that such changes are detected and provable. The information should not be changed except by an authorized agent (*i.e.* contains no modification, insertion, deletion or replay). The receiver of a message should be able to check whether the message was modified during transmission either accidentally or deliberately. Integrity can be maintained at many levels, from the hardware all the way to the application logic.

1.3.3. Availability (Ensure Information Delivery)

Availability refers, unsurprisingly, to the availability of information resources. An information system that is not available when you need it is at least as bad as none at all. Availability provides assurance that the information is accessible when needed, by those who need them. High availability systems aim to remain available at all times, preventing service disruptions due to power outages, hardware failures, and system upgrades. Ensuring availability also involves preventing denial-of-service attacks.

1.4. SECURITY SERVICES AND MECHANISM

The **International Telecommunication Union-Telecommunication Standardization Sector** (ITU-T) provides some security services and some mechanisms to implement those services.

1.4.1. Security Services

ITU-T (X.800) defines a security services as a service provided by a protocol layer of communicating open system, which ensures adequate security of the system or of data transfers. It divides these services in to **five** categories:

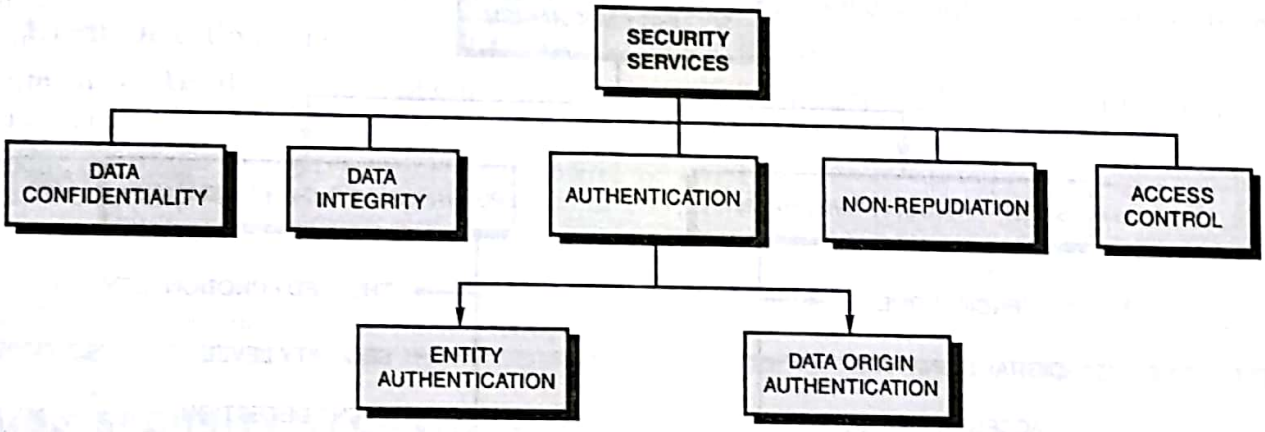


Fig. 1.2. *Security Services*

(*a*) Data Confidentiality: Data confidentiality is designed to protect data from attacks. It is a property of data which prevents it from unauthorized disclosure means "the wrong person" and limited information access to authorized user means "the right person".

(*b*) Data Integrity: Integrity refers to the trustworthiness of information resources. "Data Integrity" refers to the accuracy and consistency of data. It is designed to protect data from modification, insertion, deletion and replaying of message by intruder. It provides assurance that data received are the same as send by an authorized entity.

(*c*) Authentication: Authentication provides the assurance that the communicating entity means sender or receiver is the one that it actually claims to be. In order to guaranty the success of an

exchange and whether at the origin or at the receiving end of the message, exchanging parties must be authentified **before** the process is initiated. Two specific Authentication services are defined in X.800:

(1) **Peer Entity Authentication:** Used in association with a logical connection to provide confidence in the identity of the entities connected. It is provided for use at the establishment of a connection at the time of Data transfer. Supported Security mechanism for Peer Entity Authentication are encipherment, digital signature and authentication exchange.

(2) **Data Origin Authentication:** In a connectionless transfer, provides assurance that the source of received data is as claimed. It does not provide protection against the duplication or modification of data unit. This type of service supports application like e-mail where there are no prior interaction between the communicating entities. Supported Security mechanism for Data Origin Authentication are encipherment and digital signature.

(*d*) **Non-Repudiation:** It provides protection against denial by one of the entities involved in a communication of having participated in all or part of the communication means when a message is sent, the receiver can prove that the alleged, sender in fact sent the message. Similarly, when a message is received, the sender can prove that the alleged receiver in fact received the message.

(*e*) **Access Control:** It provides protection against unauthorized use of resources. It is required to that access to information resources may be controlled by the target system or application. To achieve this, each entity trying to gain access must first be identified.

1.4.2. Security Mechanism

A mechanism is designed to implement security services. ITU-T (X.800) defines security mechanism into two parts as shown in Fig. 1.3.

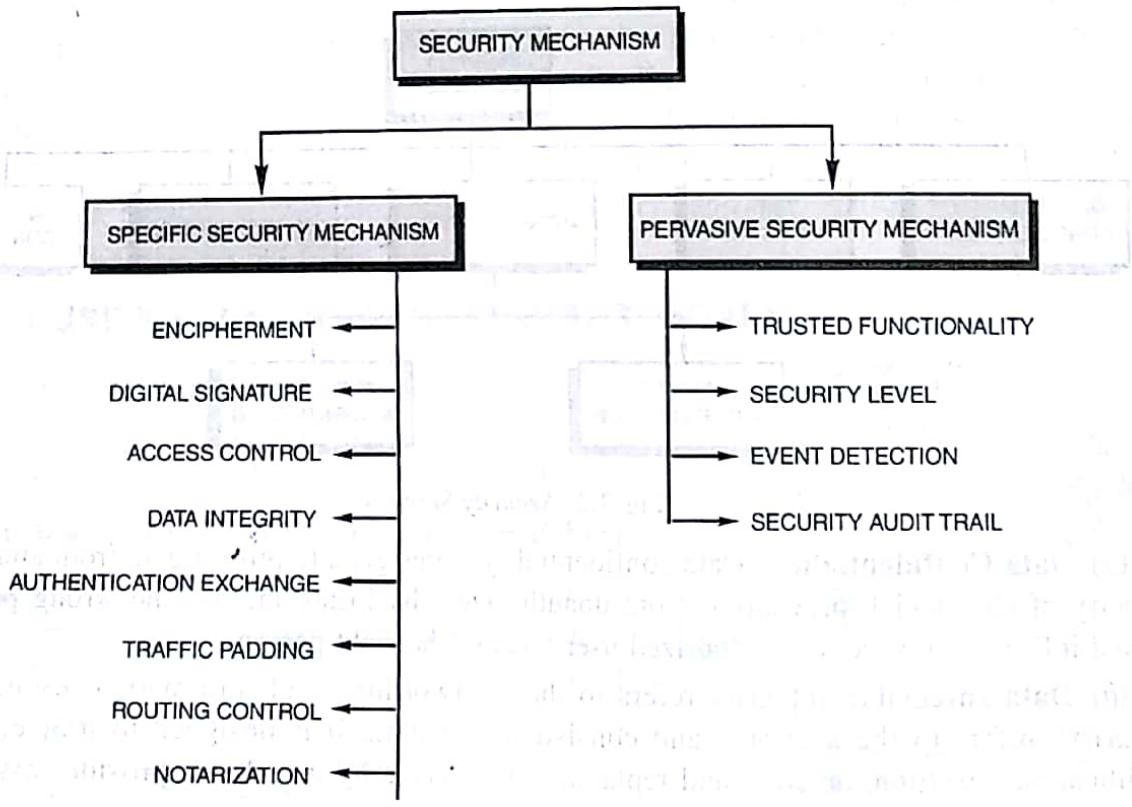


Fig. 1.3. *Security Mechanism*

(1) Specific Security Mechanism

Mechanism may be incorporated into the appropriate protocol layer in order to provide some of the OSI security services.

Encipherment : The use of mathematical algorithms to transform data into a form that is not readily intelligible. The transformation and subsequent recovery of data depend on an algorithm and zero or more encryption keys.

Digital Signature : Data appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery.

Access Control : A variety of mechanisms that enforce access rights to resource.

Data Integrity : A variety of mechanism used to assure the integrity of a data unit or stream of data units.

Authentication Exchange : A mechanism intended to ensure the identity of an entity by means of information exchange.

Traffic Padding : The insertion of bits into gaps in a data stream to frustrate traffic analysis attempts.

Routing Control : Enables selection of particular physical secure routes for certain data and allows routing changes, especially when a breach of security is suspected.

Notarization : The use of a trusted third party to assure certain properties of a data exchange.

(2) Pervasive Security Mechanisms

Mechanism those are not specific to any particular OSI security service or protocol layer.

Trusted Functionality : That which is perceived to a some criteria.

Security Level : The marking bound to a resource that names the security attributes of that resource.

Event Detection : Detection of security–relevant events.

Security Audit Trail : Data collected and potentially used to facilitate a security audit, which is an independent review and examination of system records and activities.

NOTE ▶

X.800 is RFC (Request for Comment). RFC is used to make Internet standards. To make new Internet standard first Internet draft is prepared and after recommendation from Internet authorities, a draft is published as a Request for Comment (RFC).

1.5. SECURITY TECHNIQUES

There are two techniques to implement security goals.

1.5.1. Cryptography

Cryptography is the science of using mathematics to encrypt and decrypt data. Cryptography enables you to store sensitive information or transmit it across insecure networks (like the Internet) so that it cannot be read by anyone except the intended recipient. A cryptographic algorithm works in combination with a *key*—a word, number, or phrase—to encrypt the plaintext. The same plaintext encrypts to different ciphertext with different keys. The security of encrypted data is entirely dependent on two things: the strength of the cryptographic algorithm and the secrecy of the key.

1. Symmetric Key Encryption

In private key cryptography, also called *secret-key* or *symmetric-key* encryption, one key is used both for encryption and decryption. The Data Encryption Standard (DES) is an example of a symmetric key cryptosystem that is widely employed by the U.S. government. Figure 1.4 is an illustration of the symmetric key encryption process. For a sender and recipient to communicate securely using symmetric key encryption, they must agree upon a key and keep it secret between themselves.

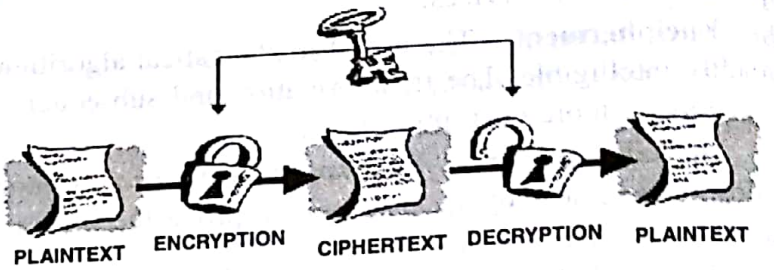


Fig. 1.4. *Symmetric Key Encryption*

2. Asymmetric Key Encryption

In private key cryptography, also called *asymmetric-key* encryption, uses a *pair* of keys for encryption. A *public key*, which encrypts data, and a corresponding *private key* (*secret key*) for decryption. You publish your public key to the world while keeping your private key secret. It is computationally infeasible to deduce the private key from the public key. Anyone who has a public key can encrypt information but cannot decrypt it. Only the person who has the corresponding private key can decrypt the information. The need for sender and receiver to share secret keys via some secure channel is eliminated; all communications involve only public keys, and no private key is ever transmitted or shared.

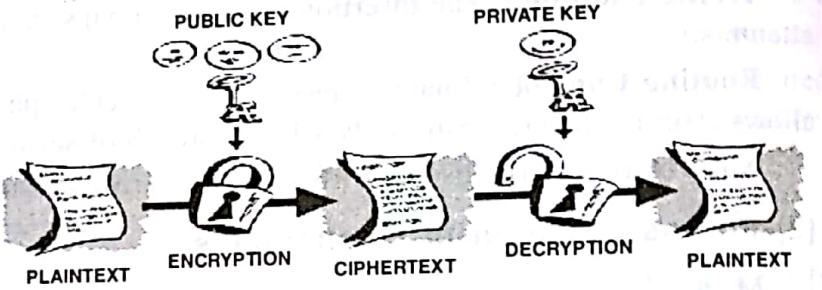


Fig. 1.5. *Asymmetric Key Encryption*

1.5.2. 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, suspects the existence of the message, a form of security through obscurity. The word steganography is of Greek origin and means "concealed writing" from the Greek words steganos meaning "covered or protected", and graphein meaning "to write".

The first recorded use of the term was in 1499 by Johannes Trithemius. Generally, messages will appear to be something else: images, articles, shopping lists, or some other cover text and, classically, the hidden message may be in invisible ink between the visible lines of a private letter.

The advantage of steganography, over cryptography alone, is that messages do not attract attention to themselves, whereas cryptography protects the contents of a message, steganography can be said to protect both messages and communicating parties.

Steganography includes the concealment of information within computer files. In digital steganography, electronic communications may include steganographic coding inside of a transport layer, such as a document file, image file, program or protocol. Media files are ideal for steganographic transmission because of their large size. As a simple example, a sender might start

with an innocuous image file and adjust the colour of every 100th pixel to correspond to a letter in the alphabet, a change so subtle that someone not specifically looking for it is unlikely to notice it.

1.5.2.1. Steganographic Techniques

1. Physical Steganography

Steganography has been widely used, including in recent historical times and the present day. Possible permutations are endless and known examples include:

- Hidden messages within wax tablets — in ancient Greece, people wrote messages on the wood, and then covered it with wax upon which an innocent covering message was written.
- Hidden messages on messenger's body — also used in ancient Greece. Herodotus tells the story of a message tattooed on a slave's shaved head, hidden by the growth of his hair, and exposed by shaving his head again. The message allegedly carried a warning to Greece about Persian invasion plans. This method has obvious drawbacks, such as delayed transmission while waiting for the slave's hair to grow, and the restrictions on the number and size of messages that can be encoded on one person's scalp.
- During World War II, the French Resistance sent some messages written on the backs of couriers using invisible ink.
- Hidden messages on paper written in secret inks, under other messages or on the blank parts of other messages.
- Messages written in Morse code on knitting yarn and then knitted into a piece of clothing worn by a courier.
- Messages written on envelopes in the area covered by postage stamps.
- During and after World War II, espionage agents used photographically produced microdots to send information back and forth. Microdots were typically minute, approximately less than the size of the period produced by a typewriter. World War II microdots needed to be embedded in the paper and covered with an adhesive, such as collodion. This was reflective and thus detectable by viewing against glancing light. Alternative techniques included inserting microdots into slits cut into the edge of post cards.
- During World War II, a spy for Japan in New York City, Velvalee Dickinson, sent information to accommodation addresses in neutral South America. She was a dealer in dolls, and her letters discussed how many of this or that doll to ship. The stegotext was the doll orders, while the concealed "plaintext" was itself encoded and gave information about ship movements, etc. Her case became somewhat famous and she became known as the Doll Woman.
- Cold War counter-propaganda. In 1968, crew members of the USS *Pueblo* intelligence ship held as prisoners by North Korea, communicated in sign language during staged photo opportunities, informing the United States they were not defectors, but rather were being held captive by the North Koreans. In other photos presented to the U.S., crew members gave "the finger" to the unsuspecting North Koreans, in an attempt to discredit photos that showed them smiling and comfortable.

2. Digital Steganography

Modern steganography entered the world in 1985 with the advent of the personal computer being applied to classical steganography problems. Development following that was slow, but has

since taken off, going by the number of "stego" programs available: Over 800 digital steganography applications have been identified by the Steganography Analysis and Research Center. Digital steganography techniques include:

- Concealing messages within the lowest bits of noisy images or sound files.
- Concealing data within encrypted data or within random data. The data to be concealed is first encrypted before being used to overwrite part of a much larger block of encrypted data or a block of random data (an unbreakable cipher like the one-time pad generates ciphertexts that look perfectly random if you don't have the private key).
- Chaffing and winnowing.
- Mimic functions convert one file to have the statistical profile of another. This can thwart statistical methods that help brute-force attacks identify the right solution in a ciphertext—only attack.
- Concealed messages in tampered executable files, exploiting redundancy in the targeted instruction set.
- Pictures embedded in video material (optionally played at slower or faster speed).
- Injecting imperceptible delays to packets sent over the network from the keyboard. Delays in key presses in some applications (telnet or remote desktop software) can mean a delay in packets, and the delays in the packets can be used to encode data.
- Changing the order of elements in a set.
- Content-Aware Steganography hides information in the semantics a human user assigns to a datagram. These systems offer security against a non-human adversary/warden.
- Blog-Steganography. Messages are fractionalized and the (encrypted) pieces are added as comments of orphaned web-logs (or pin boards on social network platforms). In this case the selection of blogs is the symmetric key that sender and recipient are using; the carrier of the hidden message is the whole blogosphere.
- Modifying the echo of a sound file (Echo Steganography).

3. Network Steganography

All information hiding techniques that may be used to exchange steganograms in telecommunication networks can be classified under the general term of network steganography. This nomenclature was originally in 2003. Contrary to the typical steganographic methods which utilize digital media (images, audio and video files) as a cover for hidden data, network steganography utilizes communication protocols' control elements and their basic intrinsic functionality. As a result, such methods are harder to detect and eliminate.

Typical network steganography methods involve modification of the properties of a single network protocol. Such modification can be applied to the PDU (Protocol Data Unit), to the time relations between the exchanged PDUs, or both (hybrid methods).

Network steganography covers a broad spectrum of techniques, which include, among others:

- **Steganophony:** The concealment of messages in Voice-over-IP conversations, e.g. the employment of delayed or corrupted packets that would normally be ignored by the receiver (this method is called LACK—Lost Audio Packets Steganography), or, alternatively, hiding information in unused header fields.
- **WLAN Steganography:** The utilization of methods that may be exercised to transmit steganograms in Wireless Local Area Networks. A practical example of WLAN Steganography is the HICCUPS system (Hidden Communication System for Corrupted Networks).

4. Printed Steganography

Digital steganography output may be in the form of printed documents. A message, the plaintext, may be first encrypted by traditional means, producing a ciphertext. Then, an innocuous covertext is modified in some way so as to contain the ciphertext, resulting in the stegotext. For example, the letter size, spacing, typeface, or other characteristics of a covertext can be manipulated to carry the hidden message. Only a recipient who knows the technique used can recover the message and then decrypt it. Francis Bacon developed Bacon's cipher as such a technique.

The ciphertext produced by most digital steganography methods, however, is not printable. Traditional digital methods rely on perturbing noise in the channel file to hide the message, as such, the channel file must be transmitted to the recipient with no additional noise from the transmission. Printing introduces much noise in the ciphertext, generally rendering the message unrecoverable. There are techniques that address this limitation, one notable example is ASCII Art Steganography.

5. Text Steganography

Steganography can be applied to different types of media including text, audio, image and video etc. However, text steganography is considered to be the most difficult kind of steganography due to lack of redundancy in text as compared to image or audio but still has smaller memory occupation and simpler communication. The method that could be used for text steganography is data compression. Data compression encodes information in one representation into another representation. The new representation of data is smaller in size. One of the possible schemes to achieve data compression is Huffman coding. Huffman coding assigns smaller length codewords to more frequently occurring source symbols and longer length codewords to less frequently occurring source symbols.

6. Steganography using Sudoku Puzzle

This is the art of concealing data in an image using Sudoku which is used like a key to hide the data within an image. Steganography using Sudoku puzzles has as many keys as there are possible solutions of a Sudoku puzzle. This is equivalent to around 70 bits, making it much stronger than the DES method which uses a 56 bit key.

1.6. SECURITY ATTACKS

Any action that compromises the security of information owned by an organization is termed as security attack. Security attack can happen at the application level on the network level. Threats are commonly categorized according to the goal of the attack. There are two types of Attack:

(1) Passive Attack (2) Active Attack

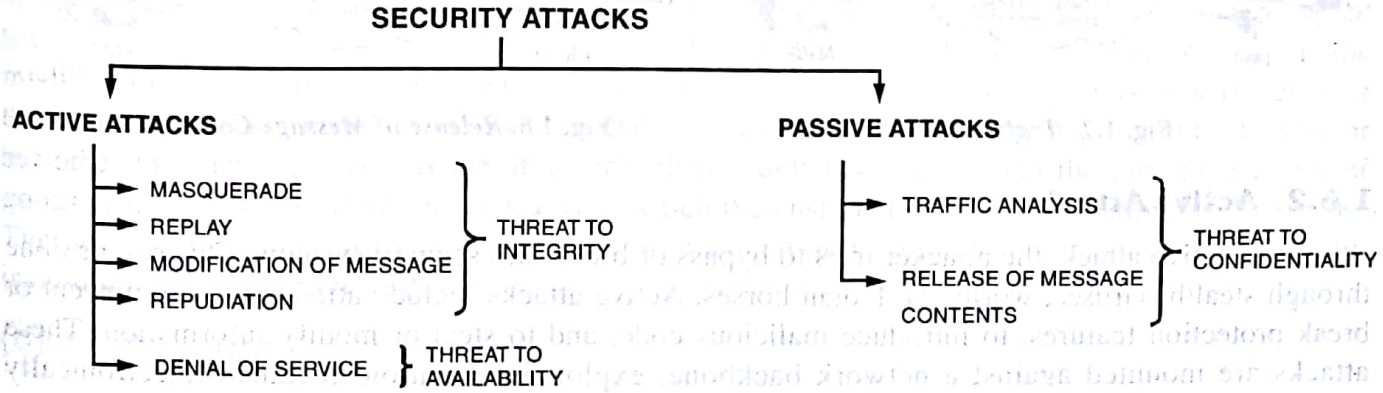


Fig. 1.6. *Taxonomy of Security Attacks with Relation of Security Goals*

1.6.1. Passive Attack

A passive attack on a communication system is one in which the attacker only read messages but he does not alter the messages. A passive attack monitors unencrypted traffic and looks for clear text passwords and sensitive information that can be used in other types of attacks. Passive attacks include traffic analysis, monitoring of unprotected communications, decrypting weakly encrypted traffic, and capturing authentication information such as passwords. Passive interception of network operations enables adversaries to see upcoming actions. Passive attacks result in the disclosure of information or data files to an attacker without the consent or knowledge of the user. Passive attack can be either traffic analysis or release of message contents are described below:

(a) Traffic Analysis

Traffic analysis is the process of intercepting and examining messages in order to deduct information from patterns in communication as shown in Fig. 1.6. 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. Traffic analysis can be performed in the context of military intelligence or counter intelligence. And is a concern in computer security. For example, we can encode messages using a code language so that only the desired parties understand the contents of a message, because only they know the code language. However if many such messages are passing through, a passive attacker could try to figure out the similarities between them to come up with some sort of pattern that provides some clues regarding the communication that is taking place. Such attempts of analyzing encoded messages to come up with likely pattern are the work of this attack.

(b) Release of Message Contents

Release of message contents may refer to unauthorized access of confidential information by third party as shown in Fig. 1.7. An e-mail, transferring of file containing sensitive data are only accesses by the authorized person to whom they are sent. Otherwise the contents of the message are released and any unauthorized entity can use this information for their own benefit.

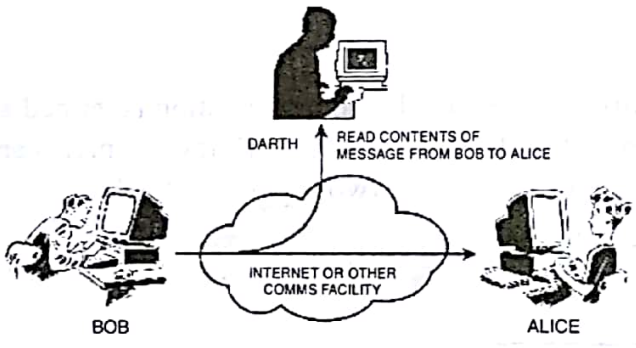


Fig. 1.7. *Traffic Analysis*

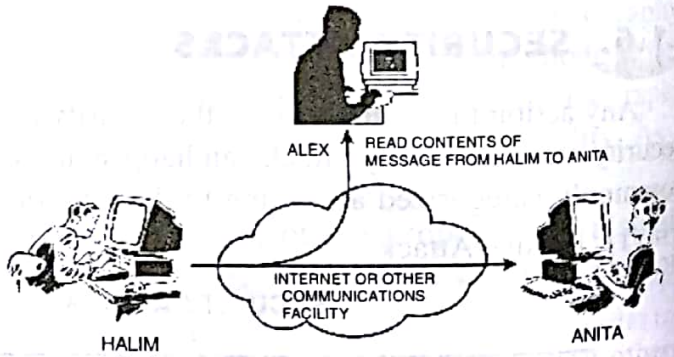


Fig. 1.8. *Release of Message Contents*

1.6.2. Active Attack

In an active attack, the attacker tries to bypass or break into secured systems. This can be done through stealth, viruses, worms or Trojan horses. Active attacks include attempts to circumvent or break protection features, to introduce malicious code, and to steal or modify information. These attacks are mounted against a network backbone, exploit information in transit, electronically penetrate an enclave, or attack an authorized remote user during an attempt to connect to an

enclave. Active attacks result in the disclosure or dissemination of data files, DoS or modification of message.

(a) Masquerade

Masquerade attack is a type of attack in which one system assumes the identity of another. In this attack the attacker pretends to be an authorized user of a system in order to gain access to it or to gain greater privileges that they are authorized for. Masquerade attacks can occur in several different ways. In general terms, a masquerader may get access to a legitimate user's account either by stealing a victim's password or through a break in and installation of a rootkit or keylogger. Another perhaps more common case in laziness and misplaces trust by a user, such as the case when a user leaves his or her terminal or client open and logged in allowing any nearby co-worker to pose as a masquerader. In these two cases, the identify thief must log in with the victim's credentials and begin issuing commands within the bounds of one user session.

(b) Replay

A replay attack occurs when an attacker copies a stream of messages between two parties and replays the stream to one or more of the parties as shown in Fig. 1.9. Imagine a scenario in which a client sends an encrypted user name and password to a server to log in. If a hacker intercepts the communication (using monitoring software) and replays the sequence, he will obtain the same rights as the user. If the system enables password modification, he could even replace it with another, depriving the user of his access.

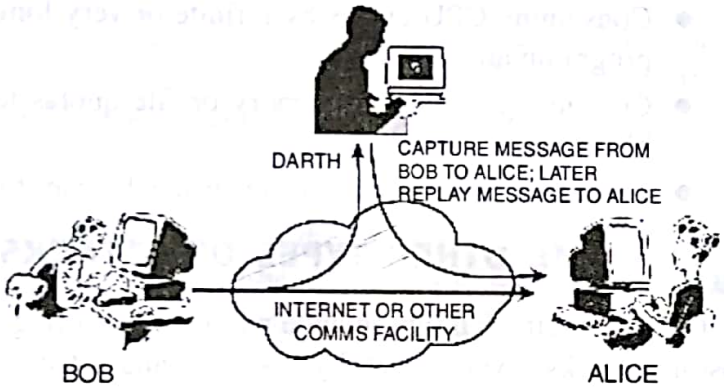


Fig. 1.9. *Reply Attack*

(c) Modification of Message

Cut and Paste is a type of message modification attack: the attacker removes a message from network traffic, alerts it and reinserts it. A cut and paste attack is an assault on the integrity of a security system in which the attacker substitutes a section of cipher text (encrypted text) with a different section that looks like (but is not the same as) the one removed. The substituted section appears to decrypt normally, along with the authentic sections, but results in plaintext (unencrypted text) that serves a particular purpose for the attacker. A typical use for a cut and paste attack is the modification of information on a customer order form for the purchase of goods or services over the web. The attacker modifies the form so that the victim's credit card number is sent to the vendor but other information—such as the attacker's chosen delivery address and the type of quantity of goods ordered—is pasted into the form from which the customer's valid information has been cut". The apparently unaltered form, assembled from a "cut-and –pasted" combination of valid and invalid data, is submitted to the vendor.

(d) Repudiation

The goal of a repudiation attack is to perform an authorized or unauthorized action and to eliminate any evidence that could prove the identity of the attacker. This type of attack is performed

by one of the two parties in communication: sender or receiver. The issue of repudiation is concerned with a user denying that he or she performed an action or initiated a transaction.

(e) Denial of Service

The goal of a denial-of-service attack is the loss of access by legitimate users to a server or to services. Generally speaking, denial-of-service attacks occur when a malicious user either disables critical services on a computer or consumes so many resources on a system that no resources are available for legitimate users. The resources that can be exhausted might include CPU cycles, disk space, memory, server connections, or network bandwidth, among others. Denial-of-service attacks include:

- Consuming CPU cycles by infinite or very long programmatic looping.
- Consuming excessive memory or file quotas to block legitimate use.
- Causing a crash, restart, or error mechanism to interfere with normal use.

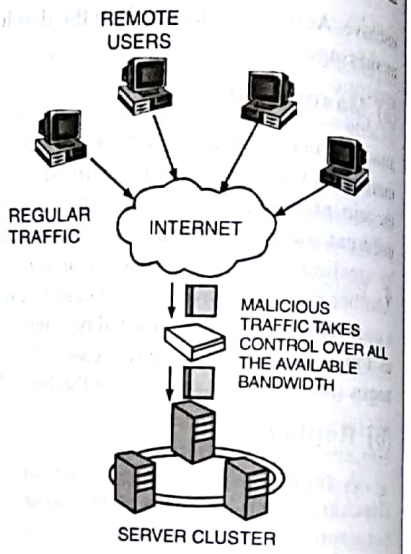


Fig. 1.10. *Denial of Service*

1.7. SOME OTHER TYPES OF ATTACKS

Classes of attack might include passive monitoring of communications, active network attacks, close-in attacks, exploitation by insiders, and attacks through the service provider. Information systems and networks offer attractive targets and should be resistant to attack from the full range of threat agents, from hackers to nation-states. A system must be able to limit damage and recover rapidly when attacks occur. There are various types of attacks. Some of them are described below:

1. Distributed Attack

A distributed attack requires that the adversary introduce code, such as a Trojan horse or back-door program, to a "trusted" component or software that will later be distributed to many other companies and users. Distribution attacks focus on the malicious modification of hardware or software at the factory or during distribution. These attacks introduce malicious code such as a back door to a product to gain unauthorized access to information or to a system function at a later date.

2. Insider Attack

An insider attack involves someone from the inside, such as a **disgruntled** employee, attacking the network. Insider attacks can be malicious or no malicious. Malicious insiders intentionally eavesdrop, steal, or damage information; use information in a fraudulent manner; or deny access to other authorized users. No malicious attacks typically result from carelessness, lack of knowledge, or intentional circumvention of security for such reasons as performing a task.

3. Close-in Attack

A close-in attack involves someone attempting to get physically close to network components, data, and systems in order to learn more about a network. Close-in attacks consist of regular

individuals attaining close physical proximity to networks, systems, or facilities for the purpose of modifying, gathering, or denying access to information. Close physical proximity is achieved through surreptitious entry into the network, open access, or both.

One popular form of close in attack is **social engineering**. In a social engineering attack, the attacker compromises the network or system through social interaction with a person, through an e-mail message or phone. Various tricks can be used by the individual to revealing information about the security of company. The information that the victim reveals to the hacker would most likely be used in a subsequent attack to gain unauthorized access to a system or network.

4. Phishing Attack

In phishing attack the hacker creates a fake web site that looks exactly like a popular site such as the SBI bank or Paypal. The phishing part of the attack is that the hacker then sends an e-mail message trying to trick the user into clicking a link that leads to the fake site. When the user attempts to log on with their account information, the hacker records the username and password and then tries that information on the real site.

5. Hijack Attack

In a hijack attack, a hacker takes over a session between you and another individual and disconnects the other individual from the communication. You still believe that you are talking to the original party and may send private information to the hacker by accident.

6. Spoof Attack

In a spoof attack, the hacker modifies the source address of the packets he or she is sending so that they appear to be coming from someone else. This may be an attempt to bypass your firewall rules.

7. Buffer Overflow

A buffer overflow attack is when the attacker sends more data to an application than is expected. A buffer overflow attack usually results in the attacker gaining administrative access to the system in a command prompt or shell.

8. Exploit Attack

In this type of attack, the attacker knows of a security problem within an operating system or a piece of software and leverages that knowledge by exploiting the vulnerability.

9. Password Attack

An attacker tries to crack the passwords stored in a network account database or a password-protected file. There are three major types of password attacks: a dictionary attack, a brute-force attack, and a hybrid attack. A dictionary attack uses a word list file, which is a list of potential passwords. A brute-force attack is when the attacker tries every possible combination of characters.

10. Sniffer Attack

A sniffer is an application or device that can read, monitor, and capture network data exchanges and read network packets. If the packets are not encrypted, a sniffer provides a full view of the data inside the packet. Even encapsulated (tunneled) packets can be broken open and read unless they are encrypted and the attacker does not have access to the key.

11. Snooping Attack

Snooping is similar to eavesdropping but is not necessarily limited to gaining access to data during its transmission. Snooping can include casual observance of an e-mail that appears on another's computer screen or watching what someone else is typing. More sophisticated snooping uses software programs to remotely monitor activity on a computer or network device.

Difference between Threat and Attack

Threat	Attack
A threat tends to be a promise of an attack to come.	An attack tends to be an act which is in process.
A potential occurrence—malicious or anything that may harm an asset.	An action taken to harm an asset.

Definitions

Cryptography : Cryptography is an art or can say a science of achieving security by encoding messages to make them non-readable.

Plain Text : This is the original message or data which we have to send.

Encryption : In cryptography, encryption is the process of transforming information (referred to as plaintext) using an algorithm (called cipher) to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key. The result of the process is encrypted information (in cryptography, referred to as ciphertext). Encryption is now commonly used in protecting information within many kinds of civilian systems.

Decryption : Decryption is the process of extracting the original information (plaintext) from the encrypted data (ciphertext). In many contexts, the word decryption also implicitly refers to the reverse process to make the encrypted information readable again (*i.e.* to make it unencrypted).

Cipher text : When a plaintext message is encoded using any suitable scheme, the resulting message is called as cipher text.

Cryptanalysis : Cryptanalysis is technique of decoding ciphertext into plaintext without knowing how they were initially converted from plaintext to ciphertext. In other words, Cryptanalysis is an art of breaking secret codes.

Cryptanalyst : Cryptanalyst is a person who attempts to break a cipher text message to obtain the original plain text message. In other words, Cryptanalyst is a person who performs Cryptanalysis.

Attack : To begin acting upon harmfully or destructively.

Vulnerability : A weakness that makes a threat possible.

Threat : It is an expression of intension to inflict evil, injury or damage the one that threatens.

Cryptology : The field encompassing both cryptography and cryptanalysis.

Cryptology = Cryptography + Cryptanalysis

SUMMARY

- Confidentiality, Integrity and Availability (known as the CIA triad) to be the core principles of information security.
- There are five security services—Authentication, Data integrity, Data confidentiality, Non-repudiation, Access control.

CHAPTER 2

Classical Encryption Techniques

2.1. INTRODUCTION TO CLASSICAL CIPHERS

Inside This Chapter

Cryptography began thousands of years ago. Until recent decades, it has been the story of what might be called classic cryptography—that is, of methods of encryption that use pen and paper, or perhaps simple mechanical aids.

Many classical ciphers were used by well-respected people, such as Julius Caesar and Napoleon, who created their own ciphers which were then popularly used. Many ciphers had their origins in the military and were used for transporting secret messages among people on the same side. Secret codes have been used for centuries! The first known cipher in history was developed by the Roman leader Julius Caesar. His code was very simple. In fact, you could probably crack it, if you took a bit of time. He just replaced one letter of the alphabet with another and it never changed. However, his enemies didn't catch on very quickly. A code was still a new idea!

As people became smarter about the idea of codes, harder ciphers were developed. An Italian, named Leon Battista Alberti, made a new invention, called a cipher wheel. This had two circles, both engraved with alphabet letters. When you matched each wheel in a certain way, a code could be both created and cracked. However, if the enemy didn't know where to match the wheel, you could hide some pretty good secrets, even if they had a similar wheel!

As time progressed, codes and ciphers have gotten more and more sophisticated. Technology began to be used to make more complicated codes. They have even been used for everyday people, who weren't spies. When the telegram was used to send messages, they charged by the word. You could write up to ten letters in a word for the same price. To cut costs, people made up codes. A group of letters meant a certain phrase. If you stop and think about it, we still use codes in this way today. Just think about the last text message you sent!

2.2. CATEGORIES OF CLASSICAL CIPHERS

There are two broad categories of classical cipher :

1. Substitution Ciphers
2. Transposition Ciphers

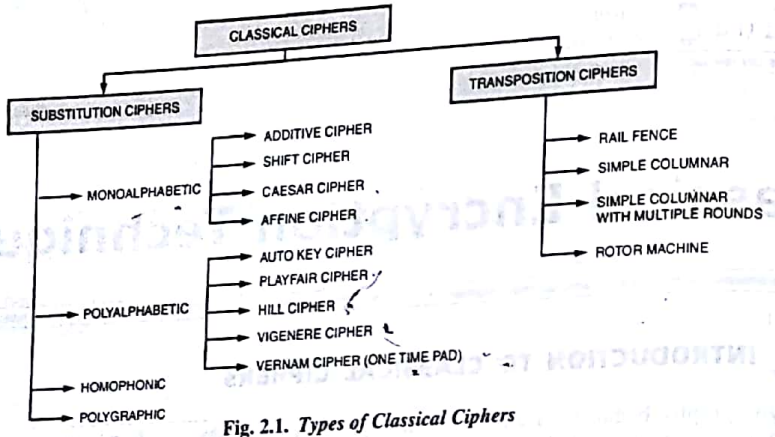


Fig. 2.1. *Types of Classical Ciphers*

Figure 2.1 shows the types of classical ciphers. There are other classical ciphers also but few of them are described here.

2.2.1. Substitution Ciphers

Substitution cipher involves replacing one or more entities (generally letters) in a message with one or more other entities. In Substitution Techniques, the characters of a plain text are replaced by other characters, numbers or symbols. For example, we can replace *A* with *E*, and letter *S* with *Z*. If the symbols are digits, we can replace 2 with 6 and 1 with 5.

There are several types of substitution cryptosystems:

(a) **Monoalphabetic substitution** involves replacing each letter in the message with another letter of the alphabet regardless of its position.

(b) **Polyalphabetic substitution** involves each occurrence of a character may have a different substitute character.

(c) **Homophonic substitution** makes it possible to have each letter of the plaintext message correspond to a possible group of other characters.

(d) **Polygraphic substitution** involves replacing a group of characters in the message with another group of characters.

(a) **Monoalphabetic Ciphers :** In Monoalphabetic Ciphers, a character or symbol in the plaintext is always changed to same character or symbol in the ciphertext regardless of its position in the text. For example, here a scheme for codifying message by replacing each alphabet with an alphabet three places down the line is shown.

C	R	Y	P	T	O	G	R	A	P	H	Y
F	U	B	S	W	R	J	U	D	S	K	B

Monoalphabetic ciphers are easy to break because they reflect the frequency data of the original alphabet.

Following are the types of Monoalphabetic ciphers :

1. Additive Cipher

The simplest monoalphabetic cipher is the additive cipher. This cipher is sometimes called a shift cipher and sometimes a Caesar cipher. In this cipher the plain text consists of lower case letters (*a* to *z*) and that the cipher text consists of upper case letters (*A* to *Z*). Each character (lowercase or uppercase) is assigned an integer from 0 to 25. The secret key *K* is also an integer between 0 and 25.

Encryption: C = (P + K) mod26	*Decryption: P= (C – K) mod26*

Additive cipher can be summarized as follows.

1. Read the plaintext and assign the integer from 0 to 25 to each letter of plaintext.
2. Add the given key *k* to each letter
3. Take the mod26 of the output of step 2.
4. Convert the output of step 3 into corresponding letter.

Example 2.1. *Encrypt the Plain text "hello" with key = 15 using additive cipher.*

Solution.

	h	e	l	l	o
	7	4	11	11	14
+	15	15	15	15	15
	22	19	26	26	29
Taking mod26	22	19	0	0	3
	W	T	A	A	D

So, the cipher text is – **"WTAAD"**

2. Shift Cipher

Historically, additive ciphers are called Shift ciphers. The reason is that the encryption algorithm can be interpreted as "Shift key characters down" and the decryption algorithm can be interpreted as "Shift key characters up". For example, if the key = 15, the encryption algorithm shifts 15 characters down (towards the end of the alphabet). The decryption algorithm shifts 15 characters up (towards the beginning of the alphabet).

3. Caesar Cipher

The Caesar cipher involves replacing each letter of the alphabet with the letter standing three places further down the alphabet (Caesar used a key of 3).

Encryption: C = (P+3) mod26	*Decryption: P= (C-3) mod26*

Caesar cipher can be summarized as follows.

1. Read the plaintext and assign the integer from 0 to 25 to each letter of plaintext.
2. Add the key *k* = 3 to each letter
3. Take the mod26 of the output of step 2.
4. Convert the output of step 3 into corresponding letter.

Example 2.2. *Encrypt the Plain text "hello" using Caesar cipher*

Solution. Here $k = 3$

	h	e	l	l	o
	7	4	11	11	14
+	3	3	3	3	3
	10	7	14	14	17
Taking mod26	10	7	14	14	17
	K	H	O	O	R

So, the cipher text is – **"KHOOR"**

Example 2.3. *Encrypt the given plain text using caeser cipher.*

Plain Text: m e e t m e a f t e r t h e t o g a p a r t y

Solution.

Cipher Text: P H H W P H D I W H U W K H W R J D S D U W B

Note that the alphabet is wrapped around; so that the letter following Z is A. we can define the transformation by listing all possibilities, as follows:

$$25 + 3 = 28 \bmod 26 = 2 \text{ means C}$$

Attacks on Additive Cipher, Shift Cipher and Caesar Cipher

1. Brute-Force Attack

An attack on a cipher text, wherein the attacker attempts to use all possible permutations and combinations is called a Brute-force attack. 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.

The key domain of the additive cipher is very small; there are only 26 keys. However, one of the keys, zero, is useless (the ciphertext is same as the plaintext). This leaves only 25 possible keys. So it is very easy to break the cipher.

2. Frequency Analysis Attack

It is easy to spot a Caesar cipher from frequency analysis of the ciphertext. Patterns occur in the letter frequencies of any language. Here are the patterns for English shown in Table given below.

Letters	Frequency	Letters	Frequency	Letters	Frequency
A	1111111	J		S	111111
B	1	K		T	111111111
C	111	L	1111	U	111
D	1111	M	111	V	1
E	111111111111	N	1111111	W	11
F	111	O	1111111	X	
G	11	P	111	Y	11
H	1111	Q		Z	
I	1111111	R	11111111		

Because a Caesar cipher just translates the letters of the plaintext alphabet to the right, it translates to the right the frequency patterns we expect with plaintext. For any pattern there are some points for elementary cryptanalysis:

1. *a*, *e*, and *i* are all high frequency letters (at the beginning of the plaintext alphabet), and they are equally spaced (four letters apart) with *e* the most frequent.
2. *n* and *o* form a high frequency pair (near the middle of the plaintext alphabet).
3. *r*, *s*, and *t* form a high frequency triple (about 2/3 of the way through the plaintext alphabet).
4. *j* and *k* form a low frequency pair (just before the middle of the plaintext alphabet).
5. *u*, *v*, *w*, *x*, *y* and *z* form a low frequency six-letter string (at the end of the plaintext alphabet).

For Example : For a caesar cipher of additive key 5 the expected frequencies are

Letters	Frequency	Letters	Frequency	Letters	Frequency
A	1	J	111111111111	S	11111111
B	11	K	111	T	1111111
C		L	11	U	111
D	11	M	1111	V	
E		N	1111111	W	11111111
F	1111111	O		X	111111
G	1	P		Y	111111111
H	111	Q	1111	Z	111
I	1111	R	111		

Notice that the usual frequencies have just shifted 5 places further in the alphabet.

(*a*) Instead of having *a*, *e*, and *i* be all high frequency letters spaced four letters apart with *e* the most frequent, we now have that *F*, *J*, and *N* have that property with *J* being the most frequent letter.

(*b*) Instead of *n* and *o* forming a high frequency pair (near the middle of the plaintext alphabet), we have that *S* and *T* form such a pair.

(*c*) Instead of *r*, *s*, and *t* forming a high frequency triple, we have that *W*, *X*, and *Y* form such a triple.

(*d*) Instead of *j* and *k* forming a low frequency pair, we now have that *O* and *P* form such a pair. Instead of *u*, *v*, *w*, *x*, *y*, and *z* forming a low frequency six-letter string (at the end of the plaintext alphabet), we now have that *Z*, *A*, *B*, *C*, *D*, and *E* form such a string.

Such shifts of frequency patterns should be easy to spot. They identify a Caesar cipher, and they exhibit the shift – the key.

4. Affine Cipher

The **Affine cipher** is a type of monoalphabetic substitution cipher, wherein each letter in an alphabet is mapped to its numeric equivalent and then encrypted using a simple mathematical function. It inherits the weaknesses of all substitution ciphers. A simple Caesar shift is a type of affine cipher, wherein each letter is enciphered with the function $(x + b) \bmod 26$, where b is the magnitude of the shift. The Caesar cipher is the Affine cipher when $a = 1$ since the encrypting function simply reduces to a linear shift.

Example 2.2. *Encrypt the Plain text "hello" using Caesar cipher*

Solution. Here $k = 3$

	h	e	l	l	o
	7	4	11	11	14
+	3	3	3	3	3
	10	7	14	14	17
Taking mod26	10	7	14	14	17
	K	H	O	O	R

So, the cipher text is – **"KHOOR"**

Example 2.3. *Encrypt the given plain text using caeser cipher.*

Plain Text: m e e t m e a f t e r t h e t o g a p a r t y

Solution.

Cipher Text: P H H W P H D I W H U W K H W R J D S D U W B

Note that the alphabet is wrapped around; so that the letter following Z is *A*. we can define the transformation by listing all possibilities, as follows:

$$25 + 3 = 28 \bmod 26 = 2 \text{ means C}$$

Attacks on Additive Cipher, Shift Cipher and Caesar Cipher

1. Brute-Force Attack

An attack on a cipher text, wherein the attacker attempts to use all possible permutations and combinations is called a Brute-force attack. 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.

The key domain of the additive cipher is very small; there are only 26 keys. However, one of the keys, zero, is useless (the ciphertext is same as the plaintext). This leaves only 25 possible keys. So it is very easy to break the cipher.

2. Frequency Analysis Attack

It is easy to spot a Caesar cipher from frequency analysis of the ciphertext. Patterns occur in the letter frequencies of any language. Here are the patterns for English shown in Table given below.

Letters	Frequency	Letters	Frequency	Letters	Frequency
A	ıııııı	J		S	ıııı!
B	ı	K		T	ııııııı
C	ıı	L	ııı	U	ıı
D	ııı	M	ıı	V	ı
E	ııııııııııı	N	ıııııı	W	ıı
F	ıı	O	ııııı	X	
G	ı	P	ıı	Y	ıı
H	ııı	Q		Z	
I	ııııı	R	ıııııı		

Because a Caesar cipher just translates the letters of the plaintext alphabet to the right, it translates to the right the frequency patterns we expect with plaintext. For any pattern there are some points for elementary cryptanalysis:

1. *a*, *e*, and *i* are all high frequency letters (at the beginning of the plaintext alphabet), and they are equally spaced (four letters apart) with *e* the most frequent.
2. *n* and *o* form a high frequency pair (near the middle of the plaintext alphabet).
3. *r*, *s*, and *t* form a high frequency triple (about 2/3 of the way through the plaintext alphabet).
4. *j* and *k* form a low frequency pair (just before the middle of the plaintext alphabet).
5. *u*, *v*, *w*, *x*, *y* and *z* form a low frequency six-letter string (at the end of the plaintext alphabet).

For Example : For a caesar cipher of additive key 5 the expected frequencies are

Letters	Frequency	Letters	Frequency	Letters	Frequency
A	ı	J	ııııııııııı	S	ııııııı
B	ıı	K	ııı	T	ıııııı
C		L	ıı	U	ııı
D	ıı	M	ııı	V	
E		N	ıııııı	W	ıııııı
F	ıııııı	O		X	ııııı
G	ı	P		Y	ııııııı
H	ııı	Q	ııı	Z	ııı
I	ııı	R	ııı		

Notice that the usual frequencies have just shifted 5 places further in the alphabet.

(*a*) Instead of having *a*, *e*, and *i* be all high frequency letters spaced four letters apart with *e* the most frequent, we now have that *F*, *J*, and *N* have that property with *J* being the most frequent letter.

(*b*) Instead of *n* and *o* forming a high frequency pair (near the middle of the plaintext alphabet), we have that *S* and *T* form such a pair.

(*c*) Instead of *r*, *s*, and *t* forming a high frequency triple, we have that *W*, *X*, and *Y* form such a triple.

(*d*) Instead of *j* and *k* forming a low frequency pair, we now have that *O* and *P* form such a pair. Instead of *u*, *v*, *w*, *x*, *y*, and *z* forming a low frequency six-letter string (at the end of the plaintext alphabet), we now have that *Z*, *A*, *B*, *C*, *D*, and *E* form such a string.

Such shifts of frequency patterns should be easy to spot. They identify a Caesar cipher, and they exhibit the shift – the key.

4. Affine Cipher

The **Affine cipher** is a type of monoalphabetic substitution cipher, wherein each letter in an alphabet is mapped to its numeric equivalent and then encrypted using a simple mathematical function. It inherits the weaknesses of all substitution ciphers. A simple Caesar shift is a type of affine cipher, wherein each letter is enciphered with the function $(x + b) \bmod 26$, where b is the magnitude of the shift. The Caesar cipher is the Affine cipher when $a = 1$ since the encrypting function simply reduces to a linear shift.

Affine cipher can be summarized as follows.

1. Read the plaintext and assign the integer from 0 to 25 to each letter of plaintext.
2. Find the value of ciphertext by given mathematical function.
3. Take the mod26 of the output of step 2.
4. Convert the output of step 3 into corresponding letter.

Example 2.4. *Encrypt the given plain text using caesar cipher*

Plaintext: AFFINE CIPHER

The encryption function is $y = E(x) = 5x + 8$.

Solution. The first step in encrypting the message is to write the numeric values of each letter. Now, take each value of x, and solve the first part of the equation, $(5x + 8)$. After finding the value of $(5x + 8)$ for each character, take the remainder when dividing the result of $(5x + 8)$ by 26. The final step in encrypting the message is to look up each numeric value in the table for the corresponding letters. In this example, the encrypted text would be IHHWVCSWFRCP. The table below shows the completed table for encrypting a message in the Affine cipher.

plaintext:	A	F	F	I	N	E	C	I	P	H	E	R
x:	0	5	5	8	13	4	2	8	15	7	4	17
$5x + 8$	8	33	33	48	73	28	18	48	83	43	28	93
$(5x + 8)$(mod 26)	8	7	7	22	21	2	18	22	5	17	2	15
ciphertext:	I	H	H	W	V	C	S	W	F	R	C	P

Decryption

In the affine cipher the letters of an alphabet of size m are first mapped to the integers in the range 0 to $m - 1$. It then uses modular arithmetic to transform the integer that each plaintext letter corresponds to into another integer that corresponds to a cipher text letter. The encryption function for a single letter is

$$E(x) = (ax + b) \bmod m$$

where modulus m is the size of the alphabet and a and b are the key of the cipher. The value a must be chosen such that a and m are co-prime. The decryption function is

$$D(x) = a^{-1}(x - b) \bmod m$$

where a^{-1} is the modular multiplicative inverse of a modulo m. That is, it satisfies the equation

$$1 = a.a^{-1} \bmod m$$

It can be shown as follows that decryption function is the inverse of the encryption function,

$$\begin{aligned} D(E(x)) &= a^{-1}(E(x) - b) \bmod m \\ &= a^{-1}(((ax + b) \bmod m) - b) \bmod m \\ &= a^{-1}(ax + b - b) \bmod m \\ &= a^{-1} ax \bmod m \\ &= x \bmod m \end{aligned}$$

For the given example we must find out the a^{-1} of a to compute the decrypted text of Cipher text. So, $a^{-1}.a \bmod m = 1$.

21 is the inverse of 5 mod 26.

Cipher text	I	H	H	W	V	C	S	W	F	R	C	P
X:	8	7	7	22	21	2	18	22	5	17	2	15
$21(E(x) - 8)$	0	−21	−21	294	273	−126	210	294	−63	189	−126	147
$21(y - 8)) \bmod 26$:	0	5	5	8	13	4	2	8	15	7	4	17
Plain Text	A	F	F	I	N	E	C	I	P	H	E	R

(b) Polyalphabetic Ciphers

In polyalphabetic ciphers, each occurrence of a character may have a different substitute. The relationship between a character in the plaintext to a character in the ciphertext is one-to-many. The main features of polyalphabetic substitution cipher are:

1. A set of related monoalphabetic rules is used.
2. It uses a key that determines which rule is used for which transformation.

For example, "a" could be enciphered as "E" in the beginning of the text, but as "N" at the middle. Polyalphabetic cipher has the advantage of hiding the letter frequency of the underlying language.

To create a polyalphabetic cipher, we make each ciphertext character dependent on both the plaintext character and the position of the plaintext character in the message. In other words, we need to have a key stream $k = (k_1, k_2, k_3 \ldots)$ in which k_i is used to encipher the i^{th} character in the plaintext to create the i^{th} character in the ciphertext.

Following are the types of Monoalphabetic ciphers :

1. Auto Key Cipher

One of the polyalphabetic ciphers is the Autokey cipher. In this cipher, the key is a stream of subkeys, in which each subkey is used to encrypt the corresponding character in the plain text. The first subkey is predetermined and secret. The second subkey is the value of the first plain text character (between 0 and 25). The third subkey is the value of the second plain text character. And so on.

Encryption: $C_i = (P_i + k_i) \bmod 26$	*Decryption:* $P_i = (C_i - K_i) \bmod 26$

The name of the cipher, autokey, implies that the subkeys are automatically created from the plaintext characters during the encryption process.

Auto key cipher can be summarized as follows :

1. Each character in the plain text is first replaced by an integer value from 0 to 25.
2. The first subkey (given) is added to first plaintext character to create the first cipher text character.
3. The second subkey will be the first plaintext character, third subkey will be second plaintext character and so on.
4. Repeat the step 2 for remaining plaintext characters.
5. Take the mod of the output of step 4.
6. Convert the output of step 5 into corresponding letter.

Example 2.5. *Consider the, Plain text – attack is today*
Initial secret key – 12.

Solution.

Plain text:		*a*	*t*	*t*	*a*	*c*	*k*	*i*	*s*	*t*	*o*	*d*	*a*	*y*
P's value:		00	19	19	00	02	10	08	18	19	14	03	00	24
Key:	+	12	00	19	19	00	02	10	08	18	19	14	03	00
		12	19	38	19	02	12	18	26	37	33	17	03	24
C's value:		12	19	12	19	02	12	18	00	11	07	17	03	24
(mod 26)														
Ciphertext:		M	T	M	T	C	M	S	A	L	H	R	D	Y

Ciphertext- MTMTCMSALHRDY

In this cipher, enciphering is done character by character as shown above. This is a polyalphabetic cipher because the three occurrences of "a" in the plain text are encrypted differently.

Attack on Auto key cipher

The autokey cipher definitely hides the single-letter frequency statistics of the plaintext. The first subkey can be only one of the 25 values (1 to 25). So Brute-force attack can easily break it. So we need polyalphabetic ciphers because of having large key domains.

2. Playfair Cipher

Playfair cipher is a polyalphabetic cipher used by British Army during World War I. This cipher was actually invented by British Scientist **Sir Charles Wheatstone** in 1854, but it is named after his friend Baron Playfair.

The secret key in this cipher is made of 25 alphabet letters arranged in a 5 * 5 matrix. (Usually letters I and J count as one letter). Different arrangement of the letters in the matrix can create many different secret keys.

Playfair cipher can be summarized as follows :

1. Make the key matrix/Secret key from the given keyword by applying following rules.
   - (*a*) The matrix (5*5) will be filled up from left to right and top to bottom.
   - (*b*) First fill the spaces in the matrix with the letters of the keyword (Given) by dropping any duplicate letters.
   - (*c*) Fill the remaining spaces with the rest of the letters of the alphabet in order. (Usually consider I and J as one letter).
2. Write the plaintext in pairs (two letters in a pair), if both the letters in a pair are same or only one letter is left, then add an *X* after the first letter.
3. Now to find out ciphertext for each pair plaintext, apply the following rules.

Case 3(*a*) If the 2 letters in a pair are located in the same row of the secret key, the corresponding encrypted character for each letter is the next letter to the right in the same row.

Case 3(*b*) If the 2 letters in a pair are located in the same column of the secret key, the corresponding encrypted character for each letter is the letter beneath it in the same column.

Case 3(*c*) If the letters are not in the same row or column, replace them with the letter on the same row respectively, but at the other pair of corners of the rectangle defined by the original pair.

Example 2.6. *Let the keyword be "playfair", and the plaintext is "hide the gold".*

Solution. The Secret key will be

Plain text :- hide the gold
In Pair :- hi de th eg ol dx
Ciphertext:- EB IM QM GH VR CZ
Ciphertext = EBIMQMGHVRCZ

Encryption Process:

(*i*) For *hi* pair, case 3(*c*) will be apply. So for *h* letter corresponding ciphertext will be E (intersection of H's row and *i*'s column) and for *i* letter corresponding ciphertext will be B (intersection of *i*'s row and *h*'s column)

Secret Key =

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

(*ii*) For *de* pair also case 3(*c*) apply again.

(*iii*) For *th* pair also case 3(*c*) will apply.

(*iv*) For pair *eg*, case 3(*a*) will be apply. So for *e* letter corresponding ciphertext will be G (next letter in same row) and for *g* letter corresponding ciphertext will be H (next letter in same row).

(*v*) For *ol* pair, case 3(*b*) will be apply. So for *o* letter corresponding ciphertext will be V (beneath letter in same column) and for *l* letter corresponding ciphertext will be R (beneath letter in same column).

(*vi*) For *dx* pair also case 3(*c*) will apply.

In same manner ciphertext for others are also found.

Decryption Process :

Cipher text is EBIMQMGHVRCZ

As playfair cipher encrypted into pairs we must again separate them into pairs. Same secret key will be used for decryption.

Cipher text : EB IM QM GH VR CZ

(*i*) For *EB* pair, the case 3(*c*) will be apply. So, for '*E*' corresponding text will be *H* (intersection of *E*'s row and *B*'s column) and for '*B*' corresponding text will be *I* (intersection of *B*'s row and *I*'s column)

(*ii*) For '*IM*' pair case 3(*c*) apply again.

(*iii*) For '*QM*' pair again case 3(*c*) will apply.

(*iv*) For *GH* pair, the reverse of case 3(*a*) will be apply. So, for '*G*' corresponding text will be *E* and for '*H*' corresponding text will be *G*.

Example 2.5. *Consider the, Plain text – attack is today*
Initial secret key – 12.

Solution.

Plain text:		*a*	*t*	*t*	*a*	*c*	*k*	*i*	*s*	*t*	*o*	*d*	*a*	*y*
P's value:		00	19	19	00	02	10	08	18	19	14	03	00	24
Key:	+	12	00	19	19	00	02	10	08	18	19	14	03	00
		12	19	38	19	02	12	18	26	37	33	17	03	24
C's value:		12	19	12	19	02	12	18	00	11	07	17	03	24
(mod 26)														
Ciphertext:		M	T	M	T	C	M	S	A	L	H	R	D	Y

Ciphertext- MTMTCMSALHRDY

In this cipher, enciphering is done character by character as shown above. This is a polyalphabetic cipher because the three occurrences of "a" in the plain text are encrypted differently.

Attack on Auto key cipher

The autokey cipher definitely hides the single-letter frequency statistics of the plaintext. The first subkey can be only one of the 25 values (1 to 25). So Brute-force attack can easily break it. So we need polyalphabetic ciphers because of having large key domains.

2. Playfair Cipher

Playfair cipher is a polyalphabetic cipher used by British Army during World War I. This cipher was actually invented by British Scientist **Sir Charles Wheatstone** in 1854, but it is named after his friend Baron Playfair.

The secret key in this cipher is made of 25 alphabet letters arranged in a 5 * 5 matrix. (Usually letters I and J count as one letter). Different arrangement of the letters in the matrix can create many different secret keys.

Playfair cipher can be summarized as follows :

1. Make the key matrix/Secret key from the given keyword by applying following rules.
   - (*a*) The matrix (5*5) will be filled up from left to right and top to bottom.
   - (*b*) First fill the spaces in the matrix with the letters of the keyword (Given) by dropping any duplicate letters.
   - (*c*) Fill the remaining spaces with the rest of the letters of the alphabet in order. (Usually consider I and J as one letter).
2. Write the plaintext in pairs (two letters in a pair), if both the letters in a pair are same or only one letter is left, then add an *X* after the first letter.
3. Now to find out ciphertext for each pair plaintext, apply the following rules.

Case 3(*a*) If the 2 letters in a pair are located in the same row of the secret key, the corresponding encrypted character for each letter is the next letter to the right in the same row.

Case 3(*b*) If the 2 letters in a pair are located in the same column of the secret key, the corresponding encrypted character for each letter is the letter beneath it in the same column.

Case 3(*c*) If the letters are not in the same row or column, replace them with the letter on the same row respectively, but at the other pair of corners of the rectangle defined by the original pair.

Example 2.6. *Let the keyword be "playfair", and the plaintext is "hide the gold".*

Solution. The Secret key will be

Plain text :- hide the gold

In Pair :- hi de th eg ol dx

Ciphertext:- EB IM QM GH VR CZ

Ciphertext = EBIMQMGHVRCZ

Encryption Process:

(*i*) For *hi* pair, case 3(*c*) will be apply. So for *h* letter corresponding ciphertext will be E (intersection of H's row and *i*'s column) and for *i* letter corresponding ciphertext will be B (intersection of *i*'s row and *h*'s column)

Secret Key =

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

(*ii*) For *de* pair also case 3(*c*) apply again.

(*iii*) For *th* pair also case 3(*c*) will apply.

(*iv*) For pair *eg*, case 3(*a*) will be apply. So for *e* letter corresponding ciphertext will be G (next letter in same row) and for *g* letter corresponding ciphertext will be H (next letter in same row).

(*v*) For *ol* pair, case 3(*b*) will be apply. So for *o* letter corresponding ciphertext will be V (beneath letter in same column) and for *l* letter corresponding ciphertext will be R (beneath letter in same column).

(*vi*) For *dx* pair also case 3(*c*) will apply.

In same manner ciphertext for others are also found.

Decryption Process :

Cipher text is EBIMQMGHVRCZ

As playfair cipher encrypted into pairs we must again separate them into pairs. Same secret key will be used for decryption.

Cipher text : EB IM QM GH VR CZ

(*i*) For *EB* pair, the case 3(*c*) will be apply. So, for *'E'* corresponding text will be *H* (intersection of *E*'s row and *B*'s column) and for *'B'* corresponding text will be *I* (intersection of *B*'s row and *I*'s column)

(*ii*) For *'IM'* pair case 3(*c*) apply again.

(*iii*) For *'QM'* pair again case 3(*c*) will apply.

(*iv*) For *GH* pair, the reverse of case 3(*a*) will be apply. So, for *'G'* corresponding text will be *E* and for *'H'* corresponding text will be *G*.

(v) For '*VR*' pair, the reverse of case 3(*b*) will be apply. So, for '*V*' corresponding text will be *O* and for '*R*' corresponding text will be '*L*'.

(vi) For '*CZ*' pair case 3(*c*) apply again

Plaintext : HI DE TH EG OL DX and X is used to make the pair then discard it and the message will be "hide the gold".

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

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

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

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

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

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

Note: However, there are other methods. The letters can be put into the key square column-by-column, in a clockwise spiral, or in an anticlockwise spiral:

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

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

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

It is also possible to begin with the first letter in one of the other four corners. For example,

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

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

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

Altogether there are 32 ways in which the rest of the alphabet can be added to the keyword and the 25 letters placed in the key square.

Attacks on the Playfair cipher

(1) Brute force attack on a Playfair cipher is very difficult, since the size of the key domain is 25!

(2) Also there are 26 * 26 = 676 diagrams so that identification of individual diagram is more difficult.

(3) However, the frequencies of diagrams are preserved, so a cryptanalyst can use a cipher text only attack based on the diagram frequency test to find the key.

3. Hill Cipher

Hill cipher is a polyalphabetic cipher developed by the Mathematician **Lester S. Hill** in 1929. In this cipher the plain text is divided into equal size blocks. The blocks are encrypted one at a time in such a way that each character in the block contributes to the encryption of other characters in the block. For this reason, the Hill cipher belongs to a category of ciphers called Block ciphers.

The encryption algorithm takes m successive plaintext letters and substitutes for them m cipher text letters. The substitution is determined by m linear equations in which each character is assigned a numerical value ($a = 0$ to $z = 25$). For $m = 3$, the system can be described as follows:

$$\begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} \bmod 26$$

$$C = KP \bmod 26$$

where C and P are column vectors of length 3, representing the plain text and cipher text, and K is a 3 * 3 matrix, representing the encryption key. Operation is performed mod 26.

In order to decrypt, we turn the ciphertext back into a vector, then simply multiply by the inverse matrix of the key matrix.

$$P = K^{-1} C \bmod 26$$

NOTE

*The size of plaintext taken for encryption depend on given secret key, if key is given of length 16 then we can make secret key matrix of size 4*4. So that plaintext will be of size 4*1.*

Example 2.7. *Suppose the plain text to be sent is 'AT' and the key is 'FDDC'.*

So, $P = \begin{pmatrix} 0 \\ 19 \end{pmatrix}$ *and* $K = \begin{pmatrix} 5 & 3 \\ 3 & 2 \end{pmatrix}$.

Solution. For Encryption

$$C = KP \bmod 26$$

$$C = \begin{pmatrix} 5 & 3 \\ 3 & 2 \end{pmatrix} \begin{pmatrix} 0 \\ 19 \end{pmatrix} \bmod 26$$

$$C = \begin{pmatrix} 5\times0+3\times19 \\ 3\times0+2\times19 \end{pmatrix} \bmod 26$$

$$C = \begin{pmatrix} 57 \\ 38 \end{pmatrix} \bmod 26 = \begin{pmatrix} 5 \\ 12 \end{pmatrix} = \text{FM}$$

So, Cipphertext (C) = FM

For Decryption

Suppose C is FM, now find plaintext

Using equation $\boxed{P = K^{-1}C \bmod 26}$

First find K^{-1} $\quad K^{-1} = \dfrac{\text{Adj (K)}}{\det(\text{K})}$

So $\quad K^{-1} = \dfrac{\begin{pmatrix} 2 & -3 \\ -3 & 5 \end{pmatrix}^T}{|5\times2-3\times3|} = \begin{bmatrix} 2 & -3 \\ -3 & 5 \end{bmatrix}$

Now $\quad P = \begin{pmatrix} 2 & -3 \\ -3 & 5 \end{pmatrix}\begin{pmatrix} 5 \\ 12 \end{pmatrix} \bmod 26 = \begin{pmatrix} 0 \\ 19 \end{pmatrix} \bmod 26 = AT$

So, plaintext (P) = AT

Attacks on Hill Cipher

(1) A brute force attack is extremely difficult because the key is an $m \times m$ matrix. Each entry in the matrix can have one of the 26 values. This means size of the key is $26^{m \times m}$

(2) It does not preserve the statistics of plaintext. As frequency analysis on single letters, diagrams or trigrams is rare.

4. Vigenere Cipher

One interesting kind of polyalphabetic cipher was designed by **Blaise De Vigenere**, a sixteenth century French Mathematician. A Vigenere cipher uses a different strategy to create the key stream. The key stream is a repetition of an initial secret key stream of length m, where we have $1<=m<=26$.

$P = P_1P_2P_3\ldots \qquad C = C_1C_2C_3\ldots \qquad K = [(k_1,k_2,\ldots,k_m),(k_1,k_2,\ldots,k_m),\ldots]$

Encryption: $C_i = P_i + k_i$ $\qquad$ *Decryption:* $P_i = C_i - k_i$

Vigenere cipher is the simplest best known example of Polyalphabetic cipher. In this, 26 Caesar ciphers make up the monoalphabetic substitution rules. Here, a shifting mechanism is used from count 0 to 25. For every character of plain text, there is a key.

NOTE

One important difference between the Vigenere key stream and the other polyalphabetic ciphers is that the Vigenere key stream does not depend on the plaintext characters; it depend only on the position of the character in the plaintext.

Example 2.8. *Encrypt the message "She is listening" using keyword "PASCAL".*

Solution. Write the PASCAL into integer form as (15, 0, 18, 2, 0, 11), this is the initial key stream. Now repeat this initial key stream as needed.

Plaintext:	*S*	*h*	*e*	*i*	*s*	*l*	*i*	*s*	*t*	*e*	*n*	*i*	*n*	*g*
P's values:	18	07	04	08	18	11	08	18	19	04	13	08	13	06
Key stream:	15	00	18	02	00	11	15	00	18	02	00	11	15	00
+	33	07	22	10	18	22	23	18	37	06	13	19	28	06
C's values:	07	07	22	10	18	22	23	18	11	6	13	19	02	06
(mod 26)														
Ciphertext:	H	H	W	K	S	W	X	S	L	G	N	T	C	G

So CT = HHWKSWXSLGNTCG

Table 2.1. Look Up Table for Vigenere Cipher

	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

Cryptanalysis of Vigenere Cipher

Vigenere cipher do not preserve the frequency of characters. The cryptanalysis consists of two parts: finding the length of the key and finding the key itself.

1. Several methods have been devised to find the length of the key. In the so-called **kasiski test**, the cryptanalyst searches for repeated text segments of at least three characters, in the ciphertext. The cryptanalyst assumes that d/m where m is the key length. If more repeated segments can be found in distances $d_1, d_2, \ldots, d_n$, then gcd $(d_1, d_2, \ldots, d_n)/m$. This is logical because if two characters are the same and are $k*m$ ($k = 1, 2, \ldots$) characters apart in the plaintext, they are the same and $k*m$ characters apart in the ciphertext. Cryptanalyst uses segments of atleast three characters to avoid the cases where the characters in the key are not distinct.
2. After the length of the key has been found, the cryptanalyst uses the idea. She divides the ciphertext into m different pieces and applies the method used to cryptanalyze the additive cipher, including frequency attack. Each ciphertext piece can be decrypted and put together to create the whole plaintext. In other words, the whole ciphertext does not preserve the single letter frequency of the plaintext, but each piece does.

5. Vernam Cipher (One Time Pad)

An Army Signal Corp officer, Joseph Mauborgne, proposed an improvement to the Vernam cipher that yields the ultimate in security. Mauborgne suggested using a random key that is as long as the message, so that the key need not be repeated. In addition the key is to be used to encrypt and decrypt a single message, and then is discarded. Each new message requires a new key of the same length as the new message. Such a scheme, known as a **one-time pad**, is unbreakable. It produces random output that bears no statistical relationship to the plaintext.

The algorithm is as follows:

1. Treat each plain text alphabet as a number in an increasing sequence, *i.e.* $A = 0$, $B = 1$, ... $Z = 25$.
2. Do the same for each character to the input cipher text.
3. Add each number corresponding to the plain text alphabet to the corresponding input cipher text alphabet number.
4. If the sum thus produced is greater than 26, subtract 26 from it.
5. Translate each number of the sum back to the corresponding alphabet. This gives the output cipher text.

Example 2.9. *Encrypt plain text "how are you" using one time pad "NCBTZQARX".*
Solution.

Plain Text:	H	O	W	A	R	E	Y	O	U
P's Value:	7	14	22	0	17	4	24	14	20
One-time Pad:	13	2	1	19	25	16	0	17	23
	N	C	B	T	Z	Q	A	R	X
Total	20	16	23	19	42	20	24	31	43
C's Value (mod 26)	20	16	23	19	16	20	24	5	17
Cipher Text:	U	Q	X	T	Q	U	Y	F	R

It should be clear that since the one-time pad is discarded after a single use, this technique is highly secure and suitable for small plain text message, but is clearly impractical for large messages. The Vernam Cipher was first implemented at **AT&T** with the help of a device called as the Vernam Machine.

The one-time pad offers complete security but, has two fundamental difficulties:

(1) There is the practical problem of making large quantise of random keys. Any heavily used system might require millions of random characters on a regular basis. Supplying truly random characters in this volume is a significant task.

(2) Even more daunting is the problem of key distribution and protection. For every message to be sent, a key of equal is needed by both sender and receiver. Thus a mammoth key distribution problem exists.

Because of these difficulties, the one-time pad is of limited utility, and is useful primarily for low-bandwidth channels requiring very high security.

(c) Homophonic Substitution Cipher

The Homophonic Substitution Cipher involves replacing each letter with a variety of substitutes, the number of potential substitutes being proportional to the frequency of the letter.

For example: The letter '*a*' accounts for roughly 8% of all letters in English, so we assign 8 symbols to represent it. Each time an '*a*' appears in the plaintext it is replaced by one of the 8 symbols chosen at random, and so by the end of the encipherment each symbol constitutes roughly 1% of the ciphertext.

The letter '*b*' accounts for 2% of all letters and so we assign 2 symbols to represent it. Each time '*b*' appears in the plaintext either of the two symbols can be chosen, so each symbol will also constitute roughly 1% of the ciphertext.

This process continues throughout the alphabet, until we get to 'z', which is so rare that it has only one substitute. Since more than 26 characters will be required in the ciphertext alphabet, various solutions are employed to invent larger alphabets. Perhaps the simplest is to use a numeric substitution 'alphabet'. Another method consists of simple variations on the existing alphabet; uppercase, lowercase, upside down, etc.

(d) Polygraphic Substitution Cipher

In a polygraphic substitution cipher, plaintext letters are substituted in larger groups, instead of substituting letters individually.

The first advantage is that the frequency distribution is much flatter than that of individual letters (though not actually flat in real languages; for example, 'TH' is much more common than 'XQ' in English). Second, the larger number of symbols require correspondingly more ciphertext to productively analyze letter frequencies. To substitute *pairs* of letters would take a substitution alphabet 676 symbols long ($(26)^2 = 676$).

2.2.2. Transposition Techniques

In classical cryptography, a transposition cipher changes the position of the symbols rather than substitute one symbol for another as in substitution cipher. That is, the order of the characters is changed. To decrypt the reverse is done.

Following are the types of Transposition ciphers :

1. Rail Fence Technique

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.

The **Rail fence technique** is an example of transposition. It uses a simple algorithm as shown:

1. Write down the plain text message as sequence of diagonals.
2. Read the plain text written in step 1 as a sequence of rows.

Example 2.10. *Plain Text: Come home tomorrow.*

Solution. Write the plaintext as:

c	*m*	*h*	*m*	*t*	*m*	*r*	*o*
o	*e*	*o*	*e*	*o*	*o*	*r*	*w*

Now read row wise.

Cipher Text: CMHMTMROOEOEOORW.

Cryptanalysis of Rail Fence Technique

This technique is quite simple to break into. It has very little sophistication built in.

2. Simple Columnar Transposition Technique

The algorithm of this is as follows:

1. Write the plain text message row by row in a rectangle of predefined size.
2. Read the message column by column. It can be read by given permutation key.
3. The message thus obtained is the cipher text message.

Example 2.11. ***Plain Text:*** *Come home tomorrow*

In cipher, this can be written as follows:

Permutation Key:

4	6	1	2	5	3

Solution. We write the message in the rectangle row by row (Maximum columns are 6 in permutation key, so write in 6 columns)

1	2	3	4	5	6
C	o	m	e	h	o
m	e	t	o	m	o
r	r	o	w		

Now read according to the given permutation key, that is read first 4th column, then read 6th column, then read 1st column and so on.

Cipher Text: EOWOOCMROERHMMTO

Cryptanalysis of Simple Columnar Transposition Technique : It is quite simple to break into. It is a matter of trying out a few permutations and combinations of columns orders to get hold of original plain text.

3. Simple Columnar Transposition Technique with Multiple Rounds

To improve the basic **Simple Columnar Transposition Technique,** we can introduce more complexity. The idea is to use the same basic procedure as used by the Simple Transposition Technique, but do it more than once. That adds considerably more complexity for the cryptanalyst.

The algorithm is as follows:

1. Write the plain text message row by row in a rectangle of pre-defined size.
2. Read the message column by column. It can be read by given permutation key.
3. The message thus obtained is the cipher text message of round 1.
4. Repeat steps 1 to 3 as many times as desired.

Example :

Plain Text: Come home tomorrow

In cipher, this can be written as follows:

Permutation Key:

4	6	1	2	5	3

Solution. We write the message in the rectangle row by row (Maximum columns are 6 in permutation key, so write in 6 columns)

1	2	3	4	5	6
C	o	m	e	h	o
m	e	t	o	m	o
r	r	o	w		

Now read according to the given permutation key, that is read first 4th column, then read 6th column, then read 1st column and so on.

Cipher Text : EOWOOCMROERHMMTO

For multiple rounds we transform the ciphertext generated by first round again writing the text rectangle row by row.

1	2	3	4	5	6
e	o	w	o	o	c
m	r	o	e	r	h
m	m	t	o		

Now read according to the given permutation key, that is read first 4th column, then read 6th column, then read 1st column and so on.

Cipher Text: OEOCHEMMORMORWOT

4. Rotor Machine

Rotor machine is an electro-mechanical device used for encrypting and decrypting secret messages. The primary component is a set of *rotors* — 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,

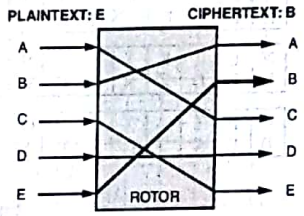


Fig. 2.2. *Idea of Rotor Machine*

changing the substitution. By this means, a rotor machine produces a complex polyalphabetic substitution cipher.

The rotor shown in Figure 2.2 shows 6 letters but the actual rotor using 26 letters. A three letter word ACE is enciphered as CEB if the rotor is stationary. Another figure 2.3 shows a three rotor machine for an eight letter alphabet before and after the first rotor has moved once. A three letter word BAG is enciphered as HFA before rotor has moved and after rotation it will result in CGE.

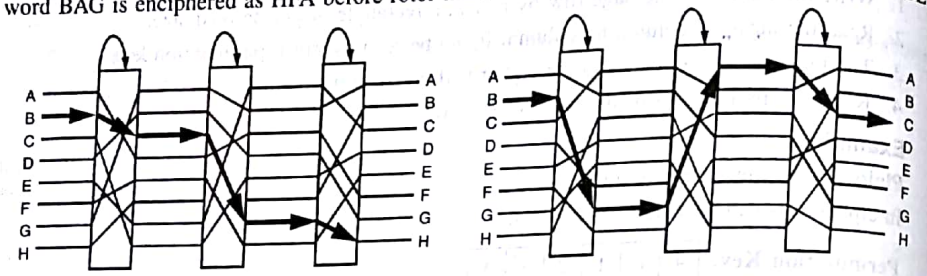


Fig. 2.3. *Three Rotor Machine (Before and After Rotation)*

Enigma Machine

In order to be as easy to decipher as encipher, some rotor machines, most notably the Enigma machine, were designed to be *symmetrical, i.e.*, encrypting twice with the same settings recovers the original message.

The enigma had three or four rotors and a reflector, which passed the signal back through the rotors. The enigma was famously broken by Alan Turing and the others at Bletchley Park.

The full Enigma consists of an input/output strip, three rotors and a reflector. (The mini-Enigma uses two rotors and no reflector while the micro-Enigma uses only one rotor and no reflector, producing what is essentially a Vinegere machine.) The Enigma is operated by locating the clear text letter on the input/output strip, following a circuitous course down through the rotors, bouncing off the reflector, and following another circuitous path back to the input/output strip where the ciphertext letter is read off. Here's what the process looks like :

The Fig. 2.4 shows the four alphabet strips of the Enigma rotors as they would appear if unwrapped from the cylindrical rotors. The top row is the input/output alphabet. The middle three

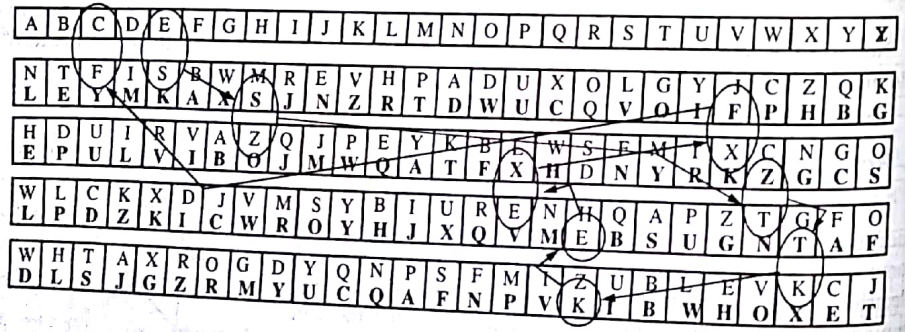


Fig 2.4. *Operation of Full Engima Machine*

rows, each consisting of two mixed alphabets, are the rotors. The bottom row, also holding two mixed alphabets, is the reflector. In this example, the encoding of the cleartext letter 'E' is explained. You will just repeat the same steps over and over for each rotor from top to bottom and then back to the top.

Follow the red lines down to the reflector, and the blues lines back up to the output. The complete process is given below-

1. Find the cleartext letter **'E'** on the input/output band.
2. Read down from the **'E'** to the top (black) alphabet of the first rotor, to **'S'**.
3. On the red band of the first rotor, find the matching **'S'**.
4. Read down from the **'S'** to the second rotor, to the black **'Z'**.
5. On the red band of the second row find the matching **'Z'**.
6. Read down from the **'Z'** to the **'T'** on the third rotor.
7. Find the corresponding **'T'** in the red alphabet of the third rotor.
8. Read down from the **'T'** to **'K'** on the reflector band.
9. Find the corresponding **'K'** in the red alphabet on the reflector.
10. Skip over the black alphabet on the reflector to the red **'E'** on the third rotor.
11. Find the corresponding **'E'** in the black alphabet of this rotor.
12. Read upwards from this **'E'** to the red **'X'** on the rotor above.
13. Find the corresponding black **'X'** on this rotor.
14. Read upwards to the red alphabet on the first rotor to the **'F'**.
15. Find the **'F'** in the black alphabet of the first rotor.
16. Read upwards from the **'F'** to the ciphertext letter **'C'** on the input/output band.
17. Write down this ciphertext letter and move on to the next cleartext letter.

Before we encode the next cleartext letter, however, we work the magic that made the Enigma famously difficult to crack. We turn the top rotor by one letter position to the right. The rotors look like Fig. 2.5 after this simple move. Notice how the cleartext letter 'E' now follows an entirely different path through the machine to be encoded as ciphertext 'O'.

After each letter is encoded the top rotor is moved one letter to the right. However, when this rotor returns to its starting position the second rotor down is moved one letter to the right. And, like the digits on an automobile odometer, when the second rotor finally returns to its starting position, the third rotor is turned by one letter position. In this way 26 × 26 × 26 = 17,576 different substitution alphabets are used over time.

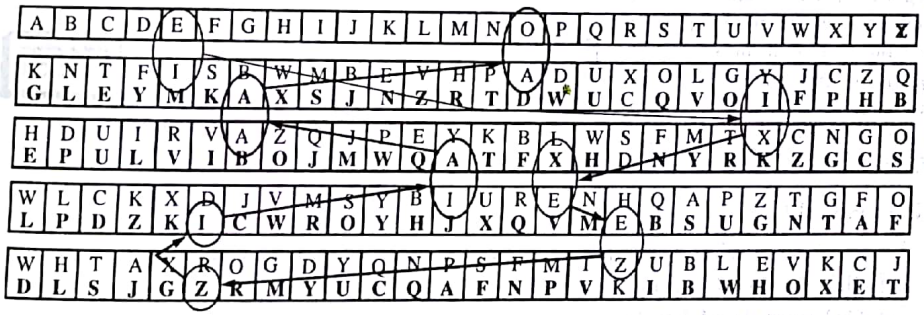


Fig. 2.5. *Same cleartext letter encoded after one move of the top rotor.*

Cryptanalysis of Enigma

The story of the cryptanalysis of the Enigma is perhaps the only story of military cryptanalysis ever recounted where a detailed view was given of that cryptanalysis on an ongoing basis, over multiple revisions and modifications of the cipher system under attack. The Enigma machines were a family of portable cipher machines with rotor scramblers. Good operating procedures, properly enforced, would have made the cipher unbreakable. We know that the enigma machine was broken during the war although the German army and the rest of the world did not hear about this until a few decades later.

Cryptanalysis of Transposition Cipher

Transposition ciphers are vulnerable to several kinds of cipher text-only attacks.

(a) Statistical Attack

A transposition cipher does not change the frequency of letters in the cipher text. It only records the letter. So the first attack that can be applied is single letter frequency analysis. This method can be useful if the length of the cipher text is long enough. We have seen this attack before. However, transposition cipher do not preserve the frequency of digram and trigrams.

(b) Genetic Algorithms

Brute Force attack has the disadvantage of high computational complexity. In order to overcome this complexity, the Meta heuristic search techniques like Genetic Algorithm are used.

A genetic algorithm is general method of solving problems to which no satisfactory, obvious, solution exists. It is based on the idea of emulating the evolution of a species in nature so the various components of the algorithm are roughly analogous to aspects of natural evolution. Common mathematical tasks amenable to genetic solutions include computing a curve to fit a set of data or approximating NP problems. Often these operators consist of flipping a single random bit of one individual or swapping two randomly selected substrings from a pair of parents to generate a new child. To simulate Darwinian survival of the fittest some representation of the fitness of the individuals must be generated.

2.3. CRYPTANALYSIS ATTACKS OR CRYPTANALYTIC ATTACKS

Cryptanalysis attacks are generally classified into two categories that distinguish the kind of information the cryptanalyst has available to mount an attack. The objective of cryptanalyst in all cases is to be able to decrypt new pieces of ciphertext without additional information. The idea for a cryptanalyst is to extract the secret key.

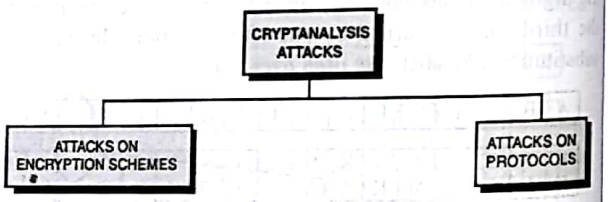


Fig 2.6. *Types of Cryptanalysis Attacks*

2.3.1. Attacks on Encryption Schemes

Cryptanalysis attacks on encryption schemes are classified into two categories.

(*a*) Plaintext-Based Attacks

(*b*) Ciphertext-Based Attacks

These attacks can be further divided as shown in Fig. 2.7.

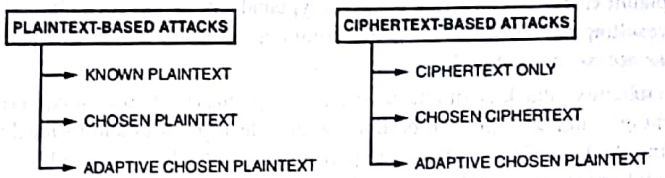


Fig. 2.7. *Types of Attacks on Encryption Schemes*

Known Plaintext and Ciphertext-Only Attacks

A known plaintext attack is an attack where a cryptanalyst has access to a plaintext and the corresponding ciphertext and seeks to discover a correlation between the two.

A ciphertext-only attack is an attack where a cryptanalyst has access to a ciphertext but does not have access to corresponding plaintext. With simple ciphers, such as the Caesar Cipher, frequency analysis can be used to break the cipher.

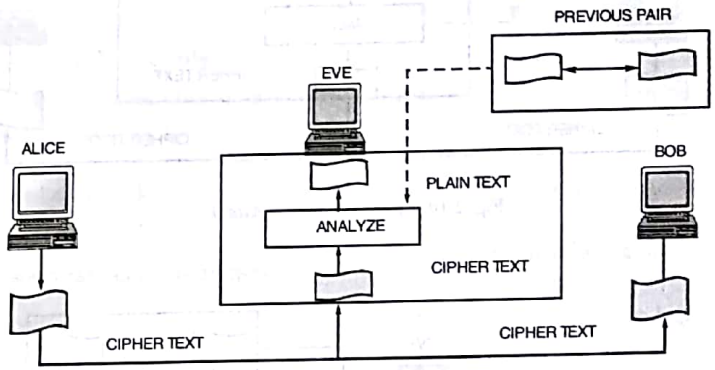


Fig. 2.8. *Known Plaintext Attack*

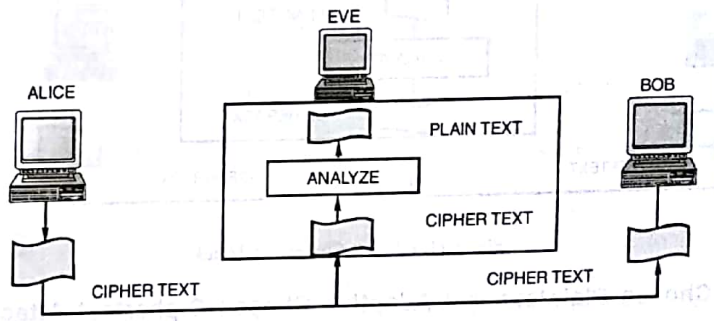


Fig. 2.9. *Ciphertext Only Attack*

Chosen Plaintext and Chosen Ciphertext Attacks

A chosen plaintext attack is an attack where a cryptanalyst can encrypt a plaintext of his choosing and study the resulting ciphertext. This is most common against asymmetric cryptography, where a cryptanalyst has access to a public key.

A chosen ciphertext attack is an attack where a cryptanalyst chooses a ciphertext and attempts to find a matching plaintext. This can be done with a decryption oracle (a machine that decrypts without exposing the key). This is also often performed on attacks versus public key encryption; it begins with a ciphertext and searches for matching publicly-posted plaintext data.

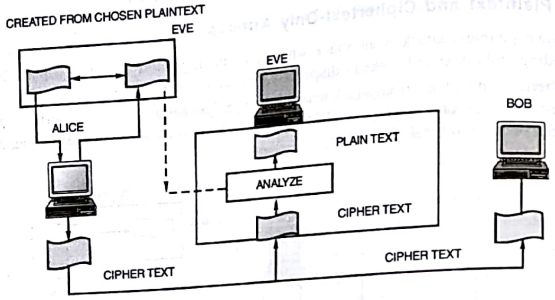


Fig. 2.10. *Chosen Plaintext Attack*

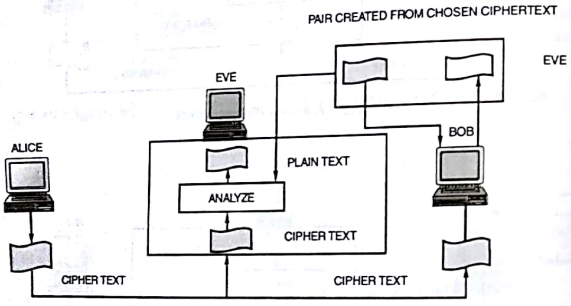


Fig. 2.11. *Chosen Ciphertext Attack*

Adaptive Chosen Plaintext and Adaptive Chosen Ciphertext Attacks

In both adaptive attacks, a cryptanalyst chooses further plaintexts or ciphertexts (adapts the attack) based on prior results.

Comparison between Different Cryptanalytic Attacks

Known Ciphertext

- Only have access to some enciphered messages
- Use statistical attacks only

Known plaintext

- Know (or strongly suspect) some plaintext-ciphertext pairs
- Use this knowledge in attacking cipher

Chosen plaintext

- Can select plaintext and obtain corresponding ciphertext
- Use knowledge of algorithm structure in attack

Chosen plaintext-ciphertext

- Can select plaintext and obtain corresponding ciphertext, or select ciphertext and obtain plaintext.
- Allows further knowledge of algorithm structure to be used.

2.3.2. Attacks on Protocols

The following is a partial list of attacks which might be mounted on various protocols. Until a protocol is proven to provide the service intended, the list of possible attacks can never be said to be complete.

1. **Known-key attack :** In this attack an adversary obtains some keys used previously and then uses this information to determine new keys.
2. **Replay :** In this attack an adversary records a communication session and replays the entire session, or a portion thereof, at some later point in time.
3. **Impersonation :** Here an adversary assumes the identity of one of the legitimate parties in a network.
4. **Dictionary :** This is usually an attack against passwords. Typically, a password is stored in a computer file as the image of an unkeyed hash function. When a user logs on and enters a password, it is hashed and the image is compared to the stored value. An adversary can take a list of probable passwords, hash all entries in this list, and then compare this to the list of true encrypted passwords with the hope of finding matches.
5. **Forward search :** This attack is similar in spirit to the dictionary attack and is used to decrypt messages.
6. **Interleaving attack :** This type of attack usually involves some form of impersonation in an authentication protocol.

NOTE

Decryption of Ciphertext : We have discussed encryption process only. Suppose ciphertext is given to you to find corresponding plaintext. To decrypt the ciphertext apply the encryption process in reverse order, e.g. if you add 3 to plaintext character in encryption process then in decryption process subtract 3 from ciphertext character.

CHAPTER 3

Modern Block Ciphers

3.1. BLOCK AND STREAM CIPHERS

Inside This Chapter

Block and Stream Ciphers are two categories of ciphers used in classical cryptography. Block and Stream Ciphers differ in how large a piece of the message is processed in each encryption operation. Block ciphers encrypt plaintext in chunks. Stream ciphers encrypt plaintext one byte or one bit at a time. A stream cipher can be thought of as a block cipher with a really small block size.

Block Ciphers

Classical block ciphers process messages in blocks, each of which is then encrypted/decrypted. A block cipher is called an *iterated cipher* if the ciphertext is computed by iteratively applying a round function several times to the plaintext. In each round a round key is combined with the text input. A symmetric key modern block cipher encrypts an n-bit block of plaintext or decrypts an n-bit block of ciphertext.

Block ciphers are, generally, more secure than stream ciphers and faster than public key ciphers, so they are used for encrypting large quantities of data, such as files and web pages. Also block ciphers can support different modes of operation, such as

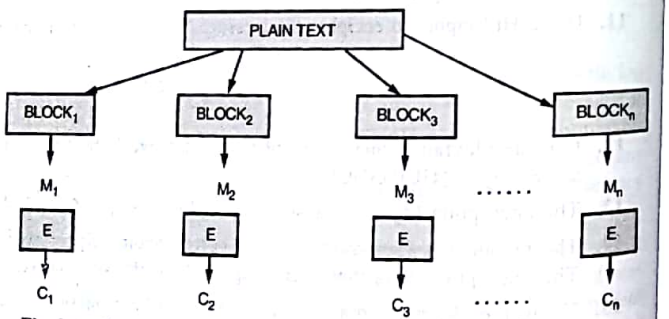


Fig. 3.1. *Block cipher ('M'= plaintext 'C' = ciphertext 'E' = encryption)*

cipher block chaining, which can enhance strength of the encryption even further. E.g. DES, AES, IDEA RC4, RC5 etc.

Stream Ciphers

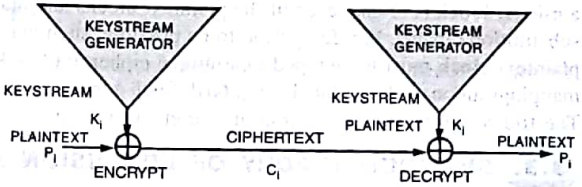


Fig. 3.2. *Stream Cipher*

Stream Ciphers operate on streams of plaintext and ciphertext one bit or byte at a time. Using a stream cipher, the same plaintext bit or byte will encrypt to a different bit or byte every time it is encrypted. A stream cipher uses keystream generators in conjunction with the plain text. The keystream generator produces a "Random" string of bits and these are used to generate a bit with each bit of the message, for example each bit of the message could be XORed with a corresponding bit from keystream generator.

Stream ciphers are extremely fast, much faster than block ciphers, but are less secure. For this reason, they tend to be used where speed is a critical factor, such as mobile phone telephony. The clever bit about stream ciphers is in the design of the algorithm that produces the key stream, as it must be able to produce a stream of bits which looks random, but is seeded from a unique key.

Table 3.1. Comparison of Block Cipher and Stream Cipher

S. No.	Block Cipher	Stream Cipher
1	Block cipher operated on large size data blocks.	Stream Ciphers usually operated on small data blocks (usually bit size).
2	Block Ciphers are slower.	Stream Ciphers are faster in comparison of block cipher.
3	Block ciphers processes the input one block of element at a time, producing an output block for each input block.	Stream ciphers process the input elements continuously producing output one element at a time.
4	In block cipher, the same key is used for encrypting every block.	In stream cipher, a different key is generated for each block.
5	Block ciphers are more efficient for computers.	Stream ciphers are easier for humans to do by hand.
6	DES is a block cipher with a 64 bit block size. AES is a block cipher with a 128 bit block size. RSA and Diffie-Hellman are block ciphers with variable block sizes.	RC4 cipher and one-time pad are examples of stream ciphers.

3.2. DESIGN PRINCIPLES OF BLOCK CIPHER

A block cipher works on units of a fixed size (known as a *block size*), but messages come in a variety of lengths which require that the final block be padded before encryption. Several padding schemes exist. The simplest is to add null bytes to the plaintext to bring its length up to a multiple of the block size, but care must be taken that the original length of the plaintext can be recovered

A block cipher transforms a plaintext block of n letters into an encrypted block. For the alphabet with 26 letters, there are 26^n possible different plaintext blocks. The most general way of encrypting a n-letter block is to take each of the plaintext blocks and map it to a cipher block (arbitrary n-letter substitution cipher). For decryption to be possible, such mapping needs to be one-to-one (*i.e.*, each plaintext block must be mapped to a unique ciphertext block). The number of different one-to-one mappings among n-letter blocks is $(26n)!$. Such a transformation is called reversible, or nonsingular. The following examples illustrate nonsingular and singular transformation for $n = 2$.

3.3. SHANNON THEORY OF CONFUSION AND DIFFUSION

The only generally accepted design principles for practical ciphers are the concepts of confusion and diffusion suggested by Shannon. Shannon's concepts of confusion and diffusion are as follows:

Confusion : In Shannon's original definitions, confusion makes the relation between the key and the ciphertext as complex as possible. Ideally, every letter in the key influences every letter of the ciphertext block. Replacing every letter with the one next to it on the typewriter keyboard is a simple example of confusion by substitution. However, good confusion can only be achieved when each character of the ciphertext depends on several parts of the key, and this dependence appears to be random to the observer. Ciphers that do not offer much confusion (such as Vigenere cipher) are vulnerable to frequency analysis.

"Confusion hides the relationship between the ciphertext and the key".

Diffusion : Diffusion refers to the property that the statistics structure of the plaintext is dissipated into long range statistics of the ciphertext. In contrast to confusion, diffusion spreads the influence of a single plaintext letter over many ciphertext letters. In terms of the frequency statistics of letters, digrams, etc. in the plaintext, diffusion randomly spreads them across several characters in the ciphertext. This means that much more ciphertexts are needed to do a meaningful statistical attack on the cipher.

"Diffusion hides the relationship between the ciphertext and the plaintext".

3.4. KERCKHOFF'S PRINCIPLE

In cryptography, **Kerckhoffs' principle** (also called **Kerckhoffs' assumption, axiom** or **law**) was stated by Auguste Kerckhoffs in the 19th century. A cryptosystem should be secure even if everything about the system, except the key, is in public knowledge.

In 1883 Auguste Kerckhoffs wrote two journal articles on *La Cryptographie Militaire*, in which he stated six design principles for military ciphers. Translated from French, they are :

1. The system must be practically, if not mathematically, indecipherable;
2. It must not be required to be secret, and it must be able to fall into the hands of the enemy without inconvenience;
3. Its key must be communicable and retainable without the help of written notes, and changeable or modifiable at the will of the correspondents;
4. It must be applicable to telegraphic correspondence;
5. It must be portable, and its usage and function must not require the concourse of several people;
6. Finally, it is necessary, given the circumstances that command its application, that the system be easy to use, requiring neither mental strain nor the knowledge of a long series of rules to observe.

Some are no longer relevant given the ability of computers to perform complex encryption, but his second axiom, now known as Kerckhoffs' Principle, is still critically important.

3.5. MODERN BLOCK CIPHER

The classical symmetric key ciphers are character oriented, but in computer we need bit oriented ciphers.

A symmetric-key modern block cipher encrypts an n-bit block of plaintext or decrypts an n-bit block of ciphertext. There are some key points.

(1) Algorithms use a k-bit key.
(2) The decryption algorithm must be the inverse of the encryption algorithm.
(3) Both algorithms uses same secret key.
(4) If the message has more than n bits, it should be divided into n-bit blocks.
(5) If the message is less then n bits, padding must be added to make it n bits.

Figure 3.3 shows the idea.

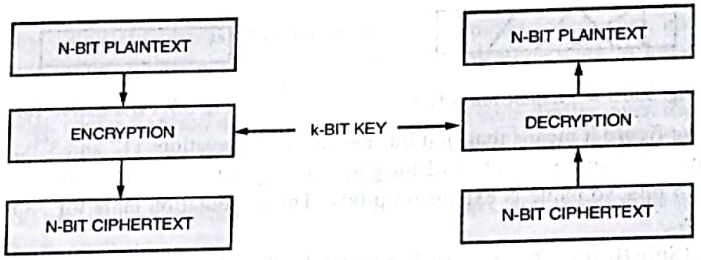


Fig. 3.3. *Modern Block Cipher*

3.6. COMPONENTS OF MODERN BLOCK CIPHER

Modern block ciphers normally are keyed substitution ciphers in which the key allows only partial mappings from the possible inputs to the possible outputs. To provide the required properties of a modern block cipher which is made up of combination of transposition units (called P-boxes), substitution units, exclusive-or, Circular shift, swap etc. Following are the components of modern block cipher :

1. P-boxes : A permutation box (or P-box) is a method of bit-shuffling used to permute or transpose bits across S-boxes inputs, retaining diffusion while transposing. There are three types of boxes.

(*a*) Straight p-boxes : A straight p-box is a p-box with m inputs and m outputs, *i.e.* if there are m bits in input then there will be m bits in output. There is $m!$ possible mappings.

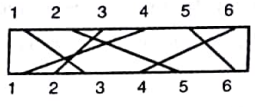


Fig. 3.4. *6 × 6 Straight P-box*

4	3	1	6	2	5

Fig. 3.5. *Permutation Table*

From above figure it means that first bit goes to 3rd position, second bit goes to 5th position, and third bit goes to 2nd position and so on in the output. The permutation table for 6 × 6 Straight P-box is shown above.

(b) Compression P-boxes : A compression p-box is a p-box with '*m*' input and '*n*' outputs, where $n < m$. Some of the inputs are blocked and do not reach the output. Compression p-boxes are used when we need to decrease the number of bits for the next stage.

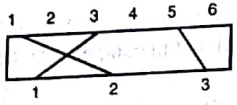


Fig. 3.6. *6 × 3 Compression P-box*

3	1	5

Fig. 3.7. *Permutation Table*

From above figure it means that first bit goes to 2nd position, third bit goes to 1st position and fifth bit goes to 3rd position. 2nd, 4th and 6th number bits are blocked. That means we compress the input from 6 bits to 3 bits, so name is compression p-box. The permutation table for 6×3 Compression P-box is shown above.

(c) Expansion P-boxes : An expansion p-box is a p-box with 'm' input and 'n' outputs, where $n > m$. Some of the inputs are repeated. Expansion p-boxes are used when we need to increase the number of bits for the next stage.

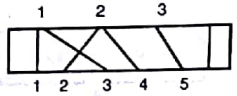


Fig. 3.8. *3 × 5 Expansion p-box*

1	2	1	2	3

Fig. 3.9. *Permutation Table*

From above figure it means that first bit repeats at two positions (1st and 3rd), second bit repeats at two positions (2nd and 4th) and third bit goes to 5th position. That means we expand the input from 3 bits to 5 bits, so name is expansion p-box. The permutation table for 3×5 Expansion p-box is given above.

2. S-box (Substitution box) : An S-box can be thought as an $m \times n$ substitution unit, where *m* and *n* are not necessarily the same. Below S-box of size 2 × 1 is shown and working of S-box is explained.

Suppose input is given 01 then 0 is left most bits and 1 is right most bit given so see in table according to this. So answer will be 1. S-box of size 2 × 1 means input is of 2 bit and output will be of 1 bit.

LEFT MOST BIT — RIGHT MOST BIT

	0	1
0	0	1
1	1	0

Fig. 3.10. *2 × 1 S-box Table*

3. Invertibility : A straight p-box is invertible, but compression and expansion p-box are not invertible. This means that we can use a straight p-box in encryption and its inverse in decryption cipher. In compression p-box an input can be dropped during encryption; the decryption does not have any clue how to replace the dropped bit. In expansion p-box an input can be repeated during encryption; the decryption does not have a clue which of several inputs are repeated.

4. Exclusive-OR : An important component in most block ciphers is exclusive-or operation due to following property.

1. Closure
2. Associativity
3. Commutativity
4. Existence of identity
5. Existence of inverse

5. Circular Shift : Shifting can be to the right or to the left by k bits. For example circular left shift by 3 bit or circular right shift by 3 bit.

6. Swap : Swap operation means shift n/2 left bits to right side and shift n/2 right bits to left side.

7. Split and Combine : Split operation splits n-bit word in the middle, creating two equal length words. The combine concatenates two equal length words to create n-bit word.

8. Complement : In complement operation 0 bit is changed to 1 and 1 bit is changed to 0.

3.7. PRODUCT CIPHERS

Shannon introduced the concept of product cipher. A product cipher is combination of substitution, permutation, and other components discussed earlier. Product ciphers can be classified into two categories.

(*a*) Feistel ciphers (*b*) Non-Feistel ciphers

3.7.1. Feistel Cipher

In cryptography, a Feistel cipher is a block cipher with a symmetric structure, named after 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.

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.

Shannon was the first to investigate the product cryptosystem (so called substitution-permutation network) and show that some sophisticated heuristic ciphers were nothing other than products of some simpler ciphers. Most importantly, Shannon identified that the necessary condition of the cipher strength increases as a result of cascading simple ciphers. One possible way to build a secret key algorithm using substitution-permutation-network is to break the input into manageable-sized chunks, do a substitution on each small chunk, and then take the outputs of all the substitutions and run them through a permuter that is as big as the input, which shuffles the letters around. Then the process is repeated, so that each letter winds up as an input to each of the substitutions.

Since modern cryptosystems are all computer-based, from now on we will assume that both plain and cipher text are strings of bits ({0, 1}), instead of strings of letters ({a, b, c ... z}).

The Feistel network shown in Figure 3.11 is a particular form of the substitution-permutation network. The input to a Feistel network is a plaintext block of n bits, and a key K. The plaintext block is divided into two halves, L0 and R0. The two halves of the data pass through r rounds of processing and then combine to produce the ciphertext block. Each round i have as input Li-1 and Ri-1, derived from the previous round, as well as a subkey Ki, derived from the overall key K. In general, the subkey Ki are different from K and from each other. In this structure, a substitution is performed via the round function F, and permutation is performed that interchanges the two halves of the data.

The exact realization of a Feistel network depends on the choices of the following parameters and design features.

- **Block size:** Larger block size means greater security, but reduces encryption/decryption speed.
- **Key size:** Larger key size means greater security but may decrease encryption/decryption speed.
- **Number of rounds:** Multiple rounds offer increasing security.
- **Subkey generation algorithm:** Greater complexity in subkey generation leads to greater security.
- **Round function:** Greater complexity in round function means greater difficulty of cryptanalysis.

There are two other considerations in the design of a Feistel cipher:

1. Fast software encryption/decryption : Sometimes encryption is embedded in applications in such a way as to preclude a hardware implementation. Accordingly, the speed of execution of the algorithm becomes a concern.

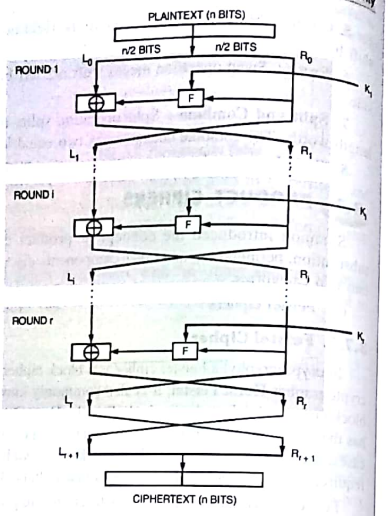


Fig. 3.11. *Feistel Cipher*

2. Ease of analysis : Although we would like to make our algorithm as difficult as possible to cryptanalyze, there is great benefit in making the algorithm easy to analyze. That is, if the algorithm can be concisely and clearly explained, it is easier to analyze that algorithm for cryptanalytic vulnerabilities and therefore develop a higher level of assurance as to its strength. DES, for example, does not have an easily analyzed functionality

Feistel Encryption/Decryption Process

It is worth noting that the process of decryption with a Feistel network is essentially the same as the encryption process by using the ciphertext as input to the network, but using the subkey K_i in reverse order, as shown in Figure 3.12. The reason is explained as follows. Let's consider the last step in encryption, which gives,

On the encryption side,

$$LE_{16} = RE_{15} \quad ...(1)$$

$$RE_{16} = LE_{15} \oplus F(RE_{15}, K_{16}) \quad ...(2)$$

On the decryption side

$$LD_1 = RD_0 = LE_{16} = RE_{15}$$

$$RD_1 = LD_0 \oplus F(RD_0, K_{16}) \quad ...(3)$$

$$= RE_{16} \oplus F(RE_{15}, K_{16}) \quad ...(4)$$

$$= [LE_{15} \oplus F(RE_{15}, K_{16})] \oplus F(RE_{15}, K_{16}) \quad ...(5)$$

$$= LE_{15} \quad ...(6)$$

$$...(7)$$

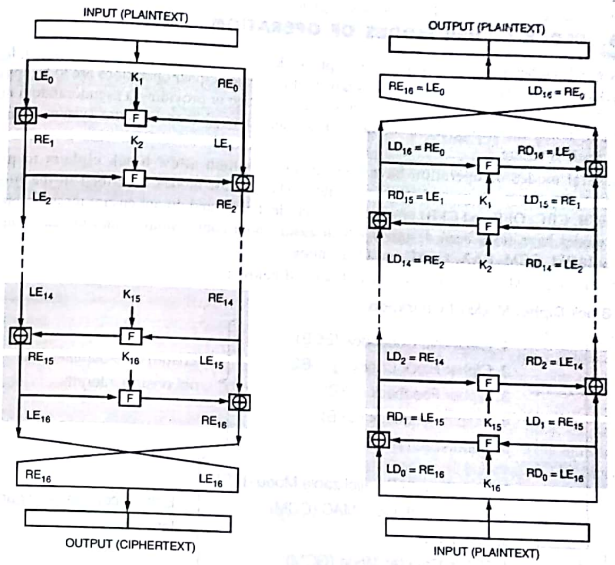


Fig. 3.12. *Feistel Encryption/Decryption Process*

The process can be done iteratively. Finally, we will see that the output of the decryption is the same as the input to the encryption (*i.e.*, original plaintext).

3.7.2. Non Feistel Cipher

A Non Feistel cipher uses only invertible components. A component in the plaintext has the corresponding component in the cipher. For example, S-boxes need to have an equal number of inputs and outputs to be compatible. No compression or expansion p-boxes are allowed, because they are not invertible. In a Non Feistel cipher, there is no need to divide the plaintext into two halves as we saw in the Feistel ciphers

Unbalanced Feistel Ciphers

It uses 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.

The Feistel construction is also used in cryptographic algorithms other than block ciphers. For example, the Optimal Asymmetric Encryption Padding (OAEP) scheme uses a simple Feistel network to randomize ciphertext in certain asymmetric key encryption schemes.

3.8. BLOCK CIPHER MODES OF OPERATION

A block cipher provides a way to encrypt blocks of plaintext to yield blocks of ciphertext. A block cipher mode of operation specifies how multiple block cipher operations are to be combined to accomplish some larger task such as encrypting a message or providing a pseudorandom number generator. Using the wrong mode for the task at hand may give an insecure system even if the cipher itself is secure.

Several modes of operation have been invented which allow block ciphers to provide confidentiality for messages of arbitrary length. The earliest modes described in the literature (e.g., ECB, CBC, OFB and CFB) provide only confidentiality, and do not ensure message integrity. Other modes have since been designed which ensure both confidentiality and message integrity, such as IAPM, CCM, EAX, GCM, and OCB modes.

Block Cipher Modes of Operation are described below :

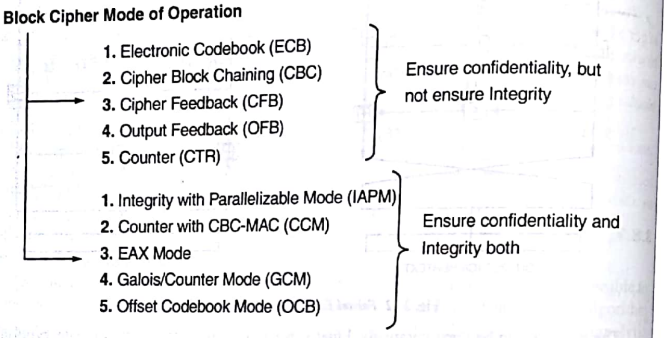


Fig. 3.13. *Block Cipher Mode of Operation*

3.8.1. Electronic Code Book (ECB) Mode

The simplest of the encryption modes is the electronic codebook (ECB) mode. The message is divided into blocks and each block is encrypted separately using the same key. The block size is n bits.

If a message is not a multiple of *n* bits then extra bits are padded to last block to make it multiple of *n* bits. e.g., suppose given message is of size 620 bits and block size is 64 bits. It is not a exact multiple of 64, so 20 bits are added to make it 640 bits (exact multiple of 64, 64*10=640 bits).

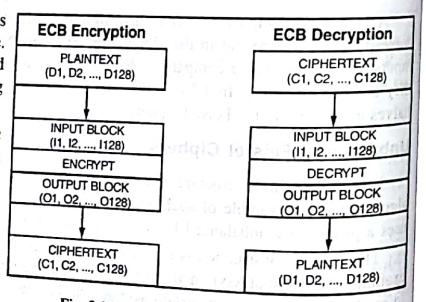


Fig. 3.14. *ECB Encryption/Decryption Process*

The disadvantage of this method is that identical plaintext blocks are encrypted into identical ciphertext blocks; thus, it does not hide data patterns well. If enough blocks are encrypted this way, the system becomes vulnerable to a code book attack. In some senses, it doesn't provide serious message confidentiality, and it is not recommended for use in cryptographic protocols at all.

The encryption and decryption process can be written as given below :

$$C_n = E(K, P_n)$$

ENCRYPTION EQUATION

$$P_n = D(K, C_n)$$

DECRYPTION EQUATION

Attack on ECB Mode

A code book attack is a technique for cryptanalysis. The name comes from the attack on a block cipher, the attacker tries to build up a "code book". Codebook is a table saying which cipher texts correspond to which plaintexts.

For example, consider a block cipher with only an 8 bit block size. Assume the enemy is able to get or guess some plaintext; he makes a little table, his own "code book" showing which ciphertext blocks corresponds to which plaintexts. If he can fill in all 256 entries, then the cipher is broken; he can read everything ever sent with that key. Such tiny block sizes are therefore never used; real block ciphers all have block sizes of at least 64 bits.

Even a partially filled in code book weakens the cipher. When the code book has around $2^{blocksize/2}$ entries, the chance that two entries will be the same, giving the enemy some information, becomes significant. For this reason, any block cipher must be re-keyed before $2^{blocksize/2}$ blocks as an absolute upper limit. The general practice is to re-key long before that.

3.8.2. Cipher Block Chaining (CBC) Mode

In the cipher-block chaining (CBC) mode, each block of plaintext (except first plaintext block) is XORed with the previous ciphertext block before being encrypted. First plaintext block is XOR with initialization vector (IV). This way, each ciphertext block is dependent on all plaintext blocks processed up to that point. Also, to make each message unique, an initialization vector must be used in the first block.

For n = 1, an **initialization vector** (IV) must be provided to act as C_0. This need not be secret, but it must be different for each message and should be random. If the same IV is repeatedly used, then if two or more messages start with the same text, they will encrypt identically for the first block or the first few blocks. This is an unnecessary weakness; using unique IVs is therefore standard practice.

The encryption and decryption process can be written as given below :

$$C_0 = IV$$
$$C_n = E(K, [P_n \text{ XOR } C_{n-1}])$$

ENCRYPTION EQUATION

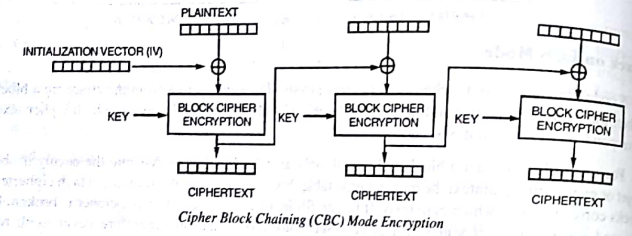
$$C_0 = IV$$
$$P_n = D(K, C_n) \text{ XOR } C_{n-1}$$

DECRYPTION EQUATION

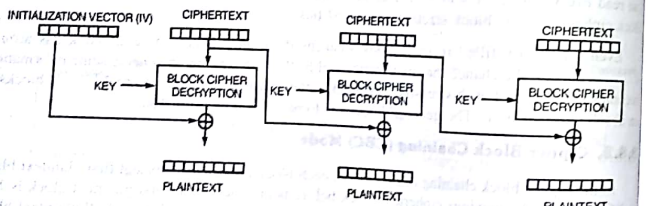
CBC mode makes the encryption of any block dependant on all blocks previously encrypted. A bit error in an encrypted block, such as might be caused by line noise, will cause the decryption of

that block and the next to be garbled, but later blocks will not be affected. CBC is self-recovering against bit-flipping errors. However, loss of synchronization is fatal; if even a single bit is dropped or added, then the affected block and all that follow it will be garbled. Authentication of the packet or message can prevent such problems if decryption is only applied to data that has passed authentication.

Cipher block chaining is much the most widely used mode. IP-Sec specifies it as the only permitted mode.



Cipher Block Chaining (CBC) Mode Encryption



Cipher Block Chaining (CBC) Mode Decryption

Fig. 3.15. CBC Encryption/Decryption Process

3.8.3. Cipher Feedback (CFB)

ECB and CBC encrypt and decrypt in blocks, but there may be situation in real life that message is not of block size. For example, DES use 64 bits block, AES use 128 bits block. So in such situation ECB and CBC mode will not be secure, so use CFB mode.

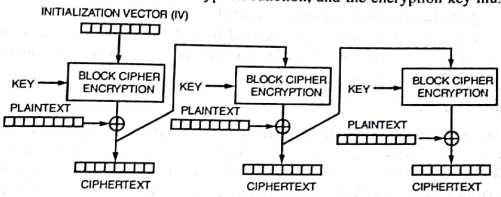
In encryption first IV (initialization vector) is encrypted then r bits (equal to plaintext block) of encrypted IV are XOR with plaintext block (r bit). Now this ciphertext works as IV for second plaintext block and this process will be continued.

In decryption first IV (initialization vector) is encrypted then r bits (equal to ciphertext block) of encrypted IV are XOR with ciphertext block (r bit). For second step, first ciphertext block will work as IV and this process will continue.

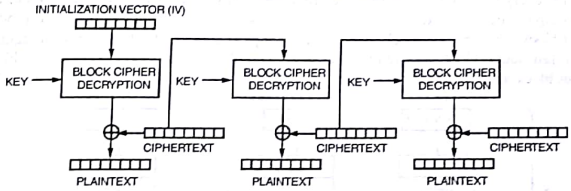
NOTE

In CFB r bits are chosen by r bit shift left operation.

For decrypting a block, the encryption function is used. This has two implications: It is not necessary to know the inverse of the encryption function, and the encryption key must be secret.



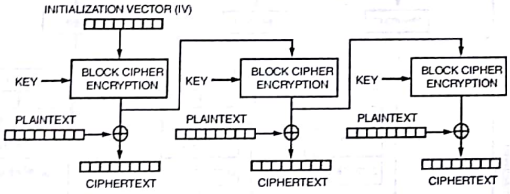
Cipher Feedback (CFB) Mode Encryption



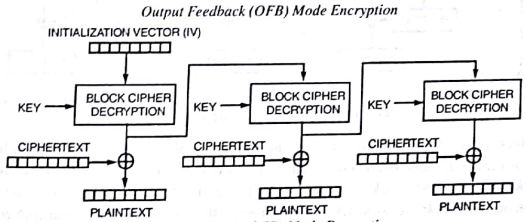
Cipher Feedback (CFB) Mode Decryption

Fig. 3.16. CFB Encryption/Decryption Process

3.8.4. Output Feedback (OFB)



Output Feedback (OFB) Mode Encryption



Output Feedback (OFB) Mode Decryption

Fig. 3.17. OFB Encryption/Decryption Process

It is very similar to CFB. In OFB mode, the output of the encryption function for one block is used as input for the next block. This effectively lets a block cipher be used as a stream cipher. A stream depending on the key and initialization vector IV.

To decrypt, the same output stream is generated from the initialization vector, then XOR-ed onto the ciphertext to recover the plaintext. As in CFB mode, the encryption function is used to decrypt, with the same implications.

3.8.5. Counter (CTR)

The Counter (CTR) mode is a confidentiality mode that features the application of the forward cipher to a set of input blocks, called counters, to produce a sequence of output blocks that are exclusive-ORed with the plaintext to produce the ciphertext, and vice versa. The sequence of counters must have the property that each block in the sequence is different from every other block.

In CTR encryption, the forward cipher function is invoked on each counter block, and the resulting output blocks are exclusive-ORed with the corresponding plaintext blocks to produce the ciphertext blocks. For the last block, which may be a partial block of u bits, the most significant u bits of the last output block are used for the exclusive-OR operation; the remaining b-u bits of the last output block are discarded.

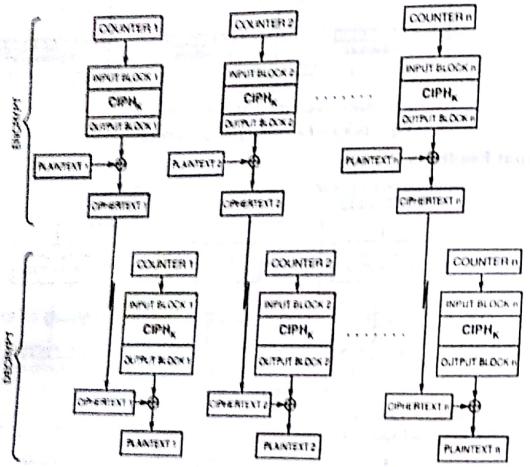


Fig. 3.18. CTR Encryption/Decryption

In CTR decryption, the forward cipher function is invoked on each counter block, and the resulting output blocks are exclusive-ORed with the corresponding ciphertext blocks to recover the plaintext blocks. For the last block, which may be a partial block of u bits, the most significant u bits of the last output block are used for the exclusive-OR operation; the remaining b-u bits of the last output block are discarded.

Advantages and Disadvantages of Block Cipher Modes of Operation

Name of Mode	Advantage	Disadvantage
ECB	ECB is simple and suitable for small messages with session keys.	Identical plaintext blocks of ECB always result in identical cipher text blocks due to which it is not suitable for large messages having repetitive plaintext blocks.
CBC	CBC also use the same key for encryption but results in different cipher text blocks for identical plain text blocks.	In CBC if any cipher text block gets corrupted, it will affect the decryption of two cipher text blocks.
CFB	CFB is suitable for block as well as stream cipher mode. It is used in real time applications as each character of plaintext can be encrypted and transmitted as when it is ready.	If any cipher text block gets corrupted in transit, it will affect the decryption of many subsequent cipher text blocks.
OFB	OFB is also suitable for block as well as stream cipher mode. If any cipher text block gets corrupted in transit, it will affect the decryption of only that block of cipher not many subsequent cipher text blocks. OFB mode has an advantage over CFB mode in that any bit errors that might occur during transmission are not propagated to affect the decryption of subsequent blocks.	
Counter	Knowing block number of a block in counter mode, any block can be encrypted/decrypted out of sequence by using Counter value.	

3.9. OTHER BLOCK CIPHER MODES OF OPERATION

1. Dual Use Modes

When a block cipher is used in a hybrid cryptosystem, a cryptographic hash is often used as well; the cipher provides secrecy and the hash provides authentication. However, this is somewhat inefficient; the system must make two passes through the data, one to encrypt it and one to hash it. We need algorithms that can do both in one pass.

2. Tweakable Modes

Where a normal block cipher has only two inputs, plaintext and key, a tweakable block cipher has a third input called the tweak. The tweak serves much the same purpose that an initialization vector does for CBC mode. The tweak, along with the key, controls the operation of the cipher. If changing tweaks is sufficiently lightweight, compared to the key scheduling operation which is often fairly expensive, then some new modes of operation become possible. For example, instead of exclusive-OR-ing the previous ciphertext into the plaintext, you can use that ciphertext to change the tweak.

3.10. LINEAR AND DIFFERENTIAL CRYPTANALYSIS

Attacks on classical cipher are not feasible on modern block cipher. For example, Brute force attack on modern cipher is not feasible because key size is large. Some new attacks on structure of modern block cipher are introduced.

Linear cryptanalysis and differential cryptanalysis are related attacks used primarily against iterative symmetric key block ciphers. An iterative cipher (also called a product cipher) conducts multiple rounds of encryption using a subkey for each round. Examples include the Feistel Network used in DES and the State rounds used in AES. In both attacks, a cryptanalyst studies changes to the intermediate ciphertext between rounds of encryption. The attacks can be combined, which is called differential linear cryptanalysis.

A goal of strong encryption is to produce ciphertexts that appear random where a small change in a plaintext results in a random change in the resulting ciphertext. This quality is called diffusion, and any changed ciphertext bit should have a 50% chance of being a 1 or a 0. Both attacks seek to discover non-randomness (cases where the 50% rule is broken) in an effort to discover potential subkeys.

Linear Cryptanalysis

It was introduced by Mitsuru Matsui in 1993. Linear cryptanalysis is a known plaintext attack that requires access to large amounts of plaintext and ciphertext pairs encrypted with an unknown key. It focuses on statistical analysis against one round of decryption on large amounts of ciphertext.

The cryptanalyst decrypts each ciphertext using all possible subkeys for one round of encryption and studies the resulting intermediate ciphertext to seek the least random result. A subkey that produces the least random intermediate cipher for all ciphertexts becomes a candidate key (the most likely subkey).

There are two parts to linear cryptanalysis. The first is to construct linear equations relating plaintext, ciphertext and key bits that have a high bias; that is, whose probabilities of holding (over the space of all possible values of their variables) are as close as possible to 0 or 1. The second is to use these linear equations in conjunction with known plaintext-ciphertext pairs to derive key bits.

Constructing Linear Equations

For the purposes of linear cryptanalysis, a linear equation expresses the equality of two expressions which consist of binary variables combined with the exclusive-or (XOR) operation. For example, the following equation, from a hypothetical cipher, states the XOR sum of the first and third plaintext bits (as in a block cipher's block) and the first ciphertext bit is equal to the second bit of the key:

$$P_1 \oplus P_3 \oplus C_1 = K_2$$

In an ideal cipher, any linear equation relating plaintext, ciphertext and key bits would hold with probability 1/2. Since the equations dealt with in linear cryptanalysis will vary in probability, they are more accurately referred to as linear *approximations*.

The procedure for constructing approximations is different for each cipher. In the most basic type of block cipher, a substitution-permutation network, analysis is concentrated primarily on the S-boxes, the only nonlinear part of the cipher (*i.e.*, the operation of an S-box cannot be encoded in a linear equation). For small enough S-boxes, it is possible to enumerate every possible linear equation relating the S-box's input and output bits, calculate their biases and choose the best ones. Linear approximations for S-boxes then must be combined with the cipher's other actions, such as permutation and key mixing, to arrive at linear approximations for the entire cipher.

Deriving Key Bits

Having obtained a linear approximation of the form:

$$P_{i_1} \oplus P_{i_2} \oplus \ldots \oplus C_{j_1} \oplus C_{j_2} \oplus \ldots = K_{k_1} \oplus K_{k_2} \oplus \ldots$$

We can then apply a straightforward algorithm using known plaintext-ciphertext pairs, to guess at the values of the key bits involved in the approximation.

For each set of values of the key bits on the right-hand side (referred to as a *partial key*), count how many times the approximation holds true over all the known plaintext-ciphertext pairs; call this count T. The partial key whose T has the greatest absolute difference from half the number of plaintext-ciphertext pairs is designated as the most likely set of values for those key bits. This is because it is assumed that the correct partial key will cause the approximation to hold with a high bias. The magnitude of the bias is significant here, as opposed to the magnitude of the probability itself.

This procedure can be repeated with other linear approximations, obtaining guesses at values of key bits, until the number of unknown key bits is low enough that they can be attacked with brute force.

Differential Cryptanalysis

Differential cryptanalysis was introduced by Eli Biham and Adi Shamir. Differential cryptanalysis is a chosen plaintext attack that seeks to discover a relationship between ciphertexts produced by two related plaintexts. It focuses on statistical analysis of two inputs and two outputs of a cryptographic algorithm.

A plaintext pair is created by applying a Boolean exclusive or (XOR) operation to a plaintext. For example, XOR the repeating binary string 10000000 to the plaintext. This creates a small difference (hence the term differential cryptanalysis) between the two. The cryptanalyst then encrypts the plaintext and its XORed pair using all possible subkeys, and it seeks signs of non-randomness in each intermediate ciphertext pair. The subkey that creates the least random pattern becomes the candidate key.

Differential cryptanalysis is a type of attack that can be mounted on iterative block ciphers. S. Murphy first introduced this technique in an attack on FEAL-4 (Fast Data Encipherment Algorithm, 4 for rounds) but this method was later improved and perfected by Biham and Shamir who used Differential Cryptanalysis to attack DES. Differential cryptanalysis looks specifically at ciphertext pairs: pairs of ciphertext whose plaintexts have particular differences, but are encrypted under the same key. By careful analysis of the available data, probabilities can be assigned to each of the possible keys and eventually the most probable key is identified as the correct one.

Differential cryptanalysis has been used against a great many ciphers with varying degrees of success. In attacks against DES, its effectiveness is limited by carefully engineered S-boxes designed for DES in the mid-1970s. Differential cryptanalysis has also been useful in attacking other cryptographic algorithms such as hash functions.

SUMMARY

- Block ciphers encrypt plaintext in chunks. Stream ciphers encrypt plaintext one byte or one bit at a time.
- A block cipher works on units of a fixed size (known as a *block size*), but messages come in a variety of lengths which require that the final block be padded before encryption.

CHAPTER 4

Symmetric Key Cryptography

4.1. INTRODUCTION TO SYMMETRIC KEY CIPHERS

Symmetric encryption is a form of cryptosystem in which encryption and decryption are performed using same key. The original message from A to B is called **plaintext;** the message that is sent through the channel is called the **ciphertext.** To create the ciphertext from the plaintext, A uses an **encryption algorithm** and a **shared secret key.** To recover the plaintext from ciphertext, B uses a decryption algorithm and the same secret key. The idea behind a symmetric key cipher is shown in the Fig. 4.1.

Inside This Chapter

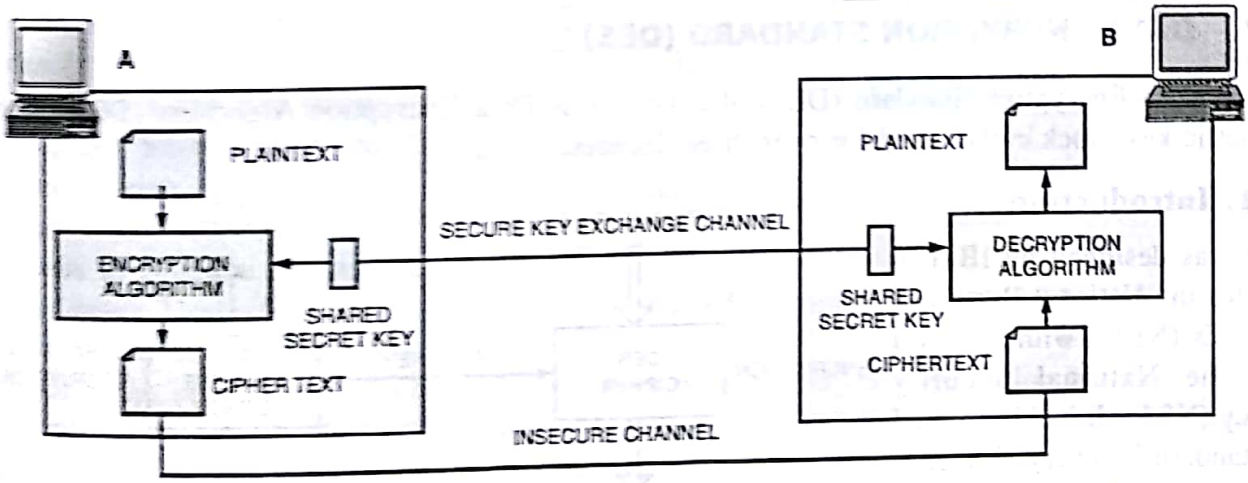


Fig. 4.1. *Idea of Symmetric Key Cryptography*

The encryption and decryption algorithms are referred as ciphers. There are some key points in symmetric key cryptography.

1. Symmetric key cryptography uses a single key for both encryption and decryption.
2. The key between A and B can be used in both directions.
3. The encryption and decryption algorithms are inverses of each other.

If P is the plaintext, C is the ciphertext, and K is the key, the encryption algorithm $E_k(P)$ creates the ciphertext from the plaintext; the decryption algorithm $D_k(C)$ creates the plaintext from the ciphertext. Which can be shown as

Encryption : $C = E_k(P)$	*Decryption* : $P = D_k(C)$

In symmetric key cryptography A and B need another channel, a secured one to exchange the secret key.

Keys required : The table shows the keys requirement in following situations.

1. When A wants to communicate only with B, we needed one key (A to B).
2. When A wants to communicate with B and C, we need two keys (A to B and A to C). If B also wants to communicate with C, we need another key (B to C). Thus we need three keys to serve the needs of three communicating pairs.
3. Let us consider the participation of four persons (A, B, C, D). Now we need six keys as A to B, A to C, A to D, B to C, B to D and C to D.

No. of Persons	Keys Required
2 (A, B)	1 (A to B)
3 (A, B, C)	3 (A to B, A to C, B to C)
4 (A, B, C, D)	6 (A to B, A to C, A to D, B to C, B to D, C to D)
5 (A, B, C, D, E)	10 (A to B, A to C, A to D, A to E, B to C, B to D, B to E, C to D, C to E, D to E)

In general if there are m persons in a group who need to communicate with each other, how many keys are needed ? The answer is $(m * (m - 1))/2$ because each person needs $(m - 1)$ keys to communicate with the rest of the group.

4.2. DATA ENCRYPTION STANDARD (DES)

The Data Encryption Standard (DES) also known as Data Encryption Algorithm (DEA) is a symmetric key block cipher used for over three decades.

4.2.1. Introduction

It was designed by IBM in 1976 for the National Bureau of Standards (NBS), with approval from the National Security Agency (NSA). It had been used as a standard for encryption until 2000. From 2001 the AES replaced DES.

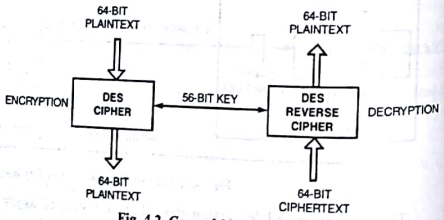


Fig. 4.2. *General Idea of DES*

The National Bureau of Standards initiated development of the DES when it published in the *Federal Register* of

May 15, 1973, a solicitation for encryption algorithms for computer data protection. Responses to this solicitation demonstrated that there was an interest in developing such a standard, but that little technology in encryption was publicly available. NBS requested assistance from the National Security Agency (NSA) in evaluating encryption algorithms, if any were received or in providing an encryption algorithm, if none were received.

IBM had initiated a research project in the late 1960s in computer cryptography. This development activity resulted in several publications, patents, cryptographic algorithms, and products. One of the algorithms was to become the Data Encryption Standard.

The DES algorithm is a block cipher that uses the same binary key both to encrypt and decrypt data blocks, and thus is called a symmetric key cipher. DES operates on 64-bit "plaintext" data blocks, processing them under the control of a 56-bit key to produce 64 bits of encrypted cipher text. Similarly, the DES decryption process operates on a 64-bit cipher text block using the same 56-bit key to produce the original 64-bit plaintext block.

4.2.2. DES Building Blocks

DES uses a sequence of operations, including several substitution and permutation primitives, to encrypt a data block. These primitives are subsequently used to reverse the encryption operation. Each set of primitive operations is called a "round," and the DES algorithm uses 16 rounds to ensure that the data are adequately scrambled to meet the security goals. The secret key is used to control the operation of the DES algorithm.

4.2.3. Encryption Process of DES

The Encryption process is made of two permutations (P-boxes), which we call initial and final permutation and 16 Feistel rounds. Each round uses a different 48 bit round key. There are total 16 round key needed (K_1, K_2, ... K_{16}).

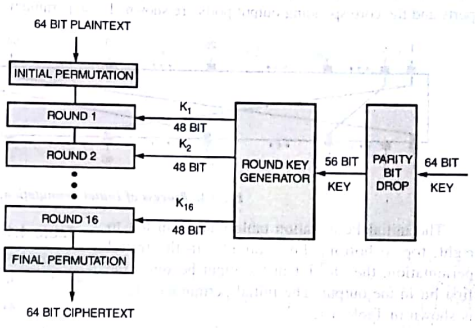


Fig. 4.3. *Encryption Process of DES*

Each of these permutations (P-boxes) takes 64-bit input and permutes them. 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 rounds of the same function, which involves both permutation and substitution function.

The output of the 16th round consists of 64-bits that is passed through a final permutation that is the inverse of the initial permutation, to produce the 64-bit cipher text.

The encryption process of DES can be summarized as given below :

1. Plaintext is broken into blocks of 64 bits. Encryption is done block wise.

2. A block is first gone through an Initial Permutation. Then produces two halves of the permuted block; say Left Plain Text (LPT) and Right Plain Text (RPT) where LPT is left 32 bits and RPT is right 32 bits.
3. Now, each of LPT and RPT goes through 16 rounds of encryption process, each with its own round key.
4. At the end, LPT and RPT are rejoined, and a Final Permutation is performed on the combined block of 64 bits.
5. The result of this process produces 64-bit cipher text.
6. Step 1-5 is performed for each block of plaintext, each of size 64-bit.
7. At last all ciphertext are combined and this will be ciphertext of complete message.

Broadly we can divide DES process into four parts.

(a) Initial Permutation
(b) Rounds
(c) Final Permutation
(d) Round Key Generation.

Now complete process is described in more detail and working of each step is described.

(a) Initial Permutation

The following figure shows the initial permutation (P-boxes). Initial permutation happens only once. This permutation takes a 64-bit input and permute according to predefined rules. Here, few input ports and the corresponding output ports are shown. These permutations are inverses of each other.

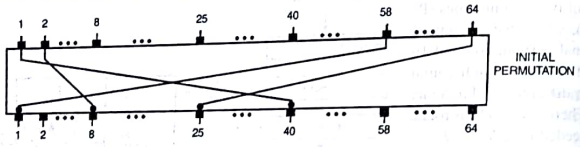


Fig 4.4. *Process of Initial Permutation*

The initial Permutation table read from left to right, top to bottom. For example, in the initial permutation, the 58th bit in the input becomes the first bit in the output. The Initial permutation table is shown in Table 4.1.

Table 4.1. Initial Permutation Table

INITIAL PERMUTATION							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	15	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

(b) Rounds

DES uses 16 rounds and each one is a Feistel round. The round takes L_{I-1} and R_{I-1} each of 32 bits and creates L_I and R_I as output. Now this L_I and R_I will work as input for next round. Each round has two cipher elements *i.e.*, mixer and swapper.

In mixer R_{I-1} passes through DES function. The output after DES function is XORed with L_{I-1}. Now swapper swaps the left half of the text with the right half.

The process of each round is shown in Fig. 4.5.

DES Function

The main component of DES cipher is its DES function. In Fig. 4.6 $f(R_{I-1}, K_I)$ is DES function for first round. For other rounds only round key will be changed. For example, for second round (K_2), for third round (K_3) and so on.

DES function applies a 48-bit round key to the rightmost 32 bits (R_{I-1}) to produce a 32-bit output.

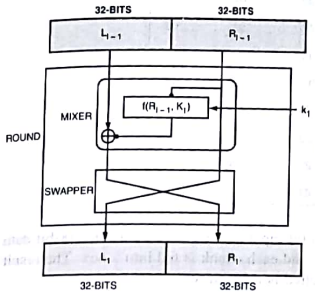


Fig. 4.5. *Process of each round in DES*

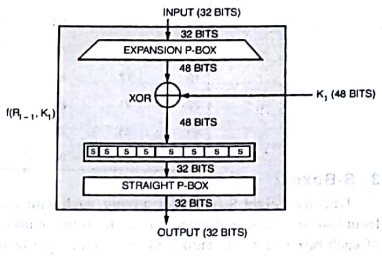


Fig. 4.6. *Process of DES function $f(R_{I-1}, K_I)$*

DES function is made up of four units :

1. An Expansion P-box
2. A Whitener (that adds key)
3. A Group of S-boxes (eight S-boxes)
4. A Straight P-box.

Now components of DES function $f(R_{I-1}, K_I)$ are described.

1. Expansion P-Box

During expansion permutation, the RPT (R_{I-1}) is expanded from 32 bits to 48 bits, the bits are permuted as well, hence the name expansion permutation (P-box). There is need of expansion P-box because RPT need to XOR with 48-bit key, so RPT should be of size 48-bit. To expand RPT (R_{I-1}), RPT is divided into eight 4-bit sections. Each 4-bit section is then expanded to 6-bits.

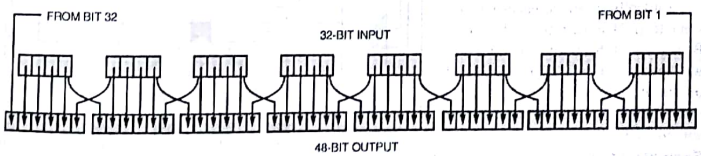


Fig. 4.7. *Process of Expansion P-box*

Although the relationship between the input and output can be defined mathematically, DES uses Table 4.2 to define this P-box.

2. Whitener (XOR)

After the expansion P-box, XOR operation is performed on expanded right sections (RPT) and round key. Both the right section and the key are 48-bits in length. Also the round key is used only in this operation.

Table 4.2. Expansion P-box Table

32	01	02	03	04
04	04	06	07	08
08	09	10	11	12
12	13	14	15	16
16	17	18	19	20
20	21	22	23	24
24	25	26	27	28
28	29	30	31	32

3. S-Boxes

DES uses eight S-boxes for mixing, each with a 6-bit input and a 4-bit output. The 48-bit data from last operation is divided into eight, 6-bit chunks, and each chunk is fed into a box. The result of each box is a 4-bit chunk and after combining resulted is a 32-bit text.

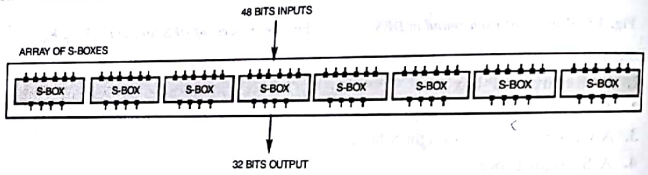


Fig. 4.8. Eight S-boxes

The substitution defined here is the combination of bits 1 and 6 of the input defines one of 4-rows and the combination of bits 2 through 5 defines one of the sixteen columns as shown in Fig. 4.9.

Typically, confusion is provided by some form of substitution ("S-boxes"). Because each S-box has its own table, we need eight tables to define the output of these boxes. Each table is of size 4*16. The eight—S box tables are shown in Table 4.3.

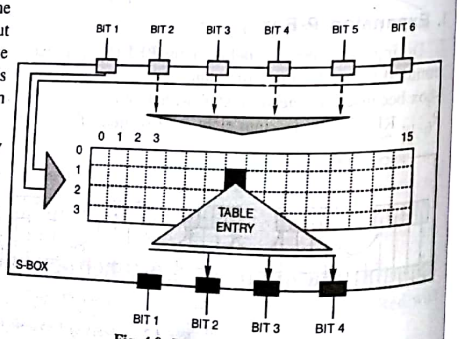


Fig. 4.9. Process of Eight S-boxes

Example 4.1. *Suppose input to s-box 1 is 100011. What will be the output ?*

Solution. Given 100011

First write the first and the last bits together, we get 11 in binary, which is 3 in decimal (row no in S-box table). The remaining bits are 0001 in binary, which is 1 in decimal (column no in S-box table).

To find the output see value in 3rd row and 1st column of S-box 1 table. The result is 12 in decimal, which is 1100 in binary. So the answer is 1100.

Note: In this manner we can find value through any S-box.

Table 4.3. Tables for Eight S-boxes

S-BOX 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	10	03	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

S-BOX 2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	01	08	14	06	11	03	04	09	07	02	13	12	00	05	10
1	03	13	04	07	15	02	08	14	12	00	01	10	05	09	11	05
2	00	14	07	11	10	04	13	01	05	08	12	06	09	03	02	15
3	13	08	10	01	03	15	04	02	11	06	07	12	00	05	14	09

S-BOX 3

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	00	09	14	06	03	16	05	01	13	12	07	11	04	02	08
1	13	07	00	09	03	04	06	10	02	08	05	14	12	11	15	01
2	13	06	04	09	08	15	03	00	11	11	02	12	05	10	14	07
3	01	10	13	00	06	09	08	07	04	15	14	03	11	05	02	12

S-BOX 4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	07	13	14	05	00	6	09	10	01	02	08	05	11	12	04	15
1	13	08	11	08	06	15	00	03	04	07	02	12	01	10	14	09
2	10	06	09	00	12	11	07	13	15	01	03	14	05	02	08	04
3	03	15	15	06	10	01	13	08	09	04	05	11	12	07	02	14

S-BOX 5

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	02	12	04	01	07	10	11	06	08	05	05	15	13	00	14	09
1	14	11	02	12	04	07	13	01	05	00	15	10	03	09	08	06
2	04	02	01	11	10	13	07	08	15	09	12	05	06	03	00	14
3	11	08	12	07	01	14	02	13	06	15	00	09	10	04	05	13

S-BOX 6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	01	10	15	09	02	06	08	00	13	03	04	14	07	06	11
1	10	15	04	07	07	12	09	05	06	01	13	14	00	11	03	08
2	08	14	15	05	02	08	12	05	07	00	04	10	01	13	11	06
3	04	15	02	12	09	05	15	10	11	14	10	07	10	00	08	13

S-BOX 7

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	04	11	02	14	15	00	08	13	03	12	09	07	05	10	06	01
1	13	00	11	07	04	09	01	10	14	03	05	12	02	16	08	06
2	01	04	00	13	12	03	07	14	10	15	06	08	00	05	09	02
3	06	11	13	08	01	04	10	07	09	05	00	15	14	02	03	12

S-BOX 8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	02	08	04	06	15	11	01	10	09	03	14	05	00	12	07
1	01	15	13	08	10	03	07	04	12	05	06	11	10	14	09	02
2	07	11	04	01	09	12	14	02	00	06	10	10	15	03	05	08
3	02	01	14	07	04	10	08	13	15	12	09	09	03	05	06	11

4. Straight Permutation (P-Box)

The last operation in the DES function is a straight permutation. It takes 32-bit input and produce 32-bit output, which is shown in Table 4.4.

Table 4.4. Straight Permutation (P-Box) Table

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

(c) Final Permutation

The following figure shows the final permutation (P-boxes). This permutation takes a 64-bit input and permute according to predefine rules. Here, few input ports and the corresponding output ports are shown. This permutation is inverse of initial permutation.

Fig. 4.10. Process of Final permutation

The final Permutation table read from left to right, top to bottom. For example, in the final permutation, the first bit in the input becomes the 40th bit in the output, which is shown in Table 4.5.

Table 4.5. Final Permutation Table

FINAL PERMUTATION							
40	08	48	16	56	56	64	32
39	07	47	15	55	23	63	31
38	06	46	14	54	22	62	30
37	05	45	13	53	21	61	29
36	04	44	12	52	20	60	28
35	03	43	11	51	19	59	27
34	02	42	10	50	18	58	26
33	01	41	09	49	17	57	25

(d) Round Key Generation

Round key generator creates sixteen rounds keys ($K_1, K_2, ..., K_{16}$) each of size 48 bits. An original cipher key of 64-bit is given in which 8 extra bits are the parity bits, which are dropped before the actual key generation process. After parity bit drop only 56 bits are left.

Parity Drop : The 64-bit original key is passed through a compression P-box. It drops the parity bit from the 64-bit original cipher key and permutes the rest bits according to parity bit table. The remaining 56-bit value is the actual cipher key which is used to generate round keys. The parity drop table is shown in Table 4.6.

Table 4.6. Parity Drop Table

57	49	41	33	25	17	09	01
58	50	42	34	26	18	10	02
59	51	43	35	27	19	11	03
60	52	44	36	63	55	47	39
31	23	15	07	62	54	46	38
30	22	14	06	61	53	45	37
29	21	13	05	28	20	12	34

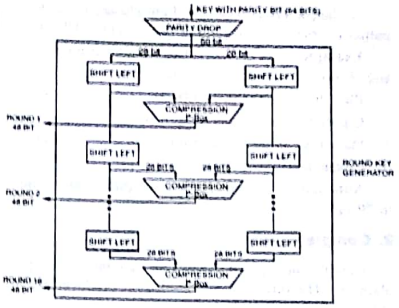


Fig. 4.11. Round Key Generation Process

The complete process of round key generation is shown in Fig. 4.11.

Shift Left

After parity bit drop 56-bit cipher key is divided into two halves each of size 28 bits. Each part is shifted left by one or two bits. In first, second, nineth and sixteenth round single bit shifts and in rest rounds, shifting is done using two bits. Later, these two parts are combined to form a 56-bit part.

Table 4.7. Number of bits Shift in each round

ROUND	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
BITS SHIFT	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Compression Permutation

The output after shift left operation is passed through a compression P-box. It permutes the 56-bits to 48 bits, which is round key K_1.

To generate other round keys shift left and compression permutation operation are performed again and again as shown in Fig. 4.11.

4.2.4. Decryption Process of DES

The only difference between encryption and decryption process is the reversal of key portions. If the original key was divided into $K_1, K_2, K_3, ..., K_{16}$ for the 16 encryption rounds, then for decryption, the key should be used as $K_{16}, K_{15}, ..., K_1$.

4.2.5. DES Analysis

Block cipher has two desired properties :

1. Avalanche Effect
2. Completeness

1. Avalanche Effect

Avalanche effect means a slight change in input (plaintext or key) results a drastic change in output (cipher text). Regarding this property, DES has been proved strong.

Example : Let us encrypt two plaintext block (with the same key) that differ only in one bit and observe the differences in the number of bits in each round.

Plain text: 0000000000000000 key: 22234512987ABB23
Cipher text: 4789FD476E82A5F1
Plain text: 0000000000000001 key: 22234512987A BB23
Cipher text: 0A4ED5C15A63FEA3

Although the two plaintext block differ only in the right most bit, the cipher text block differ in 29 bits.

2. Completeness

Completeness effect means that each bit of the cipher text needs to depend on many bits on the plaintext. The diffusion and confusion produced by P-boxes and S-boxes in DES, show a very strong completeness effect.

Strength of DES

1. The use of 56-bit keys

With a key length of 56 bits, there are 2^{56} possible keys, which is approximately $7.2*10^{16}$ keys. Thus, on the face of it, a brute-force attack appears impractical. Assuming that, on an 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.

2. The nature of the DES algorithm

Another concern is the possibility that cryptanalysis is possible by exploiting the characteristic of the DES algorithm. The focus of concern has been on the 8 substitution table, or s-boxes, that are used in each iteration. Because the design criteria for these boxes, and needed for the entire algorithm, were not need 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.

Weakness of DES

During the last few years critics have found some weaknesses in DES.

Weaknesses in Cipher Design

Here, some weaknesses that have been found in the design of the cipher are mentioned.

(*a*) Weakness in S-boxes
(*b*) Weakness in P-boxes
(*c*) Weakness in Key.

(*a*) **Weakness in S-boxes :** At least three weaknesses are mentioned in the literature for S-boxes.

- In S-box 4, the last three output bits can be derived in the same way as the first output bit by complementing some of the input bits.
- Two specifically chosen inputs to an S-box array can create the same output.

- It is possible to obtain the same output in a single round by changing bits in only three neighbouring *s*-boxes.

(*b*) **Weakness in P-boxes :** Two weaknesses were found in the design of *P*-boxes.

- It is not clear why designers use initial and final permutation, though these do not have any security benefit.
- In the expansion permutation the first and fourth bit of every 4-bit series are repeated.

(*c*) **Weakness in Key :** Several weaknesses have been found in the cipher key. To do a brute-force attack on a given ciphertext block, the adversary needs to check 2^{56} keys (roughly equal to 7.2×10^{16} keys). With available technology, it is possible to check one million keys per second. This means that we need more than two thousand years to do brute force attacks on DES using only a computer with one processor. Four out of 2^{56} possible keys are called weak keys. A weak key is the one that, after parity drop operation consists either of all 0's, all 1's or half 0's and half 1's.

Keys before parity drop (64 bits)	Actual key (56 bits)
0101 0101 0101 0101	000000 000000
1F1F1 1F1F 0E0E 0E0E	000000 FFFFFF
E0E0 E0E0 F1F1 F1F1	FFFFFF 000000
FEFE FEFE FEFE FEFE	FFFFFF FFFFFF

Security of DES

Various minor cryptanalytic properties are known, and three theoretical attacks are possible in which three are interested: brute-force attack, differential cryptanalysis and linear cryptanalysis.

Brute-Force Attack

As we discuss earlier the weakness of short cipher key in DES. In academics, various proposals for a DES-cracking machine were advanced. In 1977, Diffie and Hellman proposed a machine costing an estimated US$20 million which could find a DES key in a single day. By 1993, Wiener had proposed a key-search machine costing US$1 million which would find a key within 7 hours. So, it is clear that DES can be broken using 2^{55} encryptions. However, most of the applications use either 3DES with key size of 112 or with key size of 168 which make DES resistant to brute –force attacks.

Differential Cryptanalysis

Differential cryptanalysis was first published by Sean Murphy, Eli Biham and Adi Shamir in 1990, but it was known to the National Security Agency as far back as the early—1970's. Differential Cryptanalysis is the first published attack that is capable of breaking DES in less than 2^{55} complexities. This scheme can successfully cryptanalyze DES with an effort on the order of 2^{47} encryptions, requiring 2^{47}chosen plaintext. Although differential cryptanalysis is a powerful tool, it does not do very well against DES. The need to strengthen DES against attacks using cryptanalysis played a large part in the design of the *S*-boxes and permutation *P*. It has also been shown that increasing the number of rounds to 20 require more than 2^{64} chosen plaintexts for this attack, which is impossible because the possible number of plaintext blocks in DES is 2^{64}.

The differential cryptanalysis of DES is complex. A complete description of this attack was given by ELI BIHAM.

Consider the original plaintext block m has two halves m^1 and m^2. Each round of DES Maps the right hand input to the left hand output and this right output is a function of the left hand input and the sub key for this round. So, at each round, only one new 32-bit block is created. Let each new block is denoted by m_i, $i = 1, 2, 3, \ldots 16$, then the intermediate message halves are related as follows :

$$m_{i+1} = m_{i-1} \oplus (m_i, k_i) \qquad i = 1, 2, \ldots, 16$$

Consider two messages m^1 and m^2 such that $\Delta m = m^1 \oplus m^2$, where Δm is the XOR difference of m^1 and m^2. Now we can assume that

$$\Delta m_i = m^1{}_i \oplus m^2{}_i,$$

and

$$\begin{aligned}\Delta m_{i+1} &= m^1{}_{i+1} \oplus m^2{}_{i+1}, \\ &= [m^1{}_{i-1} \oplus f(m^1{}_i, K_i)] \oplus [m^2{}_{i-1} \oplus f(m^2{}_i, K_i)] \\ &= \Delta m_{i-1} \oplus [m^1{}_i, K_i) \oplus f(m^2{}_i, K_i)]\end{aligned}$$

The propagation of differences through three rounds of DES.

Linear Cryptanalysis

In cryptography, linear cryptanalysis is a general form of cryptanalysis based on finding affine approximations to the action of a cipher. Attacks have been developed for block ciphers and stream ciphers. Linear cryptanalysis is one of the two most widely used attacks on block ciphers; the other being differential cryptanalysis. *S*-boxes are not very resistant to linear cryptanalysis. It has been shown that DES can be broken using 2^{43} pairs of known plaintexts.

4.3. MULTIPLE DES

DES is the most widely used symmetric algorithm in the world, despite claims that the key length is too short. Ever since DES was first announced, controversy has raged about whether 56 bits is long enough to guarantee security. With available technology and the possibility of parallel processing, a brute force attack on DES is feasible. Those who consider the exhaustive key-search attack to be a real possibility (and to be fair the technology to do such a search is becoming a reality) can overcome the problem by using double or triple length keys. Consequently, two main variations of DES have merged, which are **Double DES** and **Triple DES.**

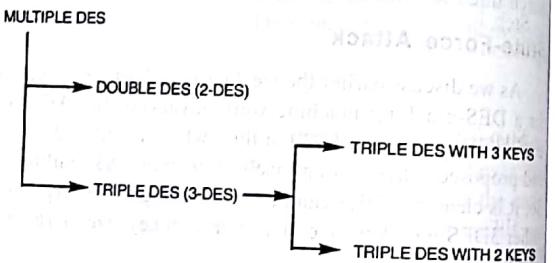


Fig. 4.12. *Multiple DES*

4.3.1. Double DES

Double DES is quite simple to understand. It does twice what DES normally does only once. Double DES uses two keys K_1 and K_2. It first performs DES encryption process on the original plaintext using key K_1 to get the ciphertext (say C_1). It again performs DES encryption process on the ciphertext (C_1) but this time with the other key K_2. The final output is the ciphertext.

Double DES is the simplest form of multiple encryptions having a 112-bit key and enciphers blocks of 64 bits. Mathematically double DES can be shown as given below:

$$p \rightarrow E(k_1, p) \rightarrow E(k_2, E(k_1, p)) = C$$

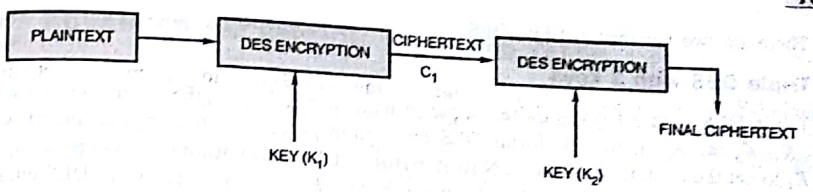


Fig. 4.13. *Encryption Process of 2-DES*

Meet-in-the–middle attack on double DES

Merkle and Hellman introduce the concept of meet-in-the-middle attack. This attack requires knowing some plaintext/ciphertext pairs. Let's assume that we have a plaintext/ciphertext pair; *i.e.*, we know the plaintext p and the corresponding (double DES enciphered) ciphertext C.

In this attack, attacker tries to break from both directions *i.e.*, from P to C and C to P. That's why name is meet-in-middle attack.

Attacks on DES have typically been brute force attacks (see "Breaking DES"); so, we will use brute force here.

The double DES encryption is:

$$p \rightarrow E(k_1, p) \rightarrow E(k_2, E(k_1, p)) = C$$

Attacker try to break in step 1 and step 2 as discussed below:

Step 1 : Attacker encrypt p using all 2^{56} possible keys, and store the results. (Storage could be a problem). The stored results will include all possible encryptions

$$p \rightarrow E(k_1, p)$$

Step 2 : Attacker decrypt C using all 2^{56} possible keys.

$$D(k_2, C) = D(k_2, E(k_2, E(k_1, p))) \rightarrow E(k_1, p)$$

After decrypting with each key, check for a match with the stored outputs of the 2^{56} possible encryptions (Step 1). When we have a match, we have located a possibly correct pair of keys. Now, perhaps more than one pair of keys will result in a match, but the number of pairs of keys that return matches should be small. We could try each possible pair of keys. If more than one plaintext/ciphertext correspondence is known (for the key pair), then other correspondences could be used to check which of the keys is correct.

Times required to break 2-DES using Meet-in-middle attack:

So, it only takes twice as long to break double DES using brute force.

$$\text{Time required} = 2^{56} \text{ (for step 1)} + 2^{56} \text{ (for step 2)} = 2 * 2^{56} = 2^{57}$$

NOTE

Meet-in-middle attack makes 2-DES useless.

4.3.2. Triple DES

3-DES was developed in 1999 by IBM – by a team led by Walter Tuchman. 3-DES prevents a meet-in-the-middle attack.

There are two versions of triple DES:

(a) Triple DES with 3 keys

Triple DES with 3 keys is quite simple to understand. Triple DES with 3 keys uses three keys K_1, K_2 and K_3. It first performs DES encryption process on the original plaintext using key K_1 to get the ciphertext (say C_1). Now it performs DES **decryption** process on the ciphertext (C_1) but this time with the second key K_2 to get the text P_1. Now again performs DES encryption process on the text (P_1) but this time with the third key K_3. The final output is the ciphertext.

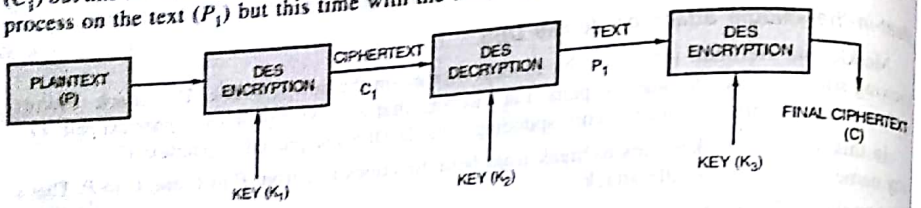


Fig. 4.14. Triple DES with 3 Keys

Triple DES with 3 keys has a 168-bit key and enciphers blocks of 64 bits. Mathematically Triple DES with 3 keys can be shown as given below:

$$C = E(k_3, D(k_2, E(k_1, p)))$$

Drawback of Triple DES with 3 keys

Triple DES with 3 keys has a drawback of requirement of 168 bits (56*3) for the key which can be slightly difficult in practical situation.

(b) Triple DES with 2 keys

Difficulty of 168-bit key with 3-DES with 3 keys a work around suggested by Tuchman, uses just 2 keys for triple DES.

Triple DES with 2 keys uses two keys K_1, K_2. It first performs DES encryption process on the original plaintext using key K_1 to get the ciphertext (say C_1). Now it performs DES decryption operation on C_1 using second key K_2. It again performs DES encryption process on the plaintext (P_1) but this time with the first key K_1 to get the ciphertext. The final output is the ciphertext.

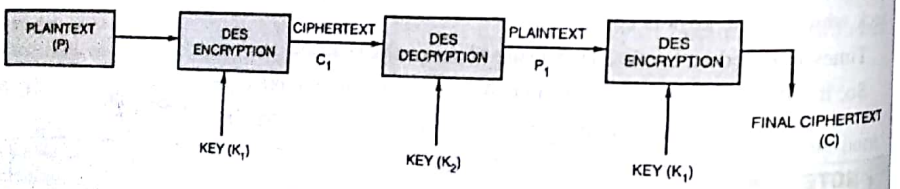


Fig. 4.15. Triple DES with 2 Keys

Triple DES with 2 keys has a 112-bit key and enciphers blocks of 64 bits. Mathematically Triple DES with 2 keys can be shown as given below:

$$C = E(k_1, D(k_2, E(k_1, p)))$$

4.4. ADVANCED ENCRYPTION STANDARD (AES)

In cryptography, the **Advanced Encryption Standard (AES)**, also known as **Rijndael**, is a block cipher adopted as an encryption standard by the U.S. government. AES was announced by National Institute of Standards and Technology (NIST) as U.S. FIPS PUB 197 (FIPS 197) on November 26, 2001 after a 5-year standardization process.

4.4.1. Introduction

AES is one of the most popular algorithms used in symmetric key cryptography. It is available by choice in many different encryption packages. The cipher was developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, and submitted to the AES selection process under the name "Rijndael".

Unlike DES, its predecessor, Rijndael is a substitution-permutation network, not a Feistel network. AES is fast in both software and hardware, is relatively easy to implement, and requires little memory. As a new encryption standard, it is currently being deployed on a large scale.

AES is not precisely Rijndael (although in practice they are used interchangeably) as Rijndael supports a larger range of block and key sizes; AES has a fixed block size of 128 bits and a key size of 128, 192, or 256 bits, whereas Rijndael can be specified with key and block sizes in any multiple of 32 bits, with a minimum of 128 bits and a maximum of 256 bits.

Due to the fixed block size of 128 bits, AES operates on a 4×4 array of bytes, termed the *state* (versions of Rijndael with a larger block size have additional columns in the state). Most AES calculations are done in a special finite field.

AES is founded on solid and well-published mathematical ground, and appears to resist all known attacks well. There's a strong indication that in fact no back-door or known weakness exists since it has been published for a long time, has been the subject of intense scrutiny by researchers all over the world, and such enormous amounts of economic value and information is already successfully protected by AES. There are no unknown factors in its design, and it was developed by Belgian researchers in Belgium therefore voiding the conspiracy theories sometimes voiced concerning an encryption standard developed by a United States government agency. 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.

4.4.2. AES Cipher

The AES cipher is specified as a number of repetitions of transformation rounds that convert the input plaintext into the final output of ciphertext. Each round consists of several processing steps, including one that depends on the encryption key. A set of reverse rounds is applied to transform ciphertext back into the original plaintext using the same encryption key.

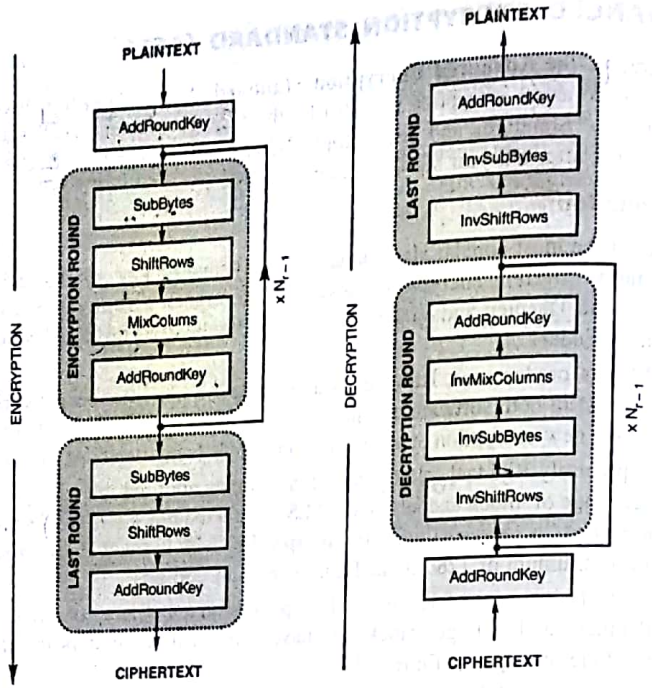


Fig. 4.16. *Encryption/Decryption process of AES*

AES is a non-Feistel cipher that encrypts and decrypts a data block of 128 bits. It uses 10,12 or 14 rounds. The key size 128, 192 or 256 will depend on number of rounds. In Fig. 4.16 N_r defines the number of rounds. The number of round keys generated by the key expansion algorithm is always one more than the number of rounds, means we need N_{r+1} for all rounds.

AES do not use a Feistel structure but process the entire data block in parallel during each round using substitutions and permutation shown in Fig. 4.16.

1. 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.
2. 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).
   - **Add Round Key:** A simple bitwise XOR of the current block with a portion of the expanded key.
3. 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.
4. 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.

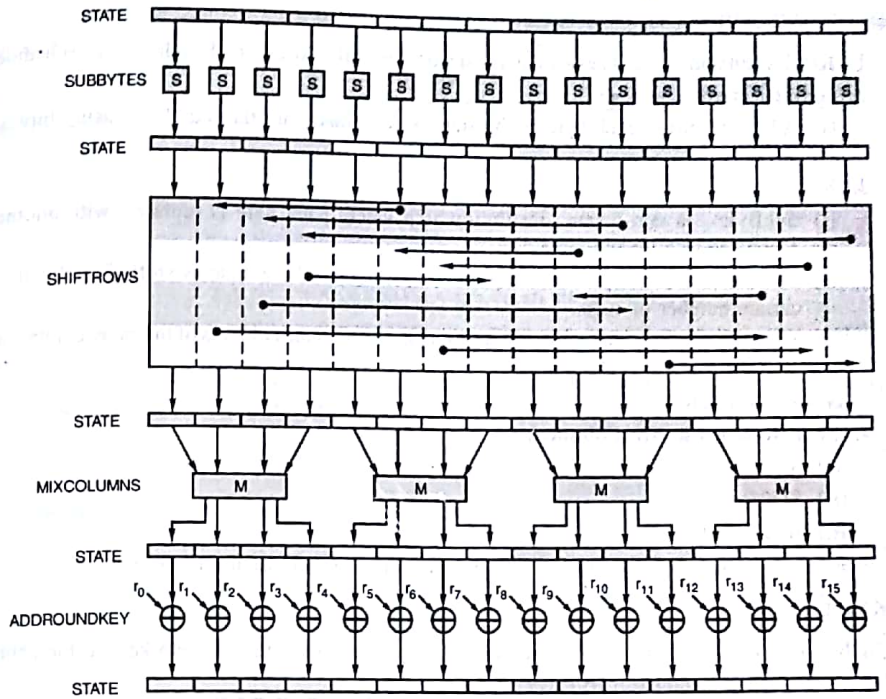


Fig. 4.17. *AES Encryption Round*

5. 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.
6. 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 $A \oplus A \oplus B = B$.
7. 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.
8. Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext.
9. 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.

Steps in Algorithm

1. **Key Expansion**—round keys are derived from the cipher key using Rijndael's key schedule.
2. **Initial Round**
   (i) AddRoundKey—each byte of the state is combined with the round key using bitwise XOR.
3. **Rounds :**
   (i) SubBytes—a non-linear substitution step where each byte is replaced with another according to a lookup table.
   (ii) ShiftRows—a transposition step where each row of the state is shifted cyclically a certain number of steps.
   (iii) MixColumns—a mixing operation which operates on the columns of the state, combining the four bytes in each column.
   (iv) AddRoundKey
4. **Final Round (no MixColumns) :**
   (i) SubBytes
   (ii) ShiftRows
   (iii) AddRoundKey

Note: In last round there is no MixColumns step, but in others round it will be:

1. Key Expansion

In the case of the AES, there are a number of rounds, each needing its own key, so the actual key is "stretched out" and transformed to give portions of key for each round. The key expansion routine shown in Fig. 4.18, as part of the overall AES algorithm, takes an *input key* of **4*Nk** bytes, or *Nk* 32-bit words. **Nk** has value either **4**, **6**, or **8**. The output is an *expanded key* (denoted **w**) of **4*Nb*(Nr+1)** bytes, where **Nb** is always **4** and **Nr** is the number of *rounds* in the algorithm, with **Nr** equal **10** in case **Nk** is **4**, **Nr** equal **12** in case **Nk** is **6**, and **Nr** equal **14** in case **Nk** is **8**.

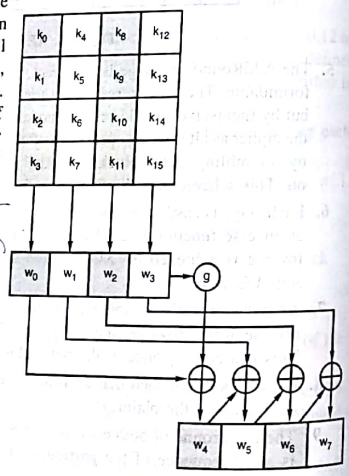


Fig. 4.18. *Key Expansion*

AES key expansion consists of several primitive operations:

1. Rotate – takes a 4-byte word and rotates everything one byte to the left, *e.g.* rotate ([1,2,3,4]) → [2, 3, 4, 1]
2. SubBytes – each byte of a word is substituted with the value in the S-Box whose index is the value of the original byte.
3. Rcon – the first byte of a word is XORed with the **round constant**. Each value of the Rcon table is a member of the Rinjdael finite field.

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	After XOR with Rcon	*w*[*i* 4]	*w*[*i*] = temp ⊕ *w*[*i* 4]
36	7F8D292F	8D292F7F	5DA515D2	1B000000	46A515D2	EAD27321	AC7766F3

2. AddRoundKey

In the AddRoundKey step, each byte of the state is combined with a byte of the round subkey using the XOR operation. For each round, a subkey is derived from the main key using Rijndael's key schedule; each subkey is the same size as the state. The subkey is added by combining each byte of the state with the corresponding byte of the subkey using bitwise shown in Fig. 4.19.

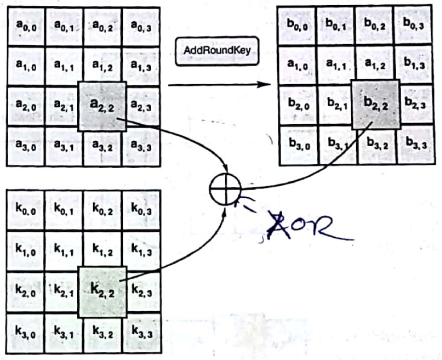


Fig. 4.19. *AddRound Key Step*

3. Round

There will be number of rounds depending upon AES version. There are four steps in each round. In last round there is no MixColumns step, but in others round it will be. The four steps are explained below.

(*i*) **SubBytes :** In the SubBytes step shown in Fig. 4.20, each byte in the array is updated using an 8-bit substitution box, the Rijndael S-box. This operation provides the nonlinearity in the cipher. The S-box used is derived from the multiplicative inverse over **GF(2^8)**, known to have good non-linearity properties. To avoid attacks based on simple algebraic properties, the S-box is constructed by combining the inverse function with an invertible affine transformation. The S-box is also chosen to avoid any fixed points (and so is a derangement), and also any opposite fixed points.

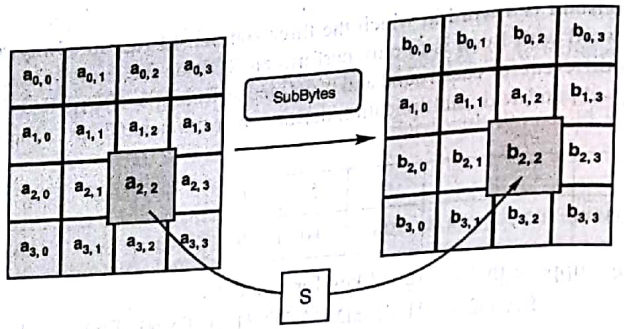


Fig. 4.20. *AES Sub Byte Operation*

(*ii*) **ShiftRows :** The ShiftRows step shown in Fig. 4.21 operates on the rows of the state; it cyclically shifts the bytes in each row by a certain offset. For AES, the first row is left unchanged. Each byte of the second row is shifted one to the left. Similarly, the third and fourth rows are shifted by offsets of two and three respectively. For the block of size 128 bits and 192 bits the shifting pattern is the same. In this way, each column of the output state of the ShiftRows step is composed of bytes from each column of the input state. (Rijndael variants with a larger block size have slightly different offsets).

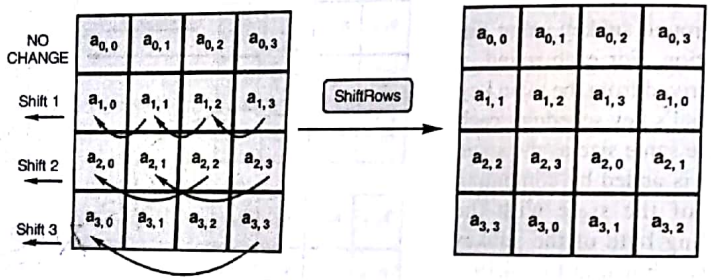


Fig. 4.21. *ShiftRows step*

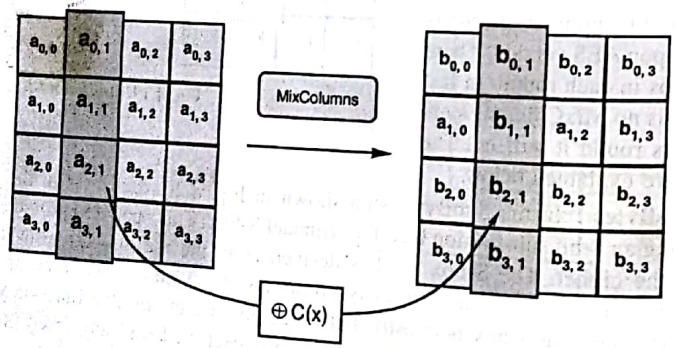


Fig. 4.22. *MixColumn Step*

In the case of the 256-bit block, the first row is unchanged and the shifting for second, third and fourth row is 1 byte, 3 bytes and 4 bytes respectively—this change only applies for the Rijndael cipher when used with a 256-bit block, as AES does not use 256-bit blocks.

(*iii*) **The MixColumns step :** In the MixColumns step shown in Fig. 4.21, each column of the state is multiplied with a fixed polynomial $c(x)$. The MixColumns function takes four bytes as input and outputs four bytes, where each input byte affects all four output bytes. Each column is treated as a polynomial over $\mathbf{GF}(2^8)$ and is then multiplied modulo $x^4 + 1$ with a fixed polynomial $c(x) = 3x^3 + x^2 + x + 2$.

Optimization of the Cipher

On systems with 32-bit or larger words, it is possible to speed up execution of this cipher by combining SubBytes and ShiftRows with MixColumns, and transforming them into a sequence of table lookups. This requires four 256-entry 32-bit tables, which utilizes a total of four kilobytes (4096 bytes) of memory—one kilobyte for each table. A round can now be done with 16 table lookups and 12 [32-bit exclusive-or operations], followed by four 32-bit exclusive-or operations in the AddRoundKey step.

If the resulting four kilobyte table size is too large for a given target platform, the table lookup operation can be performed with a single 256-entry 32-bit table by the use of circular rotates. Using a byte-oriented approach it is possible to combine the SubBytes, ShiftRows, and MixColumnsteps into a single round operation.

4.4.3. Analysis of AES

Most of the known attacks on DES were already tested on AES and none of them has broken the security of AES so far.

1. Brute-force Attacks

AES is definitely more secure than DES due to the larger size key (128, 192, and 256 bits). Let us compare DES with 56-bits cipher key and AES with 128-bit cipher key. For DES we need 2^{56} (ignoring the key complement issue) tests to find the key: for AES we need 2^{128} tests to find the key. This means that if we can break DES in t seconds, we need (2^{72} * t) seconds to break AES. This would be almost impossible.

2. Statistical Attacks

The strong diffusion and confusion provided by the combination of the SubBytes, ShiftRows and MixColumns transformations removes any frequency pattern in the plaintext. Numerous tests have failed to do statistical analysis of the ciphertext.

3. Differential and Linear Attacks

AES was designed after DES. Differential and linear cryptanalysis attacks were no doubt taken into consideration. There are no differential and linear attacks on AES as yet.

4.4.4. Security of AES

As of 2006, the only successful attacks against AES implementations have been side channel attacks :

Side channel attacks do not attack the underlying cipher and so have nothing to do with its security as described here, but attack implementations of the cipher on systems which inadvertently leak data.

In June 2003, the US Government announced that AES may be used for classified information:

"The design and strength of all key lengths of the AES algorithm (*i.e.*, 128, 192 and 256) are sufficient to protect classified information up to the SECRET level. TOP SECRET information will require use of either the 192 or 256 key lengths. The implementation of AES in products intended to protect national security systems and/or information must be reviewed and certified by NSA prior to their acquisition and use."

The most common way to attack block ciphers is to try various attacks on versions of the cipher with a reduced number of rounds. AES has 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. By 2006, the best known attacks were on 7 rounds for 128-bit keys, 8 rounds for 192-bit keys, and 9 rounds for 256-bit keys.

4.5. IDEA ALGORITHM (INTERNATIONAL DATA ENCRYPTION ALGORITHM)

4.5.1. Introduction

In Cryptography, the IDEA is a block cipher designed by Xuejia Lai and James Massey of ETH Zurich, SWITZERLAND and was first described in 1991. The algorithm was intended as a replacement for the Data Encryption Standard. IDEA is a minor revision of an earlier cipher, PES (Proposed Encryption Standard); IDEA was originally called IPES (Improved PES).

The cipher was designed under a research contract with the Hasler Foundation, which became part of Ascom-Tech AG. The cipher is patented in a number of countries but is freely available for non-commercial use. The name "IDEA" is also a trademark. The patents will expire in 2010–2011. Today, IDEA is licensed worldwide by Media Crypt.

IDEA was used in Pretty Good Privacy (PGP) v2.0, and was incorporated after the original cipher used in v1.0, BassOmatic, was found to be insecure. IDEA is an optional algorithm in the Open PGP standard.

IDEA was to develop a strong encryption algorithm, which would replace the DES procedure developed in the U.S.A. in the seventies. It is also interesting in that it entirely avoids the use of any lookup tables or S-boxes. When the famous PGP email and file encryption product was designed by Phil Zimmermann, the developers were looking for maximum security. IDEA was their first choice for data encryption based on its proven design and its great reputation.

4.5.2. Operation

IDEA operates on 64-bit blocks using a 128-bit key, and consists of a series of eight identical transformations and an output transformation (the *half-round*). The processes for encryption and decryption are similar.

The IDEA algorithm is interesting in its own right. It includes some steps which, at first, make it appear that it might be a non-invertible hash function instead of a block cipher. The substitution boxes and the associated table lookups used in the block ciphers available to-date have been completely avoided.

IDEA is a block cipher which uses a 128-bit length key to encrypt successive 64-bit blocks of plaintext. The procedure is quite complicated using sub keys generated from the key to carry out a series of modular arithmetic and XOR operations on segments of the 64-bit plaintext block.

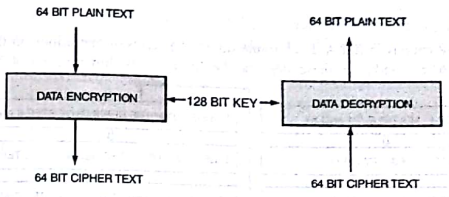


Fig. 4.23. *General Concept of IDEA*

4.5.3. Working of IDEA

Idea is a block cipher. It uses both diffusion and confusion for encryption. The 64-bit input plain text block is divided into 4 portions. Each of size 16 bits says P1 to P4. These are the inputs for rounds 1st, there are 8 such rounds. The key consist of 128 bits in each round. After 8 rounds output transformation is performed. The output of this will be the cipher text of 64-bit plaintext block. The working of this algorithm is shown in Fig. 4.24. The encryption process can be divided into four parts.

1. Rounds
2. Sub-key generation for rounds
3. Output transformation
4. Sub-key generation for Output transformation

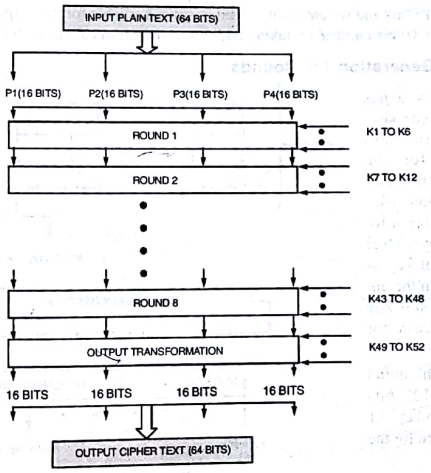


Fig. 4.24. *Steps of IDEA Algorithm*

1. Rounds

There are 8 rounds in IDEA. Each round involves a series of operations on the four data blocks using 6 keys. At a broad level these steps can be described as shown in Fig. 4.25.

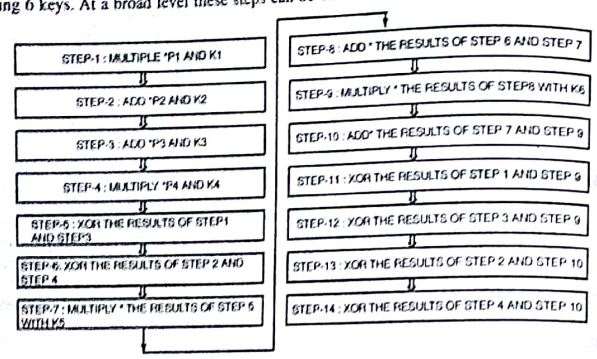


Fig. 4.25. Details of One Round in IDEA

Here, Add* and Multiply* are not only addition and multiplication .these are addition modulo 2^{16} and multiplication modulo $2^{16} + 1$. This is required as normal addition will produce a number that consists of 17 bits and we have only 16 bit position available for the output. So we must reduce this number to a 16 bit number by taking addition and multiplication modulo.

2. Sub Key Generation for Rounds

Each of the 8 rounds makes use of 6 sub keys so 8 * 6 = 48 sub keys are required for the rounds and the final output transformations uses 4 sub keys making a total of 52 sub keys. Over all from an input key of 128 bits based on the understanding of sub key generation process for first two rounds.

In round 1 the initial key consist of 128 bits from which 6 sub keys k1 to k6 are generated for the first round.

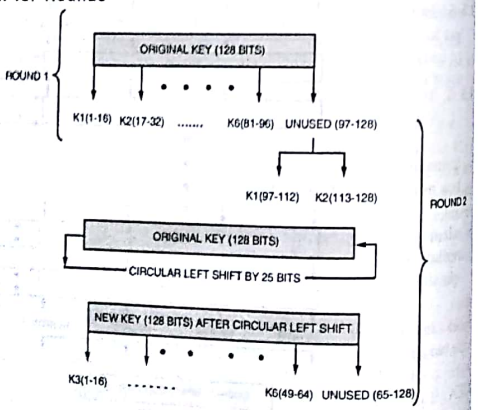


Fig. 4.26. Sub-key Generation

In round 2^{nd} there are 32 unused bits (97-128). Each round requires 6 sub keys each of 16 bits making a total of 96 bits. For the 2^{nd} round we still require 96 – 32 = 64 more bits. However we have already exhausted all the 128 bits of original key. At this stage the original key Is shifted left circular by 25 bits *i.e.*, the 26 bits of original key moves to the first position and becomes the 1^{st} bit after the shift and 25^{th} bit of the original key moves to the last position and becomes the 128^{th} bit after the shift.

3. Output Transformation

It is one-time operation. This is as usual a 64-bit value divided into 4 sub blocks say R1->R4. 4 sub keys are applied here.

Following are the steps:

```
STEP1: MULTIPLY R1 & K1
STEP2: ADD R2 & K2
STEP3: ADD R3 & K3
STEP4: MULTIPLY R4 & K4
```

4. Sub-key Generation for Output Transformation

This process is exactly similar to the sub-key generation process for eight rounds. At the end of the eighth and the final round, the key was exhausted. Hence the key is again shifted by 25 bits. Post this shift operation, first 64 bits of key are taken for the sub-keys k1 to k4 for the final output transformation round.

4.5.4. IDEA Decryption

The decryption process of IDEA is same as Encryption process. There are some alterations in the generation and pattern of sub-keys. Sub-keys used in decryption process are inverse of the sub-keys used in encryption process.

4.5.5. Idea Modes of Operation

Idea can work with any mode of block cipher operation. It is part of PGP (Pretty Good Privacy) for securing e-mails and used in encrypting sensitive data.

4.5.6. Weak keys for IDEA

A large classes of weak keys have been found for the block cipher algorithm IDEA. IDEA has a 128-bit key and encrypts blocks of 64 bits. For a class of 223 keys IDEA exhibits a linear factor. For a certain class of 235 keys the cipher has a global characteristic with probability 1. For another class of 251 keys only two encryptions and solving a set of 16 nonlinear boolean equations with 12 variables is sufficient to test if the used key belongs to this class. If it does, its particular value can be calculated efficiently. It is shown that the problem of weak keys can be eliminated by slightly modifying the key schedule of IDEA.

Two new attacks on a reduced number of rounds of IDEA are presented: truncated differential attack and differential-linear attack. The truncated differential attack finds the secret key of 3.5 rounds of IDEA in more than 86% of all cases using an estimated number of 2^{56} chosen plaintexts and a workload of about 2^{67} encryptions of 3.5 rounds of IDEA. With 2^{40} chosen plaintexts the attack works for 1% of all keys. The differential-linear attack finds the secret key of 3 rounds of IDEA. It needs at most 2^{29} chosen pairs of plaintext and a workload of about 2^{44} encryptions with 3 rounds of IDEA.

4.5.7. Strength of IDEA

The designers analyzed IDEA to measure its strength against differential cryptanalysis and concluded that it is immune under certain assumptions. No successful linear or algebraic weaknesses have been reported. As of 2007, the best attack which applies to all keys can break IDEA reduced to 6 rounds (the full IDEA cipher uses 8.5 rounds). Note that a "break" is any attack which requires less than 2^{128} operations; the 6-round attack requires 2^{64} known plaintexts and $2^{126.8}$ operations. would take one billion computers testing one billion combinations per second, 10 000 billion y to crack the code (2^{128} variants)—longer than the universe has existed.

4.6. RC4 ALGORITHM

RC4 is a stream cipher designed in 1987 by Ron Rivest for RSA Security. It is a variable k size stream cipher with byte-oriented operations.

The algorithm is based on the use of a random permutation. Analysis shows that the period the cipher is over whelmingly likely to be greater than 10^{100} [ROBS95a]. Eight to sixteen machine operations are required per output byte, and the cipher can be expected to run very quickly in software. RC4 is used in the SSL/TLS (Secure Sockets Layer/Transport Layer Security) standards that have been defined for communication between Web browsers and servers. It is also used in WEP (Wired equivalent Privacy) protocol and the newer WiFi Protected Access (WPA) protocol that are part of the IEEE 802.11 wireless LAN standard. RC4 was kept as a trade secret by RSA Security. In September 1994, the RC4 algorithm was anonymously posted on the Internet on the Cypherpunks anonymous remailers list. Following steps explain the working of RC4.

- RC4 uses a variable length key from 1 to 256 bytes to initialize a 256-byte state table. The elements S[0], S[1],....S[255].
- The state table is used for subsequent generation of pseudo-random bytes and then to generate a pseudo-random stream (i.e. key stream).
- Then, pseudo-random stream is XORed with the plaintext to give the ciphertext. Each element in the state table is swapped at least once.
- The RC4 algorithm works in two phases, **key setup and ciphering**. Key setup is the first and most difficult phase of this algorithm. During a K-bit key setup (K being your key length), the encryption key is used to generate an encrypting variable using two arrays state and key, and K-number of mixing operations. These mixing operations consist of swapping bytes, modulo operations, and other formulas. A modulo operation is the process of yielding a remainder from division.
- Once the encrypting variable is produced from the key setup, it enters the ciphering phase where it is XORed with the plain text message to create an encrypted message. XOR is the logical operation of comparing two binary bits. If the bits are different, the result is 1. If the bits are the same, the result is 0. Once the receiver gets the encrypted message, it decrypts it by XORing the encrypted message with the same encrypting variable.

Key Set Up

1. Initialization of *S* : To begin, the entries of S are set equal to the values from 0 through 255 in ascending order; that is; S[0] = 1,....S[255]. A temporary vector, T, is also created. If the length of the key K is 256 bytes, then K is transferred to T. Otherwise, for a key of length keylen bytes, the first keylen elements of T are copied from K and then K is repeated as many times as necessary to fill out T. At the end T should be completely filled.

After this T is used to produce initial permutation of S. For this, a loop executes which ranges i from 0 to 255. In each case, the byte at the position S[i] is swapped with another byte in the S array. As per an arrangement decided by T[i] which used the following logic.

```
j=0;
for i=0 to 255
j=( j + S[i] + T[i] ) mod 256;
swap( S[i],S[j] );
```

Here, the values of S are simply rearranged.

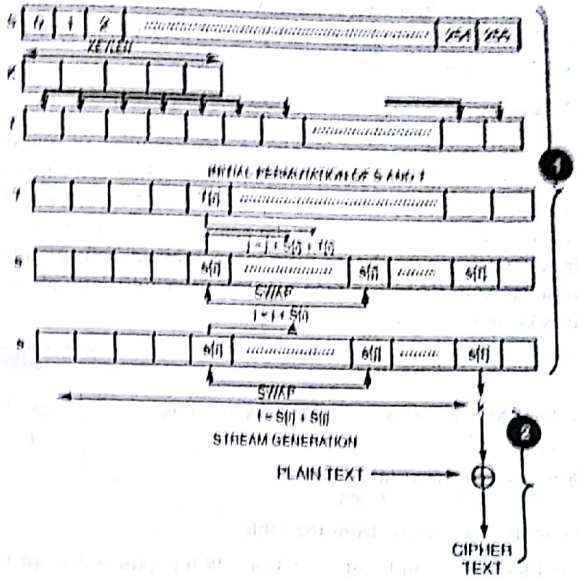


Fig. 4.27. *Working of RC4*

2. Stream Generation : Once the S vector is initialized, the input key is no longer used. Stream generation involves cycling through all the elements of S[i] with another byte in S according to a scheme dictated by the current configuration of S. After S[255] is reached, the process continues, starting from S[0].

Ciphering

After K has been created, K is XORed with a byte of plaintext for encryption and XOred with byte of ciphertext for decryption.

Strength of RC4: A numbers of papers have been published analyzing methods of attacking RC4 [e.g., [KNUD98],[FLUH00],[MANT01]). None of these approaches is practical against RC4 with a reasonable key length, such as 128 bits. A more serious problem is reported in [FLUH01]. The authors demonstrate that the WEP protocol, intended to provide confidentiality on 802.11 wireless LAN networks, is vulnerable to a particular attack approach. In essence, the problem is not with RC4 itself but the way in which keys are generated for use as input to RC4.

4.7. RC5 ALGORITHM

4.7.1. Background and History

RC5 is a fast, symmetric key block encryption algorithm developed by Ron Rivest. The ... features of RC5 are that it is quite fast as it uses only the primitive computer operations (such ... addition, XOR, shift, etc.). A novel feature of RC5 is the heavy use of data dependent rotation... Means it allows for a variable number of rounds, and a variable bit-size key to add to the flexibil... Different applications that demand varying security needs can set these values accordingly. A... important aspect is that RC5 requires less memory for execution, and is therefore suitable f... hardware and software implementations.

4.7.2. How RC5 Works

In RC5, the word size (*i.e.* input plain text block size), number of rounds and number of 8-... bytes (octets) of the key, all can be of variable length. These values can consist of the sizes as show... in the table. Of course, once decided, these values remain the same for a particular execution of ... cryptographic algorithm. These are variable in the sense that before the execution of a particu... instance of RC5, these values can be chosen from those allowed. This is unlike DES, for instance... where the block size must be of 64 bits and the key size must always be of 56 bits; or unlike IDE... which uses 64-bit blocks and 128-bit keys.

Parameter	Allowed values
Word size in bits (RC5 encrypts 2-word Blocks at a time)	16, 32, 64
Number of rounds	0-255
Number of 8-bit bytes (octets) in the key	

The following conclusions emerge from the table :

- The plain text block size can be of 32, 64 or 128 bits (since 2-word blocks are used).
- The key length can be 0 to 2040 bits (since we have specified the allowed values for 8-bit key...

The output resulting from RC5 is the cipher text, which has the same size as the input plai... text. Since RC5 allows for variable values in the three parameters, as specified a particular instanc... of the RC5 algorithm is denoted as RC5-*w*/*r*/*b*, where *w* = word size in bits, r = number of round... b = number of 8-bit bytes in the key. Thus, if we have RC5-32/16/16, it means that we are usin... RC5 with a block size of 64-bits (remember that RC5 uses 2-word blocks), 16 rounds of encryptio... and 16 bytes (*i.e.*, 128 bits) in the key. Rivest has suggested RC5-32/12/16 as the minimum safe... version.

4.7.3. Working of RC5

As shown in the Fig. 4.28, RC5 consist one initial operation consisting of two steps, then ... number of rounds. Here, the number of rounds (r) can vary from 0 to 255.

The same principles of operations will apply to other block sizes, In the first two steps o... the one-time initial operation, the input plain text of 64 bit is divided into two 32-bit blocks ... and B.

S[0] and S[1] are added to A and B, respectively. This produces C and D respectively, and marks the end of the one-time operation. Then, the rounds begin. In each round, there are following operations:

- Bitwise XOR
- Left circular-shift
- Addition with the next sub-key, for both C and D. This is the addition operation first, and then the result of the addition mod 2^*w* (since *w* = 32 here, we have 2^32) is performed.

If we observe the operations shown in the last figure carefully, we will note that the output of one block is fed back as the input to another block, making the whole logic quite complicated to decipher.

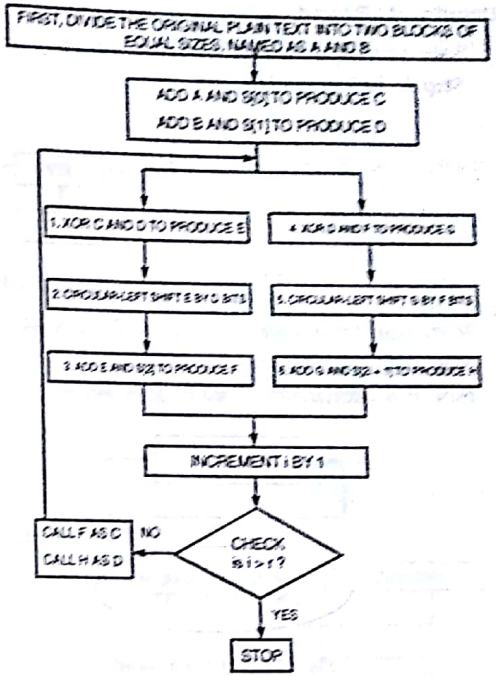


Fig. 4.28. Encryption Steps of RC5

One-Time Initial Operation

This consists of two steps: first, the input plain text is divided into two equal-sized blocks, A and B. Then the first sub-key, *i.e.* S[0] is added to A, and the second sub-key, *i.e.* S[1] is added to B. These operations are mod 2^32, and produce C and D, respectively.

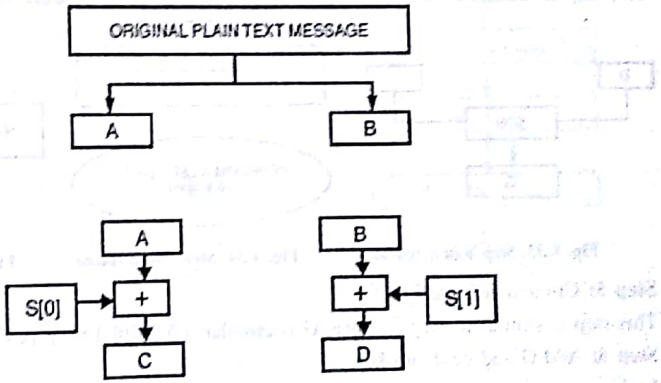


Fig. 4.29. One-Time Initial Operation in RC5

Details of Round

Same process as used in first round will apply for further rounds.

Step 1: XOR C and D

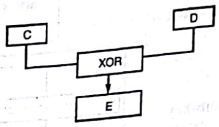


Fig. 4.30. Step 1 in Round

In the first step of each round, C and D are XORed together to form E.

Step 2: Circular-left shift E

Now, E is circular-left shifted by D positions

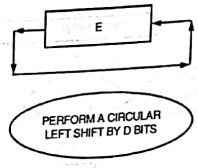


Fig. 4.31. Step 2 in a round

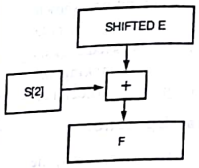


Fig. 4.32. Step 3 in a round

Step 3: Add E and next sub-key

In this step, E is added to the next sub-key (which is S[2] for the first round, and S[2i] in general, for any round, where I starts with 1. The output of this process is F.

Step 4: XOR D and F

This step is similar to step 1. Here, D and F are XORed to produce G.

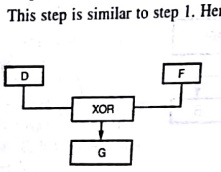


Fig. 4.33. Step 4 in a round

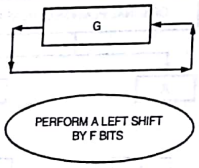


Fig. 4.34. Step 5 in a round

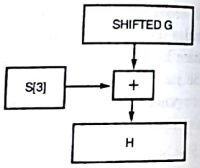


Fig. 4.35. Step 6 in a round

Step 5: Circular-left shift G

This step is similar to step 2. Here, G is circular-left shifted by F positions.

Step 6: Add G and next sub-key

In this step (which is identical to step 3), G is added to the next sub-key) which is S[3] for the first round, and S[2i + 1] in general, for any round, where i starts with 1. The output of this process is H.

Step 7: Miscellaneous tasks

In this step, we check to see if all the rounds are over or no. For this, perform the following steps:

- Increment i by 1
- Check to see if $i < r$.

Assuming that i is still less than r, we rename F as C and H as D, and return back to step 1.

Sub-key Creation

Sub-key creation is a two-step process.

1. In the first step, the sub-keys (denoted by S[0],S[1],...) are generated.

The original key is called as L. In the second step, the sub-keys (S[0],S[1],...) are mixed with the corresponding sub-portions of the original key (*i.e.*, L[0],L[1],...)

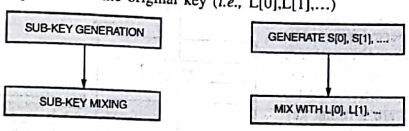


Fig. 4.36. Sub-key Generation Process

Step 1: Sub-key generation

In this step two constants P and Q are used. The array of sub-keys to be generated is called as S. The first sub-key S[0] is initialized with the value of P.

Each next sub-key (*i.e.*, S[1], S[2],...) is calculated on the basis of the previous sub-key and the constant value Q, using the addition mod 2^{32} operations. The process is done $2(r + 1) - 1$ times, where r is the number of rounds, as before. Thus, if we have 10 rounds, this process will be done $2(10 + 1) - 1$ times, *i.e.*, $2(11) - 1$ times, *i.e.*, 20 times. Thus, we will generate sub-keys S[0], S[1],...S[20].

Step 2 : Sub-key mixing

In the sub-key mixing stage, the sub-keys S[0],S[1],... are mixed with the sub-portions of the original key *i.e.*, L[0],L[1],...L[c]. Note that c is the last sub-key position in the original key.

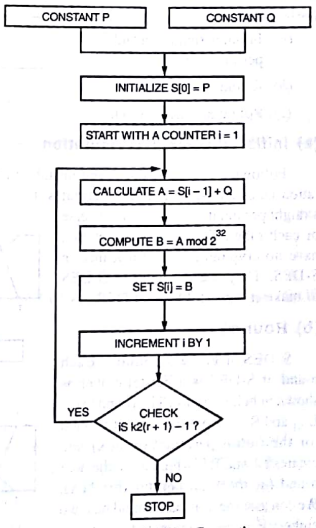


Fig. 4.37. Sub-Key Generation

4.8. SIMPLIFIED DES (S-DES)

Simplified DES(S-DES), developed by Professor Edward Schaefer of Santa Clara University, is an educational tool designed to help students learn the structure of DES.

4.8.1. Overview

S-DES encryption algorithm takes an 8-bit block of plaintext (example: 10111101) and a 10-bit key as input and produces an 8-bit block of ciphertext as output. The S-DES decryption algorithm takes an 8-bit block of ciphertext and the same 10-bit key used to produce that ciphertext as input and produces the original 8-bit block of plaintext. The general ideal of S-DES is shown in Fig. 4.38.

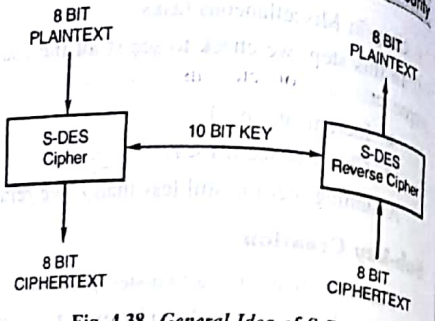


Fig. 4.38. *General Idea of S-DES*

4.8.2. Encryption Process of S-DES

The encryption process is made of two permutation (P-boxes), which we call initial and final permutation and 2 rounds. Each round uses a different 8 bit round key. There are 2 round key needed (K_1, K_2). Broadly we can divided S-DES process into three parts :

(*a*) Initial permutation and final permutation

(*b*) Rounds

(*c*) Round key generation.

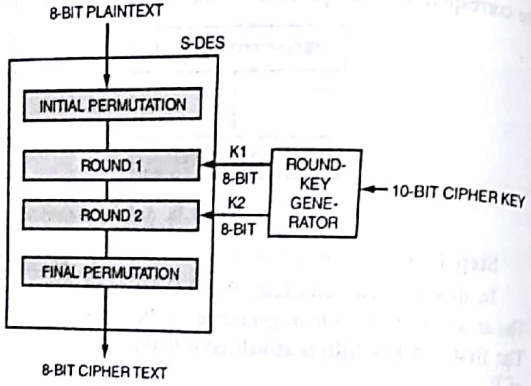


Fig. 4.39. *Encryption Process of S-DES*

(a) Initial and Final Permutation

Following Fig. 4.40 shows the initial and final permutation (P-Boxes). Each of these permutation takes an 8-bit input and permutes it according to a predefined rule. These permutation are straight permutation that are the inverse of each other. These two permutation have no cryptographic significance in S-DES. They are included in S-DES to make it compatible with DES.

(b) Rounds

S-DES uses two rounds. Each round of S-DES is a Feistel cipher as shown in below figure. The round takes L_{i-1} and R_{i-1} from the previous round (or the initial permutation box) and creates L1 and R1, which go to the next round (or the final permutation box). We can assume that each round has two cipher elements, a mixer and a swapper.

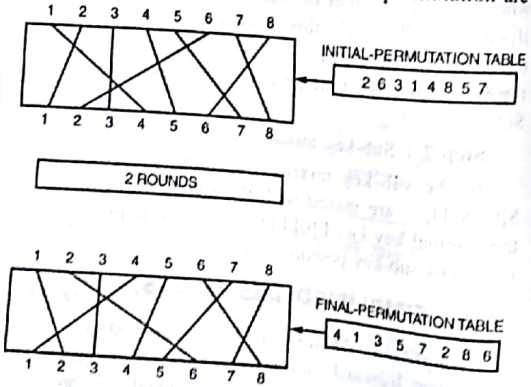


Fig. 4.40. *Initial and Final Permutations (IP and IP^{-1})*

Each of these elements is invertible. The swapper is obviously invertible. It swaps the left half of the text with the right half. The mixer is invertible because of the XOR operation. All non-invertible elements are collected inside the function, shown as $f(R_{i-1}, K_i)$.

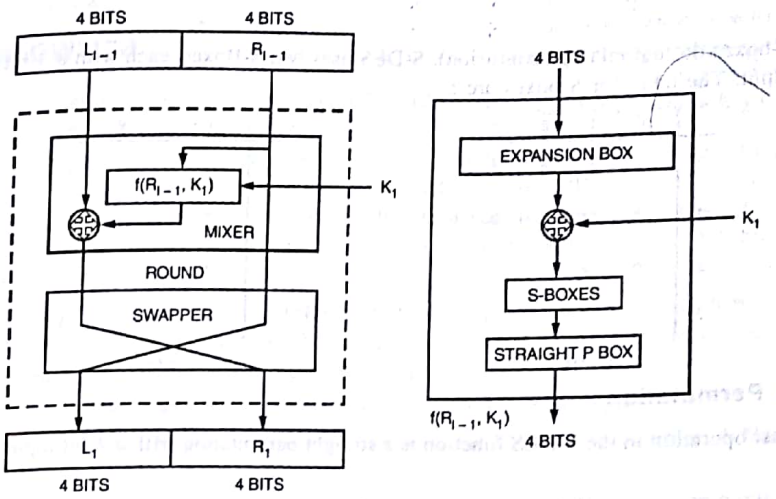


Fig. 4.41. *A Round of S-DES Encryption*

S-DES Function

The heart of S-DES function is the S-DES function. The S-DES function applies an 8-bit key to the rightmost 4 bit (R_{i-1}). To produce a 4-bit output. This function is made up of four sections:

- An expansion P-box
- A whitener (which adds key)
- A group of S-boxes
- A straight P-box

Expansion P-box

R_{i-1} is a 4-bit input and K_i is an 8-bit key, so we first need to expand R_{i-1} to 8 bits. S-DES uses a table to define the P-box as shown:

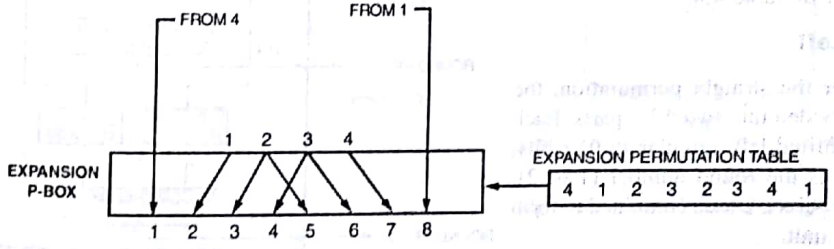


Fig. 4.42. *Expansion P-box*

Whitener (XOR)

After the expansion permutation, S-DES uses the XOR operation on the expanded right section and the round key.

S-Boxes

The S-boxes do real mixing (confusion). S-DES uses two S-Boxes, each with a 4-bit input and a 2-bit output. The table for S-boxes are :

	0	1	2	3
0	1	0	3	2
1	3	2	1	0
2	0	2	1	3
3	3	1	3	2

S-BOX 1

	0	1	2	3
0	0	1	2	3
1	2	0	1	3
2	3	0	1	0
3	2	1	0	3

S-BOX 2

Straight Permutation

The last operation in the S-DES function is a straight permutation with a 4-bit input and a 4-bit output.

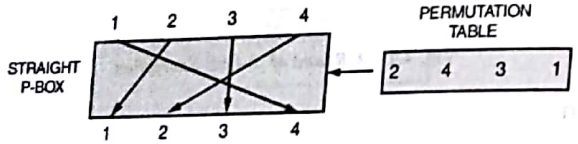


Fig. 4.43. Straight P-Box

(c) Round Key Generation

The round-key generator creates two 8-bit keys out of a 10-bit cipher key.

Straight Permutation

The first process is a straight permutation. It permutes the 10 bits in the key according to a predefined table, as shown in Table 4.8.

Shift Left

After the straight permutation, the key is divided into two 5-bit parts. Each part is shifted left (circular shift) r bits, where r is the round number (1 or 2). The two parts are then combined to form a 10-bit unit.

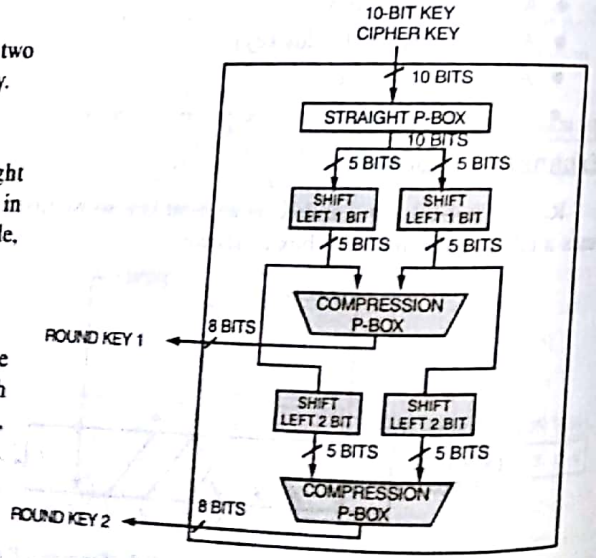


Fig. 4.44. Round Key Generator

Compression Permutation

The compression permutation (P-Box) changes the 10 bits to 8 bits, which are used as a key for a round as shown in Table 4.8.

TABLE FOR BTRAIGHT P-BOX

3	5	2	7	4	10	1	9	8	6

TABLE FOR COMPRESSION P-BOX

6	3	7	4	8	5	10	9

Table 4.8. Tables Used in Key Generation

4.9. BLOWFISH

Blowfish is an encrypted method designed in 1993 by Bruce Scheier as a fast, alternative to existing encryption algorithms such AES, DES and 3 DES etc. It includes in a large number of cipher suites and encryption products. IT is free algorithm available to everyone. Blowfish algorithm gained lots of popularity especially due to its free license. It uses a variable length key, from 32 to 448-bit. Although most of commercial and non-commercial products uses for the strongest 448-bit encryption with blowfish.

Blowfish is a symmetric block encryption algorithm designed in consideration with,

- **Fast:** It encrypts data on large 32-bit microprocessors at a rate of 26 clock cycles per byte.
- **Compact:** It can run in less than 5 K of memory.
- **Simple:** It uses addition, XOR, lookup table with 32-bit operands.
- **Secure:** The key length is variable, it can be in the range of 32-448 bits: default 128 bits key length.
- It is suitable for applications where the key does not change often, like communication link or an automatic file encryptor.
- Unpatented and royalty-free.

4.9.1. Blowfish Encryption

Blowfish symmetric block cipher algorithm encrypts block data of 64-bits at a time and creates an 64-bit output cipher text. It will follows the feistel network and this algorithm is divided into two parts.

1. Key-expansion
2. Data Encryption

1. Key-expansion

It will converts a key of at most 448 bits into several sub-key arrays totaling 4168 bytes. Blowfish uses large number of sub-keys. These keys are generate earlier to any data encryption or decryption. The p-array consists of 18, 32-bit subkeys:

$P_1, P_2, \ldots, P_{18}$

Four 32-bit S-Boxes consists of 256 entries each:

$S_{1,0}, S_{1,1}, \ldots S_{1,255}$
$S_{2,0}, S_{2,1}, \ldots, S_{2,255}$
$S_{3,0}, S_{3,1}, \ldots, S_{3,255}$
$S_{4,0}, S_{4,1}, \ldots, S_{4,255}$

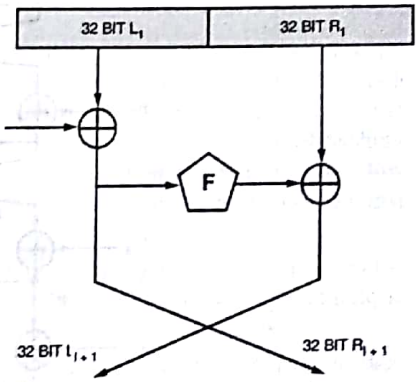


Fig. 4.45. General Idea of Blowfish

Generating the Subkeys: The subkeys are calculated using the Blowfish algorithm:

1. Initialize first the P-array and then the four S-boxes, in order, with a fixed string. This string consists of the hexadecimal digits of pi (less the initial 3): P_1 = 0x243f6a88, P_2 = 0x85a308d3, P_3 = 0x13198a2e, P_4 = 0x03707344, etc.
2. XOR P_1 with the first 32 bits of the key, XOR P_2 with the second 32-bits of the key, and so on for all bits of the key (possibly up to P_{14}). Repeatedly cycle through the key bits until the entire P-array has been XORed with key bits. (For every short key, there is at least one equivalent longer key; for example, if A is a 64-bit key, then AA, AAA, etc., are equivalent keys.)
3. Encrypt the all-zero string with the Blowfish algorithm, using the subkeys described in steps (1) and (2).
4. Replace P_1 and P_2 with the output of step (3).
5. Encrypt the output of step (3) using the Blowfish algorithm with the modified subkeys.
6. Replace P_3 and P_4 with the output of step (5).
7. Continue the process, replacing all entries of the P array, and then all four S-boxes in order, with the output of the continuously changing Blowfish algorithm.

In total, 521 iterations are required to generate all required subkeys. Applications can store the subkeys rather than execute this derivation process multiple times.

Encryption Algorithm

It is having a function to iterate 16 times of network. Each round consists of key-dependent permutation and a key and data-dependent substitution. All operations are XORs and additions on 32-bit words. The only additional operations are four indexed array data lookup tables for each round, The complex function F is shown in Fig. 4.45.

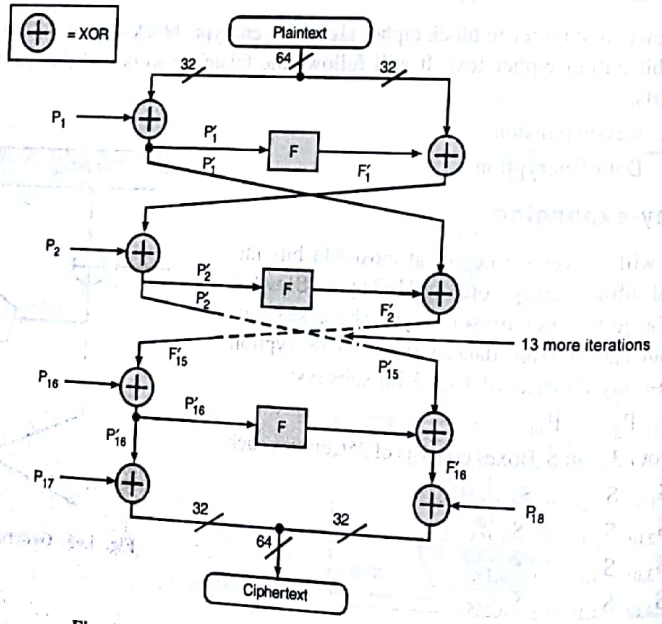


Fig. 4.46. *Blowfish Encryption Process*

Algorithm: Blowfish Encryption

```
Divide x into two 32-bit halves: xL, xR
  For i = 1to 16:
      xL = XL XOR Pi
      xR = F(XL) XOR xR
      Swap XL and xR
      Swap XL and xR (Undo the last swap.)
      xR = xR XOR P17
      xL = xL XOR P18
  Recombine xL and xR
```

4.9.2. Blowfish Decryption

Decryption for Blowfish is relatively straight forward. Ironically, decryption works in the same algorithmic direction as encryption beginning with the ciphertext as input. However, as expected, the sub-keys are used in reverse order. So the decryption Blowfish algorithm is as follows:

```
j = 1
loop from j to 16
   Rj = Lj-1 XOR P19-j
   Lj = F(Rj)XOR Rj-1
end loop
L17 = R16 XOR P1
R17 = L16 XOR P2
```

4.9.3. Benefits of Blowfish

Blowfish has been known to be very fast and compact only requiring 5K of RAM. Variability exists in key length and Blowfish is relatively simple to implement. Blowfish provides a litter stronger cryptographic process by performing operations on both halves of its input word per round which is different than the classical Feistal process. Finally, Blowfish provides a very strong avalanche affect in that every left-side input bit affects every right-side input bit per round.

4.9.4. Blowfish in Practice

Blowfish is one of the fastest block ciphers in widespread use, except when changing keys. Each new key requires pre-processing equivalent to encrypting about 4 kilobytes of text, which is very slow compared to other block ciphers. This prevents its use in certain applications, but is not a problem in others. In one application, it is actually a benefit: the password-hashing method used in OpenBSD uses an algorithm derived from Blowfish that makes use of the slow key schedule; the idea is that the extra computational effort required gives protection against dictionary attacks.

In some implementations, Blowfish has a relatively large memory footprint of just over 4 kilobytes of RAM. This is not a problem even for older smaller desktop and laptop computers, but it does prevent use in the smallest embedded systems such as early smartcards.

Blowfish is not subject to any patents and is therefore freely available for anyone to use. This benefit has contributed to its popularity in cryptographic software.

4.10. CAST-128 ALGORITHM

CAST-128, also known as CAST5, is a block cipher used in a number of products, notably as the default cipher in some versions of GNU Privacy Guard (GPG) and Pretty Good Privacy (PGP) systems. It has also been approved for Canadian government use by the Communications Security Establishment. CAST-128 was created in 1996 by Carlisle Adams and Stafford Tavares. Howard Heys and Michael Wiener also contributed to the design.

4.10.1. Description of Algorithm

CAST-128 is a design procedure for symmetric encryption algorithm developed by Carlisle Adams and Stafford Tavares. It belongs to the class of encryption algorithms known as Feistel ciphers and its overall operation is thus similar to DES. CAST-128 is a 12 or 16 round Feistel network with a 64 bit block size and the key size varies from 40 bits to 128 bits in 8-bit increments.

Algorithm employs two subkeys in each round Kmi (32 bit masking sub-key) and Kri (5-bit rotate sub-key). The function F depends on the rounds and has a structure of classical Fiestel network with 16 rounds of operation. Algorithm uses four primitive operations Addition (+) and subtraction (–) using modulo 2^{32}, Bitwise-XOR (^) and left circular rotation (<<<).

Encryption

CAST-128 is a fiestel network consisting of 16 rounds. The input is a 64-bit data element which is further split into two halves of 32-bit each namely L_0 and R_0.

```
L0 || R0 = Plaintext
For  i=1 to 16 do
Li = Ri-1;
Ri = Li-1 ^F(Ri-1,Kmi,Kri);
Ciphertext =R16 || L16
```

The function F includes the use of four 8*32 S boxes, Three round functions are used in CAST-128. The rounds are as follows (where "D" is the data input to the f function and "Ia"-"Id" are the most significant byte through least significant byte if I respectively and I is used to refer the intermediate 32-bit value after the left circular rotation function.

```
Rounds 1,4,7,10,13,16
I = ((Kmi + Ri-1) <<< Kri)
F = ((S1[Ia] ^ S2[Ib]) - (S3[Ic]) ) + S4[Id]

Rounds 2,5,8,11,14
I = ((Kmi ^ Ri-1) <<< Kri)
F= ((S1[Ia] - S2[Ib]) + (S3[Ic])) ^ S4[Id]

Rounds 3,6,9,12,15
I = ((Kmi - Ri-1) <<< Kri)
F= ((S1[Ia] + S2[Ib]) ^ (S3[Ic])) - S4[Id]
```

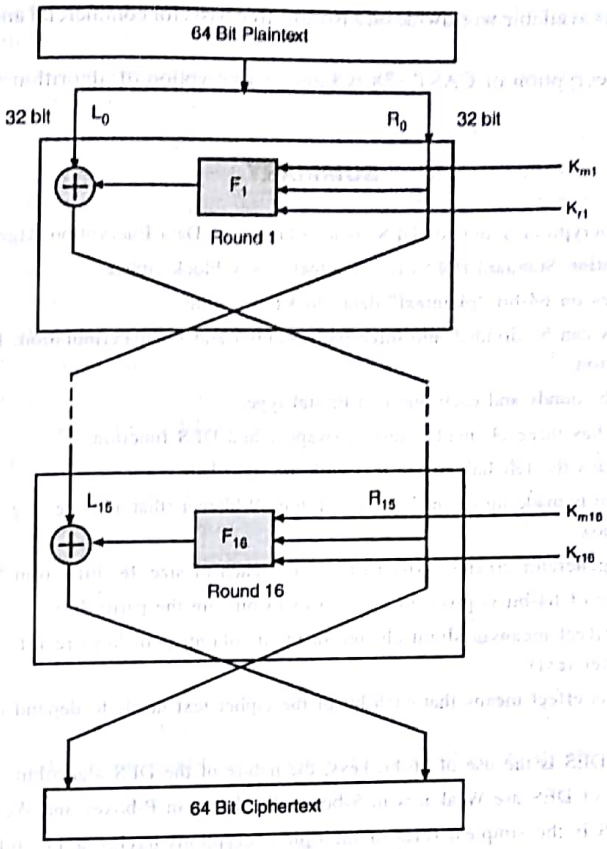



Fig. 4.47. *Cast-128 Encryption Process*

Substitution Boxes: There are Eight 32-bit S-boxes having 256 entries in each. In which four S-boxes from S[0] to S[4] are used in encryption and decryption process while other S-boxes means S[5] through S[8] are used in subkey generation.

Key Schedule : Let the 128-bit key be x0x1x2x3x4x5x6x7x8x9xAxBxCxDxExF, where x0 represents the most significant byte and xF represents the least significant byte. Let z0..zF be intermediate (temporary) bytes. Let Si[] represent S-box i and let "^" represent XOR addition.

- 16-round Feistel network with a 64-bit block size and a key size of between 40 to 128 bits (but only in 8-bit increments).
- The full 16 rounds are used when the key size is longer than 80 bits.
- Components include large 8x32-bit S-boxes based on bent functions, key-dependent rotations, modular addition and subtraction, and XOR operations.
- There are three alternating types of round function, but they are similar in structure and differ only in the choice of the exact operation (addition, subtraction or XOR) at various points.

CHAPTER 6

Asymmetric Key Cryptography

6.1. INTRODUCTION

Inside This Chapter

It is called Public key cryptography also. Secure data transmission coding scheme uses two different digital keys: one for encryption of data, the other for its decryption. In contrast as we read, symmetric key cryptography (SKC) schemes use only one key for both operations. For optimum data security, both types of cryptography are combined in an arrangement where AKC is used for authentication (of the parties and transmission of the digital keys) and SKC is used for bulk of the transmission (because of its comparatively low resource requirements).

Public-key encryption makes key-management much easier. It was invented in 1976 by two Stanford mathematicians, Whitfield Diffie and Martin Hellman. Their discovery can be phrased simply: enciphering schemes should be asymmetric. For thousands of years all ciphers were symmetric—the key for encrypting a message was identical to the key for decrypting it, but used, so to speak, in reverse. To change "5 100 100 5 15 55" or "6 120 120 6 18 66" back into "attack," for instance, one simply reverses the encryption by dividing the numbers with the key, instead of multiplying them, and then replaces the numbers with their equivalent letters. Thus sender and receiver must both have the key, and must both keep it secret. The symmetry, Diffie and Hellman realized, is the origin of the key-management problem. The solution is to have an encrypting key that is different from the decrypting key—one key to encipher a message, and another, different key to decipher it. With an asymmetric cipher, Alice could send encrypted messages to Bob without providing him with a secret key. In fact, Alice could send him a secret message even if she had never before communicated with him in any way.

Asymmetric key Encryption uses a secret key that must be kept from unauthorized users and a public key that can be made public to anyone. Both the public key and the private key are

mathematically linked; data encrypted with the public key can be decrypted only by the private key, and data signed with the private key can only be verified with the public key.

- Asymmetric encryption is a form of cryptosystem in which encryption and decryption are performed using the different keys—one a public key and one a private key. It is also known as public-key encryption. Here public and private key of receiver is used.
- Asymmetric encryption transforms plaintext into cipher text using a one of two keys and an encryption algorithm. Using the paired key and a decryption algorithm, the plain text is recovered from the cipher text.
- Asymmetric encryption can be used for confidentiality, authentication or both.
- The most widely used public- key cryptosystem is **RSA**. The difficulty of attacking **RSA** is based on the difficulty of finding the prime factors of a composite number.

The difference between public *(asymmetric)* and symmetric key cryptography basically has to do with how the keys are used.

With symmetric key cryptography, a single secret key is used both to encrypt and to decrypt the data. Thus, both parties to the communication must have access to the same secret key value. The distribution and management of such secret keys can be problematic, and has probably been a source of work for spies throughout history.

With asymmetric key cryptography, there are two keys. One is used to encrypt the data and the other is used to decrypt the data. Thus, one of the keys *(the encryption key)* can be publicly known without compromising secrecy so long as the other key *(the decryption key)* is held secret. This leads to the common name of *public key cryptography.*

Many of the key-management and distribution problems associated with symmetric key cryptography are alleviated through the use of public key cryptography.

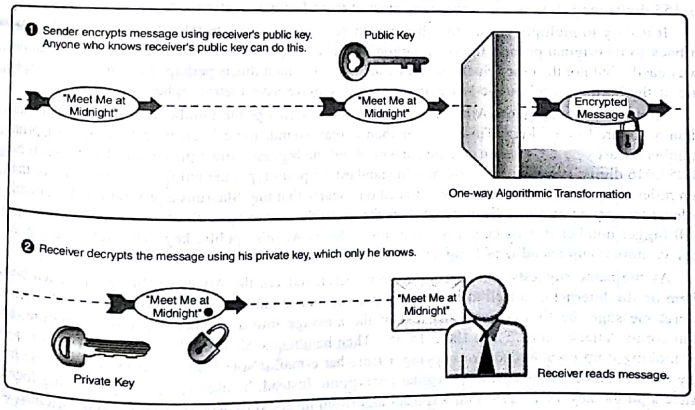


Fig. 6.1. *Idea of Symmetric Key Cryptography*

The Asymmetric Encryption system uses both the private and public keys. The private key is for yourself and the public key is published on line for others to see.

They use the public key to access the encryption code that corresponds to your private key. So, as shown in Fig. 6.1 if you are sending an encrypted message to Susan which you do not want others to see, you would use her public key to encrypt it. She will be able to decrypt it with her own corresponding private key. Likewise, if she sends a message to you, she uses your public key to encrypt the message and you would use your private key to decrypt it.

NOTE

In public key cryptography, public and private key of receiver is used.

6.2. RSA

6.2.1. Description

To be precise, Diffie and Hellman demonstrated only that public-key encryption was possible in theory. Another year passed before three MIT mathematicians—Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman—figured out a way to do it in the real world. At the base of the Rivest-Shamir-Adleman, or RSA, encryption scheme is the mathematical task of factoring. Factoring a number means identifying the prime numbers which, when multiplied together, produce that number. Thus 126,356 can be factored into $2 \times 2 \times 31 \times 1{,}019$, where 2, 31, and 1,019 are all prime. (A given number has only one set of prime factors.) Surprisingly, mathematicians regard factoring numbers—part of the elementary-school curriculum—as a fantastically difficult task. For big numbers the process is horribly time-consuming, even with fast computers. The largest number yet factored is 155 digits long. It took 292 computers, most of them fast workstations, more than seven months.

It is easy to multiply primes together. But there is no easy way to take the product and reduce it back to its original primes. In crypto jargon, this is a "trapdoor"—a function that lets you go one way easily, but not the other. Such one-way functions, of which this is perhaps the simplest example, are at the bottom of all public-key encryption. They make asymmetric ciphers possible.

To use RSA encryption, Alice first secretly chooses two prime numbers, p and q, each more than a hundred digits long. This is easier than it may sound: there is an infinite supply of prime numbers. Last year a Canadian college student found the biggest known prime: 213466917-1. It has 4,053,946 digits; typed without commas in standard 12-point type, the number would be more than ten miles long. Fortunately Alice doesn't need one nearly that big. She runs a program that randomly selects two prime numbers for her and then she multiplies them by each other, producing $p \times q$, a still bigger number that is, naturally, not prime. This is Alice's "public key." (In fact, creating the key is more complicated than I suggest here, but not wildly so.)

As the name suggests, public keys are not secret; indeed, the Alices of this world often post them on the Internet or attach them to the bottom of their e-mail. When Bob wants to send Alice a secret message, he first converts the text of the message into a number. Perhaps, as before, he transforms "attack" into "5 100 100 5 15 55." Then he obtains Alice's public key—that is, $p \times q$—by looking it up on a Web site or copying it from her e-mail. (Note here that Bob does not use his key to send Alice a message, as in regular encryption. Instead, he uses Alice's key.) Having found Alice's public key, he plugs it into a special algorithm invented by Rivest, Shamir, and Adleman to encrypt the message.

Bob knows the product $p \times q$, because Alice has displayed it on her Web site. But he almost certainly does not know p and q themselves, because they are its only factors, and factoring large numbers is effectively impossible. Yet the algorithm is constructed in such a way that to decipher the message the recipient must know both p and q individually. Because only Alice knows p and q, Bob can send secret messages to Alice without ever having to swap keys. Anyone else who wants to read the message will somehow have to factor $p \times q$. How hard is that? Even if a team of demented government agents spent a trillion dollars on custom computers that do nothing but try random numbers, the Sun would likely go nova before they succeeded. (Rivest, Shamir, and Adleman patented their algorithm and to market it created a company, RSA Data Security, in 1983.)

6.2.2. RSA Algorithm

The RSA scheme is a block cipher as shown in Fig. 6.2. Each plaintext block is an integer between 0 and $n - 1$ for some n, which leads to a block size $\leq \log_2(n)$. The typical size for n is 1024 bits. The details of the RSA algorithm are described as follows.

Key generation : Following steps are required to create public (e) and private key (d).

(1) Pick two large prime numbers p and q, $p \neq q$;
(2) Calculate $n = p \times q$;
(3) Calculate $\Phi(n) = (p - 1)(q - 1)$;
(4) Choose e, so that gcd $(e, \Phi(n)) = 1$ and $1 < e < \Phi(n)$, *i.e.*, e is not a factor of $\Phi(n)$.
(5) Calculate d, so that $d.e \bmod \Phi(n) = 1$, *i.e.*, d is the multiplicative inverse of e in mod $\Phi(n)$;
(6) Get public key as $KU = \{e, n\}$;
(7) Get private key as $KR = \{d, n\}$.

Encryption

For plaintext block $P < n$, its ciphertext will be calculated using equation

$$\boxed{C = P^e \bmod n}$$

where e is receiver's the public key.

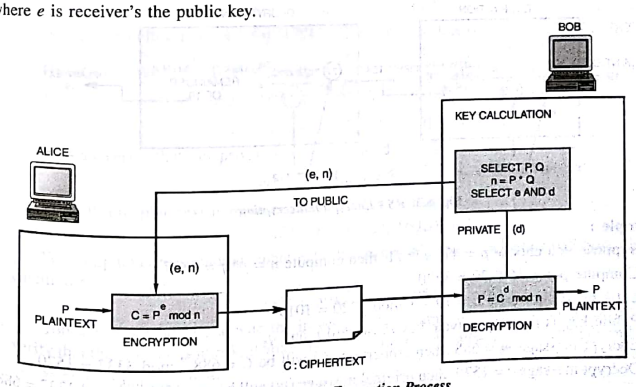


Fig. 6.2. *RSA Encryption Process*

Decryption

For ciphertext block C, its plaintext will be calculated using equation

$$P = C^d \bmod n$$

where d is receiver's private key.

6.2.3. Proof of RSA

The proof uses the *Euler-Fermat Theorem*. It works for messages m that are relatively prime to the modulus n, that is where $\gcd(m, n) = 1$.

Proof. Suppose $\gcd(m, n) = 1$. The relation $ed \equiv 1 \pmod{\varphi(n)}$ gives that $ed = 1 + k\varphi(n)$ for some integer k. If $c \equiv m^e \pmod n$ then, working modulo n,

$$\begin{aligned} c^d &\equiv m^{ed} \\ &\equiv m^{1 + k\varphi(n)} \\ &\equiv m.(m^{\varphi(n)})^k \\ &\equiv m.1^k, \text{ since } m^{\varphi(n)} \equiv 1 \pmod n, \text{ by the Euler-Fermat theorem, as } \gcd(m, n) = 1 \\ &\equiv m \pmod n. \end{aligned}$$

Hence $m = c^d \bmod n$ is a unique integer in the range $0 = m < n$.

Example :

- Randomly select two primes $p = 7$, $q = 17$
- Calculate $n = p*q = 7 \times 17 = 119$
- Calculate $\Phi(n) = (p - 1)(q - 1) = 96$
- Randomly select $e = 5$, since $\gcd(e, \Phi(n)) = 1$, *i.e.*, e should not be factor of 96
- Calculate d, $d*5 \bmod \Phi(n) = 1$, so d will be 77
- Public key $KU = \{5,119\}$
- Private key $KR = \{77,119\}$

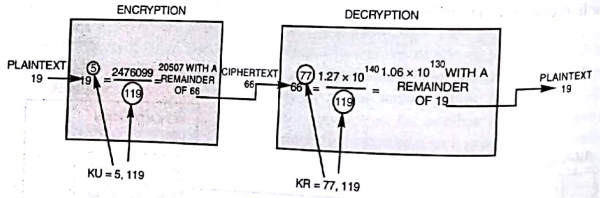


Fig. 6.3. *RSA Encryption/decryption*

Example :

- Suppose you choose $p = 47$, $q = 71$, then compute $n = p*q = 3337$
- Compute phi = 46 * 70 = 3220
- Let e be 79, compute $d = 79 - 1 \bmod 3220 = 1019$
- Public key is n and e, private key d, discard p and q.
- Encrypt message $m = 688$, then ciphertext (C) will be $C = 688^{79} \bmod 3337 = 1570$.
- Decrypt message $c = 1570$, then original message (m) will be $m = 1570^{1019} \bmod 3337 = 688$.

Example : Now think that you are sending a message then how ciphertext will be found.

This time, to make life slightly less easy for those who can crack simple Caesar substitution codes, we will group the characters into blocks of three and compute a message representative integer for each block. Suppose message is ATTACK AT SEVEN.

Now put x for each blank and break into three letters blocks.

ATTACKXATXSEVEN = ATT ACK XAT XSE VEN

In the same way that a decimal number can be represented as the sum of powers of ten, *e.g.*, $135 = 1 \times 10^2 + 3 \times 10^1 + 5$, we could represent our blocks of three characters in base 26 using $A = 0$, $B = 1$, $C = 2$, ..., $Z = 25$

$$\begin{aligned} \text{ATT} &= 0 \times 26^2 + 19 \times 26^1 + 19 = 513 \\ \text{ACK} &= 0 \times 26^2 + 2 \times 26^1 + 10 = 62 \\ \text{XAT} &= 23 \times 26^2 + 0 \times 26^1 + 19 = 15567 \\ \text{XSE} &= 23 \times 26^2 + 18 \times 26^1 + 4 = 16020 \\ \text{VEN} &= 21 \times 26^2 + 4 \times 26^1 + 13 = 14313 \end{aligned}$$

For this example, to keep things simple, we'll not worry about numbers and punctuation characters, or what happens with groups *AAA* or *AAB*.

In this system of encoding, the maximum value of a group (*ZZZ*) would be $26^3 - 1 = 17575$, so we require a modulus n greater than this value.

1. We "generate" primes $p = 137$ and $q = 131$ (we cheat by looking for suitable primes around v_n)
2. $n = p*q = 137 \times 131 = 17947$
   phi $= (p - 1)(q - 1) = 136 \times 130 = 17680$
3. Select $e = 3$
   check $\gcd(e, p - 1) = \gcd(3, 136) = 1$, OK and
   check $\gcd(e, q - 1) = \gcd(3, 130) = 1$, OK.
4. Compute $d = e^{-1} \bmod \text{phi} = 3^{-1} \bmod 17680 = 11787$.
5. Hence public key, $(n, e) = (17947, 3)$ and private key $(n, d) = (17947, 11787)$.

Question : Why couldn't we use $e = 17$ here?

To encrypt the first integer that represents "ATT", we have

$$c = m^e \bmod n = 513^3 \bmod 17947 = 8363.$$

We can verify that our private key is valid by decrypting

$$m' = c^d \bmod n = 8363^{11787} \bmod 17947 = 513.$$

Overall, our plaintext is represented by the set of integers m

(513, 62, 15567, 16020, 14313)

We compute corresponding ciphertext integers $c = m^e \bmod n$, (which is still possible by using a calculator)

(8363, 5017, 11884, 9546, 13366)

You are welcome to compute the inverse of these ciphertext integers using $m = c^d \bmod n$ to verify that the RSA algorithm still holds. However, this is now outside the realms of hand calculations unless you are very patient.

6.2.4. Notes on Practical Applications

1. To generate the primes p and q, generate a random number of bit length $b/2$ where b is the required bit length of n; set the low bit (this ensures the number is odd) and set the two highest bits (this ensures that the high bit of n is also set); check if prime (use the *Rabin-Miller* test); if not, increment the number by two and check again until you find a prime. This is p. Repeat for q starting with a random integer of length $b - b/2$. In the extremely unlikely event that $p = q$, check your random number generator. Alternatively, instead of incrementing by 2, just generate another random number each time.

   There are stricter rules in ANSI X9.31 to produce *strong primes* and other restrictions on p and q to minimise the possibility of known techniques being used against the algorithm. It is probably better just to use a longer key length.

2. In practice, common choices for e are 3, 17 and 65537 ($2^{16} + 1$). These are Fermat primes, sometimes referred to as F_0, F_2 and F_4 respectively ($Fx = 2\text{^}(2\text{^}x)+1$). They are chosen because they make the modular exponentiation operation faster. Also, having chosen e, it is simpler to test whether $\gcd(e, p - 1) = 1$ and $\gcd(e, q - 1) = 1$ while generating and testing the primes in step 1. Values of p or q that fail this test can be rejected there and then. (Even better: if e is prime and greater than 2 then you can do the less-expensive test ($p \bmod e$)!=1 instead of $\gcd(p - 1, e)==1$.)

3. To compute the value for d, use the *Extended Euclidean Algorithm* to calculate $d = e^{-1} \bmod \text{phi}$, also written $d = (1/e) \bmod \text{phi}$. This is known as *modular inversion*. Note that this is not integer division. The modular inverse d is defined as the integer value such that $ed = 1 \bmod \text{phi}$. It only exists if e and *phi* have no common factors.

4. When representing the plaintext octets as the representative integer m, it is usual to add random padding characters to make the size of the integer m large and less susceptible to certain types of attack. If $m = 0$ or 1 or $n - 1$ there is no security as the ciphertext has the same value. For more details on how to represent the plaintext octets as a suitable representative integer m, it is important to make sure that $m < n$ otherwise the algorithm will fail. This is usually done by making sure that the first octet of m is equal to 0x00.

5. Decryption and signing are identical as far as the mathematics is concerned as both use the private key. Similarly, encryption and verification both use the same mathematical operation with the public key. That is, mathematically, for $m < n$,

$$m = (m^e \bmod n)^d \bmod n = (m^d \bmod n)^e \bmod n$$

However, note these important differences in implementation :

- The signature is derived from a message digest of the original information. The recipient will need to follow exactly the same process to derive the message digest, using an identical set of data.
- The recommended methods for deriving the representative integers are different for encryption and signing (encryption involves random padding, but signing uses the same padding each time).

6.2.5. Security of RSA

A few considerations are critical for the security of RSA:

- Generating primes, which an enemy cannot guess, requires a strong random number generator; any weakness in the generator reduces security.

- Using the system securely requires that the private key never be revealed; losing that key to an enemy instantly reduces security to zero.

Discovery of an efficient solution to the integer factorization problem would break RSA.

6.2.6. Attacks on RSA

1. RSA and factoring

Given an efficient solution to the integer factorization problem, breaking RSA would become trivial. The attacker is assumed to have the public key (N, e). If he can factor N, he gets p, q and, therefore, $p - 1$, $q - 1$, and T. He knows e and can calculate its inverse mod T using the efficient Extended Euclidean algorithm. That gives him d and he already has N, so now he knows the private key (N, d). With the private key he can both read encrypted messages and forge the digital signature of the key owner. The cryptosystem would be rendered worthless.

The problem with that is that no really efficient (polynomial in the number of bits in N) solution for factoring is known, despite considerable effort by quite a few people over several decades to find one. It seems possible that no such algorithm exists, though no-one has proven that.

A number of methods have been proposed for attacking the RSA cryptosystem. These types of attacks are following.

Weiner attack

Michael Weiner proposed an attack based on continued fractions which is effective if the exponent in the secret key is small. There have since been many papers proposing improvements on or variants of that attack.

TWIRL

The Weizmann Institute Relation Locator, developed by Adi Shamir (The 'S' in RSA) and Eran Tromer, is a machine designed to speed up the seiving step in the number field seive technique for integer factorization.

2. Chosen-Ciphertext Attack

A potential attack on RSA is based on the multiplicative property of RSA. Alice creates the ciphertext $C = P^e \bmod n$ and sends C to Bob. Bob will decrypt an arbitrary ciphertext for Eve, other than C. Eve intercepts C and uses the following steps to find P :

(*a*) Eve chooses a random integer X in Zn^*.

(*b*) Eve calculates $Y = C^*X^e \bmod n$.

(*c*) Eve sends Y to Bob for decryption and get $Z = Y^d \bmod n$; This step is an instance of a chosen-ciphertext attack.

(*d*) Eve can easily find P because

$$Z = Y^d \bmod n = (C^*X^e)^d \bmod n = (C^d * X^{ed}) \bmod n = (C^d * X) \bmod n = (P^*X) \bmod n$$

$$Z = (P^*X) \bmod n \qquad P = Z^*X^{-1} \bmod n$$

3. Attack on Encryption Exponent

To reduce the encryption time, it is tempting to use a small encryption exponent e. The common value for e is $e = 3$. There are some attacks on low encryption exponent. These attacks do not generally result in a breakdown of the system. These types of attacks are following.

Coppersmith Theorem Attack

The major low encryption exponent attack is referred to as the Coppersmith theorem attack. This theorem state that in a modulo-n polynomial $f(x)$ of degree e, one can use an algorithm of the complexity log n to find the roots if one of the root is smaller than $n^{1/e}$.

Broadcast Attack

The broadcast attack can be launched if one entity sends the same message to a group of recipients with the same low encryption exponent.

Related Message Attack

The related message attack, discovered by Franklin Reiter. Alice encrypts two plaintext, P1 and P2, and encrypts them with $e = 3$ and sends C1 and C2 to Bob.

Short Pad Attack

The short pad attack, discovered by Coppersmith. Alice has a message M to send to Bob. She pad the message with $r1$, encrypts the result to get C1, and sends C1 to Bob.

4. Attack on the Decryption Exponent

Two forms of attack can be launched on the decryption exponents: revealed decryption exponent attack and low decryption exponent attack. These types of attacks are following.

Revealed Decryption Exponent Attack

It is obvious that if Eve can find the decryption exponent, d, she can decrypt the current encrypted message. However, the attack does not stop here. If Eve knows the value of d, she can use a probabilistic algorithm to factor n and find the value of p and q.

Low Decryption Exponent Attack

Bob may think that using a small private key d, would make the decryption process faster for him. Wiener showed that if $d < 1/3n^{1/4}$, a special type of attack based on continuous fraction.

5. Plaintext

Plaintext and Ciphertext in RSA are permutations of each other because they are integers in the same interval (0 to $n - 1$). Three attacks have been mentioned in the literature:

Short Message Attack

In the short message attack, if Eve knows the set of possible plaintexts, she then knows one more price of information in addition to the fact that the ciphertext is the permutation of plaintext.

Cycling Attack

The cycling attack based on the fact that if the ciphertext is a permutation of the plaintext, the continuous encryption of the ciphertext will eventually result in the plaintext. In other words, Eve continuously encrypts the intercepted ciphertext; she will eventually get the plaintext.

Unconcealed Message Attack

Another attack is based on permutation relationship between plaintext and ciphertext is the unconcealed message attack. An unconcealed message is a message that encrypts to itself. It has been proven that there are always some messages that are encrypted to themselves.

6. Attack on Modulus

Common Modulus Attack

The common modulus attack can be launched if a community uses a common modulus n. For example, people in a community might let a trusted party select p and q, calculate n and $\Phi(n)$, and create a pair of exponents (e_i, d_i) for each entity.

Example : Let Alice use n, e_a, Bob, n, e_b. Then if gcd $(e_a, e_b) = 1$, Eve can decrypt message as follows :

Eve uses Euclidean Algorithm to compute r and s such that $e_a r + e_b s = 1$. Assume r is negative (one of r and s must be, so just switch the two if r is positive),

then $(M^{-e}_a)\,(M^{-e}_b)s \equiv M \pmod n$

7. Attack on Implementation

Timing Attack

Paul Kocher elegantly demonstrated a ciphertext—only attack, called the timing attack. The attack is based on the fast algorithm. The algorithm uses only squaring if the corresponding bit in the private exponent d is 0; it uses both squaring and multiplication if the corresponding bit is. There are two methods to thwart timing attack:

1. Add random delays to the exponentiations to make each exponentiation take the same amount of time.
2. Rivest recommended blinding.

Power Attack

The power attack is similar to the timing attack. Kocher showed that if Eve can precisely measure the power consumed during decryption, she can launch a power attack based on the principle for timing attack.

6.3. ELLIPTIC CURVE CRYPTOGRAPHY (ECC)

Elliptic curves are not ellipses, instead, they are cubic curves of the form $y^3 = x^3 + ax + b$.

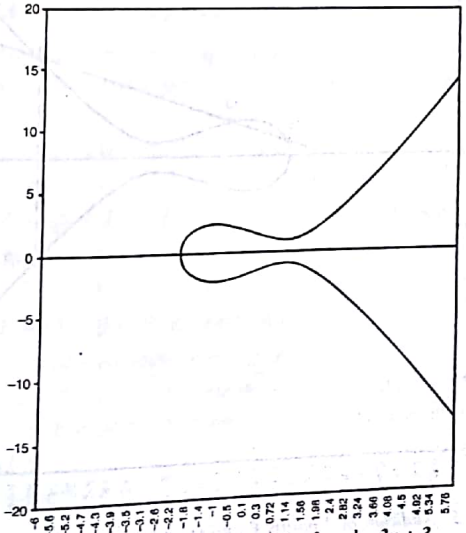


Fig. 6.4. *Elliptic curve over R^2: $y^2 = x^3 - 3x + 3$*

Elliptic curves over R^2 (R^2 is the set $R \times R$, where R = set of real numbers) is defined by the set of points (x, y) which satisfy the equation $y^3 = x^3 + ax + b$, along with a point O, which is the point at infinity and which is the additive identity element. The curve is represented as $E(R)$.

The following Fig. 6.4 is an elliptic curve satisfying the equation $y^2 = x^3 - 3x + 3$.

Elliptic Curves over Finite Fields

Elliptic Curves over F_p

An elliptic curve $E(F_p)$ over a finite field F_p is defined by the parameters $a, b \in F_p$ (a, b satisfy the relation $4a^3 + 27b^2 \neq 0$), consists of the set of points $(x, y) \in F_p$, satisfying the equation $y^2 = x^3 + ax + b$. The set of points on $E(F_p)$ also include point O, which is the point at infinity and which is the identity element under addition.

The Addition operator is defined over $E(F_p)$ and it can be seen that $E(F_p)$ forms an abelian group under addition.

The addition operation in $E(F_p)$ is specified as follows :

- $P + O = O + P = P$, $\forall P \in E(F_p)$
- If $P = (x, y) \in E(F_p)$, then $(x, y) + (x, -y) = O$. (The point $(x, -y) \in E(F_p)$ and is called the negative of P and is denoted $-P$)
- If $P = (x_1, y_1) \in E(F_p)$ and $Q = (x_2, y_2) \in E(F_p)$ and $P \neq Q$, then $R = P + Q = (x_3, y_3) \in E(F_p)$, where $x_3 = \lambda^2 - x_1 - x_2$, $y_3 = \lambda(x_1 - x_3) - y_1$, and $\lambda = (y_2 - y_1)/(x_2 - x_1)$, *i.e.*, the sum of 2 points can be visualized as the point of intersection $E(F_p)$ and the straight line passing through both the points.

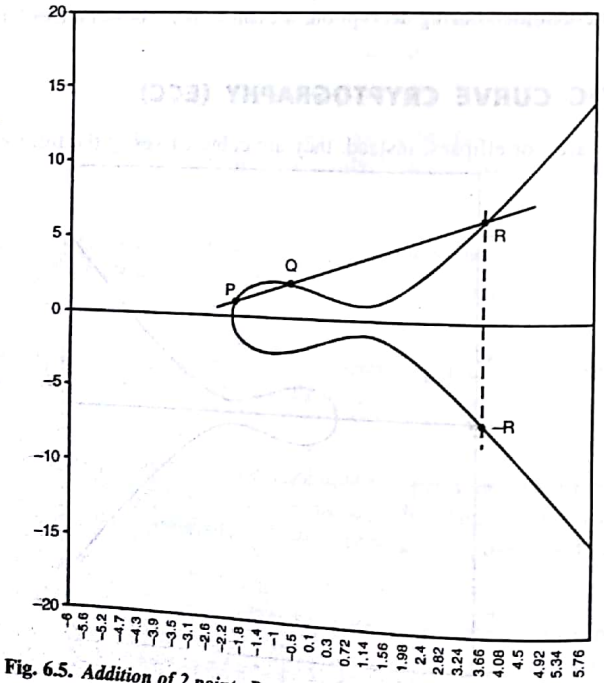


Fig. 6.5. *Addition of 2 points P and Q on the curve* $y^2 = x^3 - 3x + 3$

- Let $P = (x, y) \in E(F_p)$. Then the point $Q = P + P = 2P = (x_1, y_1) \in E(F_p)$, where $x_1 = \lambda^2 - 2x$, $y_1 = \lambda(x - x_1) - y$, where $\lambda = (3x^2 + a)/2y$. This operation is also called doubling of a point and can be visualized as the point of intersection of the elliptic curve and the tangent at P.

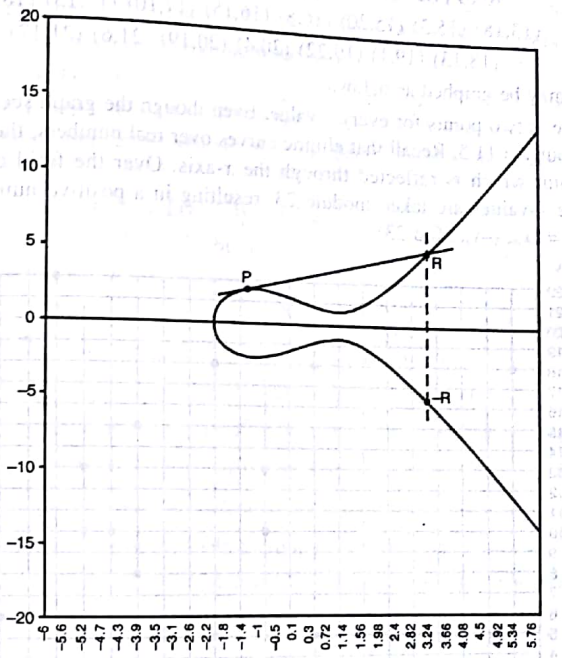


Fig. 6.6. *Doubling of a point P, R = 2P on the curve* $y^2 = x^3 - 3x + 3$

We can notice that addition over $E(F_p)$ requires one inversion, two multiplications, one squaring and six additions. Similarly, doubling a point on $E(F_p)$ requires one inversion, two multiplication, two squaring and eight additions.

Consider the set $E(F_p)$ over addition. We can see that

- $\forall P, Q \in E(F_p)$, if $R = P + Q$, then $R \in E(F_p)$ (Closure)
- $P + (Q + R) = (P + Q) + R$, $\forall P, Q, R \in E(F_p)$ (Associative)
- $\exists O \in E(F_p)$, such that $\forall P \in E(F_p)$, $P + O = O + P = P$ (Identity element)
- $\forall P \in E(F_p)$, $\exists -P \in E(F_p)$ such that, $P + (-P) = (-P) + P = O$. (Inverse element)
- $\forall P, Q \in E(F_p)$, $P + Q = Q + P$. (Commutative)

Thus we see that $E(F_p)$ forms an abelian group under addition.

Example : Consider an elliptic curve over the field Z_{23}. With $a = 1$ and $b = 0$, the elliptic curve equation is $y^2 = x^3 + x$. The point (9,5) satisfies this equation since:

$y^2 \bmod p = x^3 + x \bmod p$

$25 \bmod 23 = 729 + 9 \bmod 23$

$$25 \bmod 23 = 738 \bmod 23$$
$$2 = 2$$

The 23 points which satisfy this equation are:

(0,0) (1,5) (1,18) (9,5) (9,18) (11,10) (11,13) (13,5)
(13,18) (15,3) (15,20) (16,8) (16,15) (17,10) (17,13) (18,10)
(18,13) (19,1) (19,22) (20,4) (20,19) (21,6) (21,17)

These points may be graphed as below:

Note that there is two points for every x value. Even though the graph seems random, there is still symmetry about $y = 11.5$. Recall that elliptic curves over real numbers, there exists a negative point for each point which is reflected through the x-axis. Over the field of Z_{23}, the negative components in the y-values are taken modulo 23, resulting in a positive number as a difference from 23. Here $-P = (x_p, (-y_p \text{ Mod } 23)$

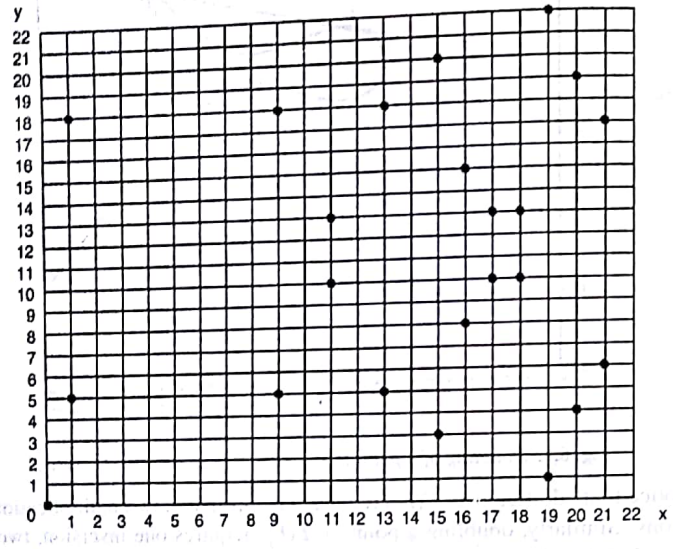


Fig. 6.7. *Elliptic Curve Equation : $y^2 = x_3 + x$ over F_{23}*

Elliptic Curves Over $F(2^m)$

An elliptic curve $E(F(2^m))$ over a finite field $F(2^m)$ is defined by the parameters $a, b \in F(2^m)$ (a, b satisfy the relation $4a^3 + 27b^2 \neq 0$, $b \neq 0$), consists of the set of points $(x, y) \in F(2^m)$, satisfying the equation $y^2 + xy = x^3 + ax + b$. The set of points on $E(F(2^m))$ also include point O, which is the point at infinity and which is the identity element under addition.

Similar to $E(F_p)$, addition is defined over $E(F(2^m))$ and we can similarly verify that even $E(F(2^m))$ forms an abelian group under addition.

The addition operation in $E(F(2^m))$ is specified as follows.

- $P + O = O + P = P, \forall P \in E(F(2^m))$

- If $P = (x, y) \in E(F(2^m))$, then $(x, y) + (x, -y) = O$. (The point $(x, -y) \in E(F(2^m))$ and is called the negative of P and is denoted $-P$)
- If $P = (x_1, y_1) \in E(F(2^m))$ and $Q = (x_2, y_2) \in E(F(2^m))$ and $P \neq Q$, then $R = P + Q = (x_3, y_3) \in E(F(2^m))$, where $x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$, $y_3 = \lambda (x_1 + x_3) + x_3 + y_1$, and $\lambda = (y_1 + y_2) / (x_1 + x_2)$, *i.e.*, the sum of 2 points can be visualized as the point of intersection $E(F(2^m))$ and the straight line passing through both the points.
- Let $P = (x, y) \in E(F(2^m))$. Then the point $Q = P + P = 2P = (x_1, y_1) \in E(F(2^m))$, where $x_1 = \lambda^2 + \lambda + a$, $y_1 = \lambda(x + x_1) + x_1 + y$, where $\lambda = x + (x/y)$. This operation is also called doubling of a point and can be visualized as the point of intersection of the elliptic curve and the tangent at P.

We can notice that addition over $E(F(2^m))$ requires one inversion, two multiplications, one squaring and eight additions. Similarly, doubling a point on $E(F(2^m))$ requires one inversion, two multiplication, one squaring and six additions.

Similar to $E(F_p)$, consider addition under $E(F(2^m))$,

- $\forall P, Q \in E(F(2^m))$, if $R = P + Q$, then $R \in E(F(2^m))$ (Closure)
- $P + (Q + R) = (P + Q) + R, \forall P, Q, R \in E(F(2^m))$ (Associative)
- $\exists O \in E(F(2^m))$, such that $\forall P \in E(F(2^m))$, $P + O = O + P = P$ (Identity element)
- $\forall P \in E(F(2^m))$, $\exists -P \in E(F(2^m))$, such that, $P + (-P) = (-P) + P = O$. (Inverse)
- $\forall P, Q \in E(F(2^m))$ $P + Q = Q + P$. (Commutative)

Thus we see that $E(F(2^m))$ forms an abelian group under addition.

Example : consider the field GF(2^4), defined by using polynomial representation with the irreducible polynomial $f(x) = x^4 + x + 1$.

The element $g = (0010)$ is a generator for the field. The powers of g are :

$g^0 = (0001)$ $g^1 = (0010)$ $g^2 = (0100)$ $g^3 = (1000)$ $g^4 = (0011)$ $g^5 = (0110)$
$g^6 = (1100)$ $g^7 = (1011)$ $g^8 = (0101)$ $g^9 = (1010)$ $g^{10} = (0111)$ $g^{11} = (1110)$
$g^{12} = (1111)$ $g^{13} = (1101)$ $g^{14} = (1001)$ $g^{15} = (0001)$

In a true cryptographic application, the parameter m must be large enough to preclude the efficient generation of such a table otherwise the cryptosystem can be broken. In today's practice, $m = 160$ is a suitable choice. The table allows the use of generator notation (g^e) rather than bit string notation, as used in the following example. Also, using generator notation allows multiplication without reference to the irreducible polynomial

$$f(x) = x^4 + x + 1.$$

Consider the elliptic curve $y^2 + xy = x^3 + g^4x^2 + 1$. Here $a = g^4$ and $b = g^0 = 1$. The point (g^5, g^3) satisfies this equation over $F(2^m)$:

$$y^2 + xy = x^3 + g^4x^2 + 1$$
$$(g^3)^2 + g^5g^3 = (g^5)^3 + g^4g^{10} + 1$$
$$g^6 + g^8 = g^{15} + g^{14} + 1$$
$$(1100) + (0101) = (0001) + (1001) + (0001)$$
$$(1001) = (1001)$$

The fifteen points which satisfy this equation are:
$(1, g^{13})$ (g^3, g^{13}) (g^5, g^{11}) (g^6, g^{14}) (g^9, g^{13}) (g^{10}, g^8) (g^{12}, g^{12})
$(1, g^6)$ (g^3, g^8) (g^5, g^3) (g^6, g^8) (g^9, g^{10}) (g^{10}, g) $(g^{12}, 0)$ $(0, 1)$
These points are graphed below:

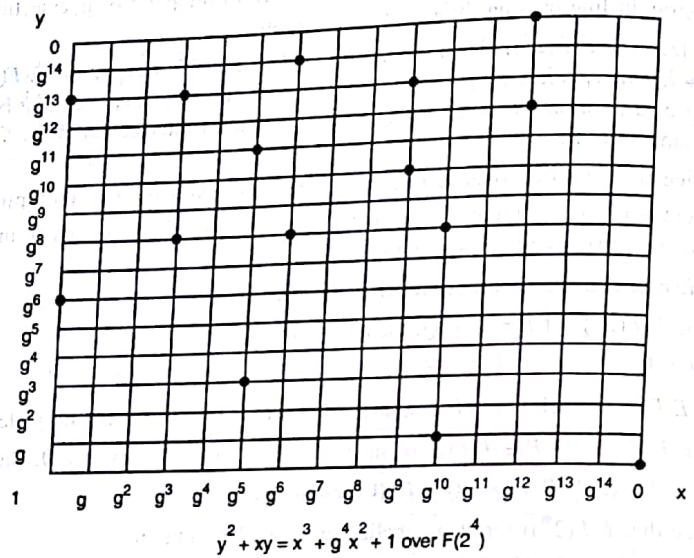


Fig. 6.8. *Curve Points Over F(2⁴)*

Example 6.1. *Determine order of point P = (0, 6) on Elliptic Curve Set $E_7(1, 1)$*

Solution. The values of $a = 1$, $b = 1$ and $p = 7$

The points lies on Elliptic Curve Set $E_7(1, 1)$ are

$$[(0, 1), (0, 6), (2, 2), (2, 5), O]$$

Base Point $P = (0, 6)$

1. For 2P = we need to determine P + P

$$\lambda = ((3 \times 0 + 1)/2 \times 6) \bmod 7 = (1/12) \bmod 7$$
$$= (1 \times 12^{-1}) \bmod 7 = (1 \times 3) \bmod 7 = 3$$
$$X_R = (\lambda^2 - xp - xq) \bmod p = (9 - 0 - 0) \bmod 7 = 2$$
$$Y_R = ((\lambda(xp - xq) - yp) \bmod p$$
$$= ((3(0 - 2) - 6) \bmod 7 = (-12) \bmod 7 = 2$$

So, $2P = (2, 2)$ which belongs to Elliptic Curve Set $E_7(1, 1)$

2. For 3P = we need to determine 2P + P

$$\lambda = (2/-4) \bmod 7 = (-1/2) \bmod 7 = (-1 \times 2^{-1}) \bmod 7$$
$$= (-1 \times 4) \bmod 7 = 3$$

$$X_R = (\lambda^2 - xp - xq) \bmod p = (9 - 0 - 2) \bmod 7 = 0$$
$$Y_R = ((\lambda(xp - xq) - yp) \bmod p = ((3(0 - 0) - 6) \bmod 7 = (-6) \bmod 7 = 1$$

So, $3P = (0, 1)$ which belongs to Elliptic Curve Set $E_7(1, 1)$

3. For 4P = we need to determine 3P + P

$$P = (0, 6)$$
$$\text{and } 3P = (0, 1) = (-P)$$
$$4P = P + 3P$$
$$4P = P + (-P) = 0$$

Since $4P = 0$, the order of a point $(0, 6)$ is 4.

Elliptic Curve: Some Definitions

- **Scalar Multiplication :** Given an integer k and a point P on the elliptic curve, the elliptic scalar multiplication kP is the result of adding Point P to itself k times.
- **Order :** Order of a point P on the elliptic curve is the smallest integer r such that $rP = O$. Further if c and d are integers, then $cP = dP$ iff $c \equiv d \pmod r$.
- **Curve Order :** The number of points on the elliptic curve is called its curve order and is denoted $\#E$.

Elliptical Curve Discrete Logarithm Problem

The strength of the Elliptic Curve Cryptography lies in the Elliptic Curve Discrete Log Problem (ECDLP). The statement of ECDLP is as follows.

Let E be an elliptic curve and $P \in E$ be a point of order n. Given a point $Q \in E$ with $Q = mP$, for a certain $m \in \{2, 3, \ldots\ldots, m - 2\}$.

Find the m for which the above equation holds.

When E and P are properly chosen, the ECDLP is thought to be infeasible. Note that $m = 0$, 1 and $m - 1$, Q takes the values O, P and $-P$. One of the conditions is that the order of P *i.e.*, n be large so that it is infeasible to check all the possibilities of m.

The difference between ECDLP and the Discrete Logarithm Problem (DLP) is that, DLP though a hard problem is known to have a sub exponential time solution, and the solution of the DLP can be computed faster than that to the ECDLP. This property of Elliptic curves makes it favourable for its use in cryptography.

Elliptic Curve Cryptography (ECC) Domain Parameters

The public key cryptographic systems involves arithmetic operations on Elliptic curve over finite fields which is determined by elliptic curve domain parameters.

The ECC domain parameters over F_q is defined by the septuple as given below

$$D = (q, FR, a, b, G, n, h), \text{ where}$$

- q: prime power, that is $q = p$ or $q = 2^m$, where p is a prime
- FR: field representation of the method used for representing field elements $\in F_q$
- a, b: field elements, they specify the equation of the elliptic curve E over F_q, $y^2 = x^3 + ax + b$.

- G : A base point represented by $G = (x_g, y_g)$ on $E(F_q)$.
- n : Order of point G, that is n is the smallest positive integer such that $nG = O$
- h : cofactor, and is equal to the ratio $\#E(F_q)/n$, where $\#E(F_q)$ is the curve order.

The primary security in ECC is the parameter n; therefore the length of ECC key is the bit length of n. For comparative length, the security of ECC keys is much more than that of other cryptosystems. That is for equivalent security, the key length of ECC key is much lesser than other cryptosystems.

Elliptic Curve protocols

Generally in the process of encryption and decryption, we have 2 entities, the one at the encryption side and the other at the decryption side. Let us assume that Alice is the person who is encrypting and Bob is the person decrypting.

Key generation : Alice's (or Bob's) public and private keys are associated with a particular set of elliptic key domain parameters (q, FR, a, b, G, n, h).

Alice generates the public and private keys as follows

1. Select a random number d, $d \in [1, n-1]$
2. Compare $Q = dG$.
3. Alice's public key is Q and private key is d.

It should be noted that the public key generated needs to be validated to ensure that it satisfies the arithmetic requirement of elliptic curve public key. A public key $Q = (x_q, y_q)$ associated with the domain parameters (q, FR, a, b, G, n, h) is validated using the following procedure

1. Check that $Q \neq O$
2. Check that x_q and y_q are properly represented elements of F_q
3. Check if Q lies on the elliptic curve defined by a and b.
4. Check that $nQ = O$

Application of Elliptical Curves in Key Exchange

There are three applications which are described below :

(1) Elliptic Curve Diffie-Helman Protocol (ECDH)

ECDH is elliptic curve version of Diffie-Hellman key agreement protocol. The protocol for generation of the shared secret using ECC is as described below.

- Alice takes a point Q and generates a random number k_a.
- Alice computes the point $P = k_a Q$ and sends it to Bob (It should be noted that Q, P are public).
- Bob generates a random number k_b and computes point $M = k_b Q$ and sends it to Alice.
- Alice now computes $P_1 = k_a M$ and Bob computes $P_2 = k_b P$.
- $P_1 = P_2 = k_b k_b Q$, this is used as the shared secret key.

An illustration of the above steps is represented below.

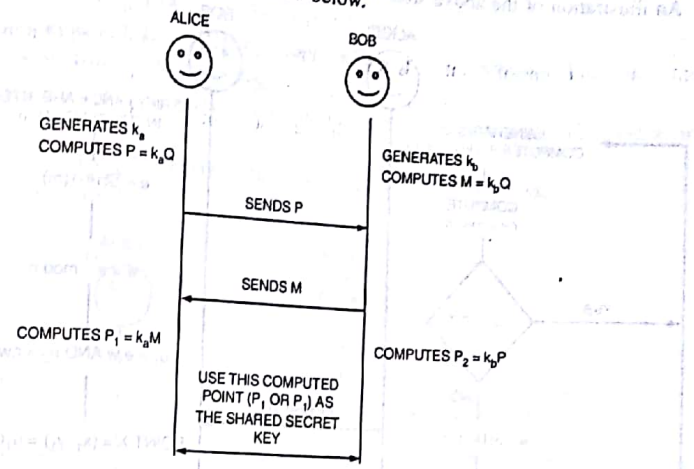


Fig. 6.9. *Illustration of Elliptic Curve Diffie-Hellman Protocol*

(2) Elliptic Curve Digital Signature Authentication (ECDSA)

Alice, with domain parameters $D = (q, FR, a, b, G, n, h)$, public key Q and private key d, does the following steps to sign the message m

Step 1 : Selects a Random number $k \in [1, n-1]$

Step 2 : Computes Point $kG = (x, y)$ and $r = x \bmod n$, if $r = 0$ then goto Step 1

Step 3 : Compute $t = k^{-1} \bmod n$

Step 4 : Compute $e = \text{SHA-1}(m)$, where SHA-1 denotes the 160 bit hash function

Step 5 : Compute $s = k^{-1}(e + d_a * r) \bmod n$, if $s = 0$ goto Step 1

Step 6 : The signature of message m is the pair (r, s)

Step 7 : Alice sends Bob the message m and her signature (r, s)

To verify Alice's signature, Bob does the following (Note that Bob knows the domain parameters D and Alice's public key Q)

Step 1 : Verify r and s are integers in the range $[1, n-1]$

Step 2 : Compute $e = \text{SHA-1}(m)$

Step 3 : Compute $w = s^{-1} \bmod n$

Step 4 : Compute $u_1 = e.w$ and $u_2 = r.w$

Step 5 : Compute Point $X = (x_1, y_1) = u_1 G + u_2 Q$

Step 6 : If $X = O$, then reject the signature

Else compute $v = x_1 \bmod n$

Step 7 : Accept Alice's signature iff $v = r$

An illustration of the above steps is represented below :

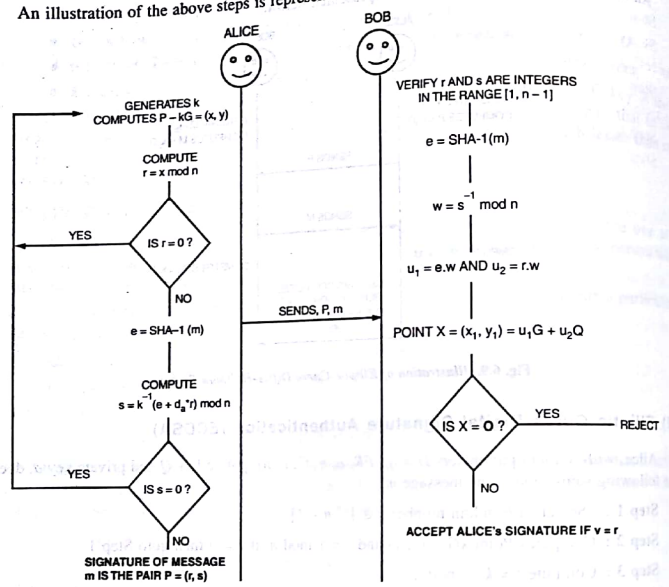


Fig. 6.10. *Illustration of Elliptic Curve Digital Signature Algorithm*

Proof for Verification

If the message is indeed signed by Alice, then $s = k^{-1}(e + d^*r) \bmod n$.

That is, $k = s^{-1}(e + d.r) \bmod n = s^{-1} e + s^{-1} d.r = w.e + w.d.r = (u_1 + u_2.d) \bmod n$ .....[1]

Now consider $u_1G + u_2Q = u_1G + u_2dG = (u_1 + u_2.d)\ G = kG$ from [1]

In step 5 of the verification process, we have $v = x_1 \bmod n$, where,

Point $X = (x_1, y_1) = u_1G + u_2Q$. Thus we see that $v = r$ since $r = x \bmod n$ and x is the x coordinate of the point kG and we have already seen that $u_1G + u_2Q = kG$

(3) Elliptic Curve Authentication Encryption Scheme (ECAES)

Alice has the domain parameters $D = (q, FR, a, b, G, n, h)$ and public key Q. Bob has the domain parameters D. Bob's public key is Q_B and private key is d_B. The ECAES mechanism is as follows.

Alice performs the following steps A does the following :

Step 1 : Selects a random integer r in $[1, n - 1]$

Step 2 : Computes $R = rG$

Step 3 : Computes $K = hrQ_B = (K_x, K_y)$, checks that $K \neq O$

Step 4 : Computes keys $k_1 \| k_2 = \text{KDF}(K_x)$ where KDF is a key derivation function, which derives cryptographic keys from a shared secret.

Step 5 : Computes $c = \text{ENC}_{k1}(m)$ where m is the message to be sent and ENC a symmetric encryption algorithm.

Step 6 : Compute $t = \text{MAC}_{k2}(c)$ where MAC is message authentication code.

Step 7 : Sends (R, c, t) to Bob.

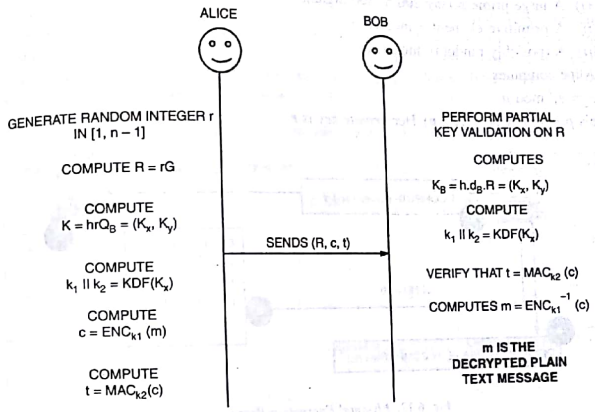


Fig. 6.11. *Illustration of Elliptic Curve Authentication Encryption Scheme*

To decrypt a cipher text, Bob performs the following steps :

Step 1 : Perform a partial key validation on R (check if $R \neq O$, check if the coordinates of R are properly represented elements in F_q and check if R lies on the elliptic curve defined by a and b)

Step 2 : Computes $K_B = h.d_B.R = (K_x, K_y)$, check $K \neq O$

Step 3 : Compute $k_1, k_2 = \text{KDF}(K_x)$

Step 4 : Verify that $t = \text{MAC}_{k2}(c)$

Computes $m = \text{ENC}_{K_1^{-1}}(c)$

We can see that $K = K_B$, since $K = h.r.Q_B = h.r.d_B.G = h.d_B.r.G = h.d_B.R = K_B$.

6.4. ELGAMAL CRYPTOSYSTEM

The ElGamal Public Key Encryption Algorithm : The ElGamal cryptographic scheme is along with RSA one of the most widely used public-key cryptosystems, discrete logarithm problem (DLP). While it can be formulated for any group, usually the multiplicative group of a finite field. The ElGamal Algorithm provides an alternative to the RSA for public key encryption.

(1) Security of the RSA depends on the (presumed) difficulty of factoring large integers.
(2) Security of the ElGamal algorithm depends on the (presumed) difficulty of computing discrete logs in a large prime modulus.

ElGamal has the disadvantage that the ciphertext is twice as long as the plaintext. It has the advantage the same plaintext gives a different ciphertext (with near certainty) each time it is encrypted.

Procedure

1. Alice chooses
   (*i*) A large prime p (say 200 to 300 digits),
   (*ii*) A primitive element e_1 modulo p.
   (*iii*) A (possibly random) integer r.
2. Alice computes
   $e_2 = e_1^{\,r} \bmod p$

Alice's *public key* is (e_1, e_2, p) Her *private key* is r.

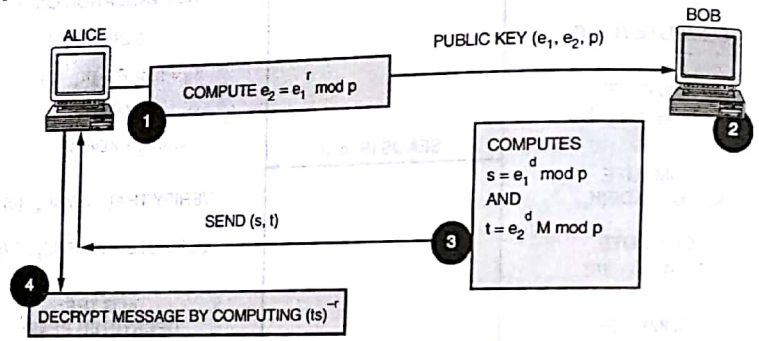


Fig. 6.12. *ElGamal Encryption Process*

Bob encrypts a short message M ($M < p$) and sends it to Alice like this:

(*i*) Bob chooses a random integer d (which he keeps secret).

(*ii*) Bob computes $s = e_1^{\,d} \bmod p$ and $t = e_2^{\,d}\, M \bmod p$ and then discards d.

Bob sends his encrypted message (s, t) to Alice.

When Alice receives the encrypted message (s, t), she decrypts (using her private key r) by computing $(t.s)^{-r}$

$$\begin{aligned}(t.s)^{-r} &= [(e_2^{\,d} \bmod p).(e_1^{\,d}\, M \bmod p)]^{-r} \\ &= [(e_1^{\,r})^{d}.\ M.(e_1^{\,d})^{-r}] \bmod p \\ &= M \bmod p = M\end{aligned}$$

Example : Alice chooses $p = 107$, $e_1 = 2$, $r = 67$, and she computes $e_2 = 2^{67} = 94 \pmod{107}$. Her public key is $(p, e_1, e_2) = (107, 2, 94)$, and her private key is $r = 67$.

Bob wants to send the message "*B*" (66 in ASCII) to Alice. He chooses a random integer $d = 45$ and encrypts M = 66 as (s, t)

$$s = e_1^{\,d} \bmod p \text{ and } t = e_2^{\,d}\, M \bmod p$$

$$s = 2^{45} \bmod 107 \text{ and } t = (94^{45}.\ 66) \bmod 107$$
$$s = 28 \text{ and } t = 9$$

He sends the encrypted message (28, 9) to Alice. Alice receives the message $(s, t) = (28, 9)$, and using her private key $r = 67$ she decrypts to

$$(t.s)^{-r} = 9.\ 28^{-67} = 9.\ 28^{106-67} = 9.\ 28^{43} = 66 \bmod 107 = 66 = M$$

6.5. COMPARISON BETWEEN SYMMETRIC AND ASYMMETRIC KEY CRYPTOGRAPHY

Symmetric key cryptography uses one secret key for encryption and decryption. But it creates the problem of key agreement and key exchange. Asymmetric key cryptography uses the receiver's public key for encryption and receiver's private key for decryption. Asymmetric key cryptography solves the problem of key agreement and key exchange. Comparison between Symmetric and Asymmetric Key Cryptography is shown in Table 6.1.

Table 6.1. Symmetric versus Asymmetric Key Cryptography

S. No.	Characteristic	Symmetric Key Cryptography	Asymmetric Key Cryptography
1	Speed of encryption/ decryption process	Very fast	Slower
2	No. of key used	More keys required ($(m*(m-1))/2$), m is number of participants	Same as number of participants
3	Scalability of keys	Serious issue (more keys required)	No Issue (less keys required)
4	Key used for encryption/ decryption	Same key is used	One key is used for encryption and another for decryption
5	Key exchange	A big problem	No problem
6	Usage	Used for encryption/decryption; not used for digital signature. So usage is limited	Used for encryption/decryption as well as for digital signature. So usage is wide.
7	Size of resulting encrypted text	Usually same as or less than original plaintext	More than original clear plaintext

SUMMARY

- Asymmetric key cryptography aims at resolving the key exchange problem of symmetric key cryptography.
- In Asymmetric key cryptography public and private key of receiver is used.
- RSA is very popular asymmetric key cryptography.
- Each communicating party needs a key pair in asymmetric key cryptography.
- Public key is shared with everybody; private key must be kept secret by the users.
- Prime numbers are the important key factor in asymmetric key cryptography.
- An elliptic curve E over the field F is given by a cubic equation of the form $y^2 = x^3 + ax + b$.
- To make operations on elliptic curve accurate and more efficient, the curve cryptography is defined over two finite fields. **Prime field Z_p and Binary field GF(2^m).**

CHAPTER 7

Message Authentication and Hash Functions

7.1. INTRODUCTION

Inside This Chapter

There are five security services which we have discuss in chapter 1. Message Confidentiality is provided using symmetric and asymmetric key cryptography, which we have discussed in previous chapters.

Now we think about message integrity and message authentication. There are occasions where need of integrity, not secrecy.

Message integrity means message is not changed. To check the message integrity message digest is created from message.

Message Digest

The message is passed through a **hash function.** The hash function creates compressed image of message. The compressed image is called **message digest (MD).**

Suppose a message M is given then message digest will be written as

$$Y = h(M)$$

Where h is hash function.

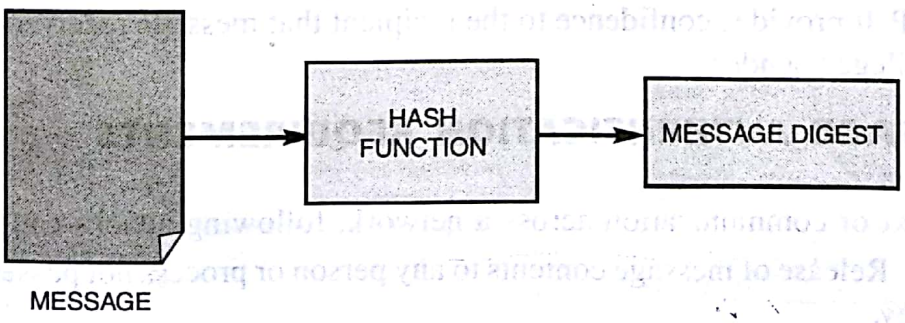


Fig. 7.1. *Message Digest*

How the Integrity Checked

If we want to preserve the integrity of message, then first create the message digest (MD1) of the message and put message digest secure. Later on to check integrity we again create message digest (MD2) of the message. Now

If MD1 = MD2, then original message is not changed

If MD1 ≠ MD2, then original message is changed

Note : We apply Cryptographic hash function algorithms to create message digest (MD).

7.2. PROPERTIES OF MESSAGE DIGEST

A cryptographic hash function must satisfy following properties-

(*i*) Given any two messages, if we calculate their message digest, the two message digest must be different. If any two messages produce same message digest, it is called collision. This property is called **Collision resistance.**

(*ii*) Given a message digest, it should be very difficult to find the message. This property is called **Preimage Resistance.**

(*iii*) Given a specific message and its digest, it is impossible to create another message with the same digest. This property is called **Second Preimage Resistance.**

7.3. MESSAGE AUTHENTICATION

Message authentication is a mechanism or service used to verify the integrity of a message. Message authentication assures that data received are exactly as sent by sender and also coming from original sender.

A message digest guarantees the integrity of a message. It guarantees that a message has not been changed. *A* message digest, however, does not authenticate the sender of the message. When *A* sends a message to *B*, *B* needs to know that the message is coming from *A*. The message digest cannot provide authentication.

Message Authentication = Integrity + Data Origin

Two most common cryptographic techniques to provide message authentication are message authentication code (MAC) and a secure hash function.

Types of Authentication : There are two types of authentication.

- **Peer-to-Peer Authentication :** This service is applicable to a connection oriented environment; like TCP. It provides confidence against masquerade or unauthorized replay.
- **Data Origin Authentication :** This service is applicable to a connection less environment; like UDP. It provides confidence to the recipient that message received has in fact been sent by the alleged sender.

7.4. MESSAGE AUTHENTICATION REQUIREMENTS

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

Disclosure : Release of message contents to any person or process not possessing the appropriate cryptographic key.

Traffic Analysis : Discovery of pattern of traffic between parties. In a connection-oriented application, the frequency and duration of connections could be determined. In either a connection-oriented or connectionless environment, the number and length of message between parties could be determined.

Masquerade : Insertion of message into the network from a fraudulent source. This includes the creation of messages by an opponent that are purported to come from an authorized entity. Also included are fraudulent acknowledgements of message receipt or non-receipt by someone other than the message recipient.

Content Modification : Changes to the content of a message, including insertion, deletion, transposition and modification.

Sequence modification : Any modification to a sequence of messages between parties including insertion, deletion and reordering.

Timing Modification : Delay or Replay of messages. In a connection-oriented application, an entire section or sequence of messages could be a replay of some previous valid session, or individual messages in the sequence could be delayed or replayed. In a connectionless application, an individual message (*e.g.*, datagram) could be delayed or replayed.

Source repudiation : Denial of transmission of message by source.

Destination Repudiation : Denial of receipt of message by destination.

Message authentication is a procedure to verify that received messages come from the alleged source and have not been altered. It may also verify sequencing and timeliness.

7.5. AUTHENTICATION FUNCTIONS

The functions that may be used for authentication purpose grouped into three classes, as follows:

- **Message Encryption :** Ciphertext of entire message works as an authenticator to check the message authentication.
- **Message Authentication Code (MAC) :** A function and secret key that produce fixed length hash code works as an authenticator to check the message authentication.
- **Hash Function :** A function that produces a fixed length hash code of variable length message works as an authenticator to check the message authentication.

Message encryption already we discussed in previous chapters. MAC and Hash function will be discussed in this and next chapter.

7.6. MODIFICATION DETECTION CODE (MDC)

MDC is used to check message integrity. A modification detection code (MDC) is a message digest that can prove the integrity of the message: that message has not been changed.

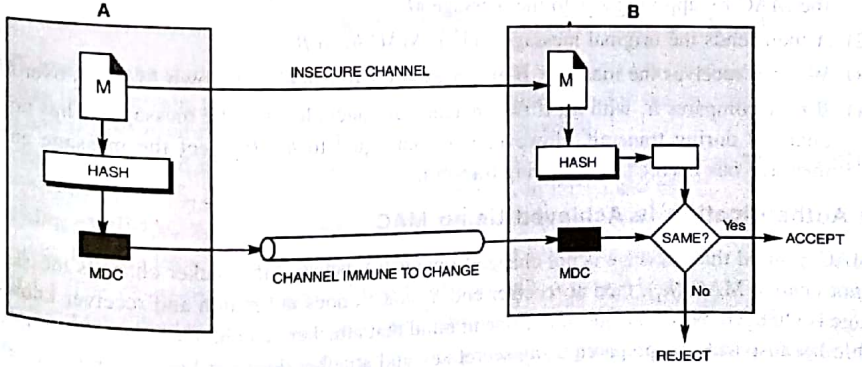


Fig. 7.2. *Modification Detection Code*

A can create a message digest, MDC, and send both the message and the MDC to B. B can create a new MDC from the message and compare the received MDC and the new MDC. If they are the same, the message has not been changed.

Figure 7.2 shows, that the message can be transferred through an insecure channel. Intruder can read or even modify the message. The MDC, however, needs to be transferred through a safe channel. The term safe here means immune to change. If both the message and the MDC are sent through the insecure channel, E can intercept the message, change it, create a new MDC from the message, and send both to B. B never knows that the message has come from intruder.

DEFINITION ▶

The term safe can mean a trusted party; the term channel can mean the passage of time.

7.7. MESSAGE AUTHENTICATION CODE (MAC)

The concept of Message Authentication Code (MAC) is quite similar to that of message digest, but there is basically one difference lies, *i.e.* message digest is simply a fingerprint of a message. It requires no such cryptographic approach as in case of MAC. MAC requires that both sender and receiver should know a shared symmetric (secret) key, which is used in preparation of MAC.

MAC = MDC + Cryptographic Approach

To ensure the integrity of the message and data origin authentication that A is the originator of message, not somebody else—we need to change a modification detection code (MDC), to a message authentication code (MAC). The Fig. 7.3 shows the idea.

A uses a hash function to create a MAC from the concatenation of key and a message. A sends the message and the MAC to B over the insecure channel. B separates the message from the MAC. B then makes the new MAC from the concatenation of message and the secret key. B then compares the newly generated MAC with the one received. If two MAC are equal that means the message is authentic and has not been modified by an adversary.

Procedure

(1) *A* and *B* share a symmetric (secret) key *k*, which is not known to anyone else. A calculates the MAC by applying key to the message *M*.

(2) *A* then sends the original message and the MAC h_1 to *B*.

(3) When *B* receives the message, *B* also uses key *k* to calculate its own MAC h_2 over *M*.

(4) *B* now compares h_1 with h_2, if two matches, *B* concludes that the message *M* has not been changed during transmit. However if h_1 not equal to h_2, *B* reject the message and that message has been changed during transmit.

How Authentication is Achieved Using MAC

MAC assured that message is not changed (integrity), because if attacker changes message but does not change MAC (h_1), then at receiver end h_1 and h_2 does not match and receiver knows that message is changed. Now one question came in mind that attacker can change digest also, this is not possible because MAC is prepared using secret key and attacker does not know secret key.

MAC assured that message is received from A (data origin), because if attacker send its own created MAC (using own key K_1) then MAC will not be opened at receiver end. This is why secret key is K.

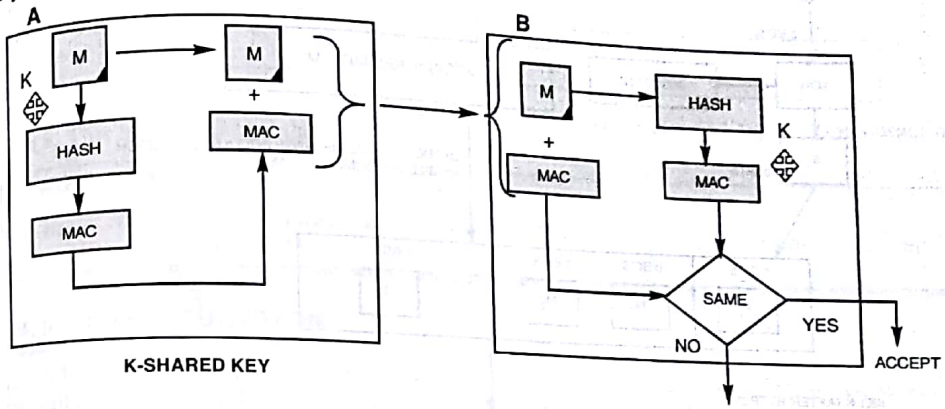


Fig. 7.3. *Message Authentication Code (MAC)*

Security of MAC

(*a*) If the size of keys allows exhaustive search, intruder may prepend all possible keys at the beginning of message and make a digest of the (KIM) to find the digest equal to one intercepted. He then replaces the message.

(*b*) The size of key in MAC is normally large she use algorithm until she finds x such that h(x) is equal to the MAC she has intercepted. She now find the key and successfully replace the message with a forged one. Because the size of key is normally very large for exhaustive search.

NOTE ▶

Security of MAC depends on the security of used hash algorithm.

7.8. HASH-BASED MESSAGE AUTHENTICATION CODE (HMAC)

NIST has issued a standard (FIPS 198) for a NESTED MAC that is often referred to as HMAC. HMAC has been chosen as a mandatory security implementation for the internet protocol (IP) security, and is also used in the secure socket layer (SSL) protocol.

The basic idea behind HMAC is to reuse existing message digest algorithm (*e.g.*, MD5 or SHA-1 etc) for calculating MAC. So HMAC is practical algorithm to implement MAC.

Working of HMAC : The working of HMAC is shown in Fig. 7.4.

To make HMAC from original message (M) follow the following steps.

Step 1 : The message (M) is divided into N blocks, each of size *b* bits.

Step 2 : Make the length of secret key (K) equal to *b* bits by left padding of 0's.

Step 3 : XOR K (output of step 2) with ipad to produce S_1 ipad is input pad, it's value is 00110110 in hexadecimal.

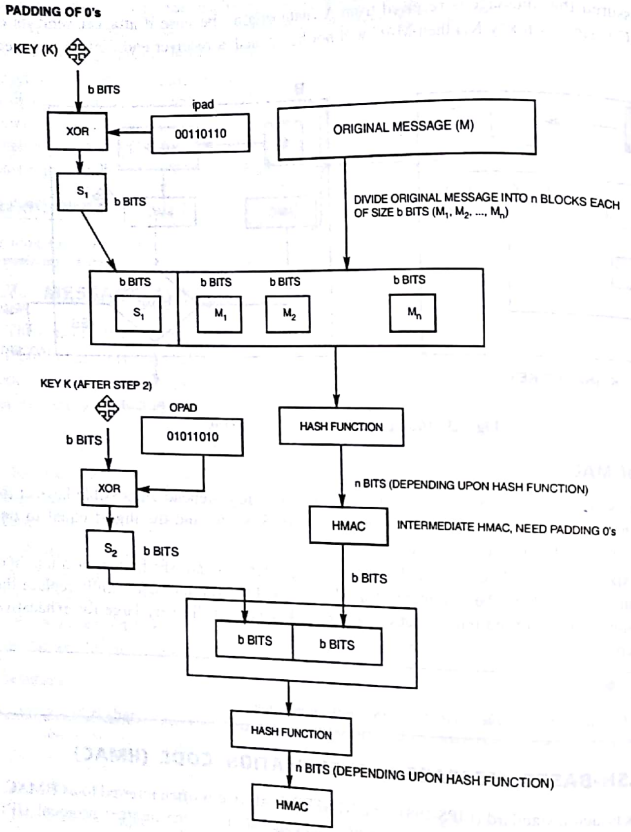


Fig. 7.4. *HMAC*

Step 4 : Combine S_1 and M, now original message M is simply append to the end of S_1.

S_1	Original Message (M) divides into blocks of size b bits

Step 5 : Select the appropriate message digest algorithm (MD5 or SHA-1 etc) and apply on the output of step 4. Say the output H. It is intermediate HMAC.

Step 6 : Make the length of H (output of step 5) equal to *b* bits by left padding of 0's. There is need of padding because H (output of step 5) may not be of b bits. Size of H depends on message digest algorithm used *e.g.*, MD5 gives 128 bits.

Step 7 : XOR K (output of step 2) with opad to produce S_2. opad is output pad, it's value is 01011010 in hexadecimal.

Step 8 : Take the message digest H (output of step 6) and append it to the end of S_2 (output of step 7).

S_2	H (Output of Step 6)

Step 9 : Select the message digest algorithm (MD5 or SHA-1 etc) and apply to the output of Step 8. This is final HMAC that we want.

NOTE

The size of HMAC depends on message digest algorithm (MD5 or SHA-1 etc) used in step 9 and step 5, e.g., if MD5 is taken then HMAC will be of size 128 bits.

7.9. HASH FUNCTION

A function that maps a message of any length into a fixed-length hash value, which serves as the authenticator. 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. Hash function accepts a variable size message *M* as input and produces a fixed size output, referred to as a hash code *H*(*M*). The hash code is also referred to as a message digest or hash value. The hash code is a function of all the bits of the message and provides an error detection capability: A change to any bit or bits in the message results in a change to the hash code.

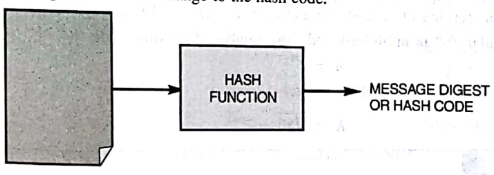


Fig. 7.5. *Idea of Hash Function*

So the integrity of message depends on the hash function we have used.

7.10. CHARACTERISTICS OF A HASH FUNCTION

Most cryptographic hash functions are designed to take a string of any length as input and produce a fixed length hash value.

To be useful for message authentication, a hash function *H* must satisfy the following properties :

1. **One Way Property :** A hash function is a one way function. It is easy to generate hash value of a given message but it impossible to determine M from given hash value *h*.
2. **Weak Collision Resistance :** Given an input M_1 it should be hard to find another input M_2 where $M_1 \neq M_2$ such that hash (M_1) = hash (M_2). Means is that it is impossible that two different messages have same hash value *h*.

3. Strong Collision Resistance : It should be extremely hard to determine a message pair (M_1, M_2) having same hash value h.

7.11. BIRTHDAY ATTACK

There are different types of attacks possible based on birthday paradox. First try to understand what is birthday paradox.

Birthday Paradox : What is the minimum number, k, of students in a classroom such that it is *likely* that at least two students have the same birthday? In other words what is the minimum value of k such that the probability is greater than 0.5 that at least two people in a group of k people have same day birthday? It is called birthday paradox. Birthday paradox is general probability problem. In terms of hash function birthday paradox can be explained as :

A general problem relating to hash functions is that given a hash function H, with n possible outputs and a specific value $H(x)$, if H is applied to k random inputs, what must be the value of k so that the probability that at least one input satisfies $H(y) = H(x)$ is 0.5?

The birthday problem can be generalized to the following problem:

Given a random variable that is an integer with uniform distribution between 1 and n and a selection of k instances ($k <= n$) of random variable, what is probability $P(n, k)$, that is at least one duplicate? The equation to find $P(n, k)$ will be

$$P(n, k) = 1 - [n!/((n - k)!\ n^k)]$$

When we solve it

$$K = \sqrt{n} \qquad ...(8.1)$$

We can state the birthday attack in terms of hash function as suppose we have a hash function H, with 2^m possible outputs. If H is applied to k random inputs, what must be the value of k so that there is the probability of at least one duplicate, *i.e.*, $H(x) = H(y)$.

In particular, for an m bit hash code, the number of possible codes is

$$n = 2^m$$

Put the value of n in Eq. (8.1), we get

$$K = \sqrt{2^m} = 2^{m/2}$$

NOTE

There are four types of birthday problem in probability, out of which one is called birthday paradox; it is related to birthday attack on hash function.

7.12. MESSAGE DIGEST OR HASH FUNCTION ALGORITHMS

There are several hash algorithm which works as a hash function to find message digest (MD). In this chapter, we discuss two important hash algorithms :

(*a*) MD5 (*b*) Secure Hash Algorithm (SHA)

7.12.1. Message Digest 5 (MD5)

MD5 is developed by Ron Rivest. The original message digest algorithm developed by Ron Rivest was MD, where MD stands for message digest. The versions of MD are MD2, MD4, and MD5.

The last version MD5 is quite fast and produce 128 bits message digest from a variable length message, *i.e.*, whatever the size of message, message digest created through MD5 will be of length 128 bits.

MD5 Logic : The algorithm takes as input a message with a maximum length of less than 2^{64} bits and produces as output of a 128-bit message digest. The message is divided into 512-bit blocks ($M_1, M_2 \ldots M_N$). The general working of MD-5 is shown in Fig. 7.7.

The initial value (hash buffer or IV) is initialized to a predefined value of 128 bits. The algorithm mixes this initial value (**IV**) with the first message block value (M_1) to create intermediate digest (H_1) of 128 bits. This digest (H_1) is then mixed with second block M_2. Finally $(N - 1)^{th}$ digest (H_{N-1}) is mixed with N^{th} block M_N to create N^{th} digest H_N. This is the message digest for entire message.

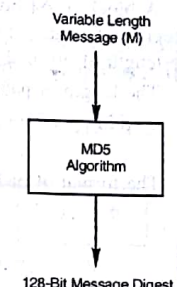


Fig. 7.6. *General Idea of MD5*

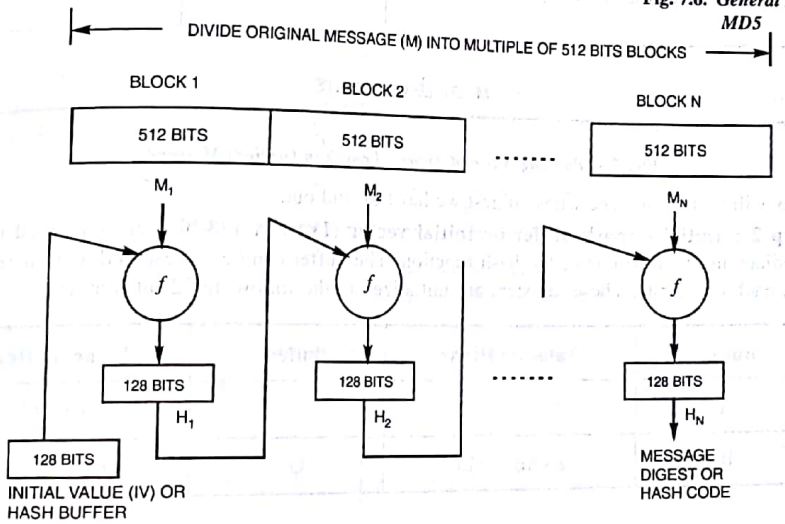


Fig. 7.7. *Idea of Message Digest Generation Using MD5*

Working of MD5

To understand the working of MD5, we divide into following steps :

Step 1 : Append Padding bits and Append length : The message is padded so that its length is 64 bits short of a multiple of 512. Padding is always added, even if message is already 64 bits short of a multiple of 512.

For example, suppose message is of length 1200 bits, now we add a padding of 272 bits. This is shown in Fig. 7.8.

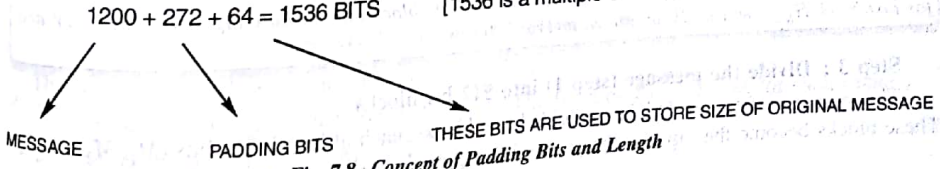


Fig. 7.8. *Concept of Padding Bits and Length*

A block of 64 bits is appended to the message. This block is treated as an unsigned 64 bits integer and contains the length of the original message. The basic purpose of these 64 bits is to store the length of original message.

The length of padding bits can be calculated as follows.

$$(|M| + |P| + 64) \bmod 512 = 0$$

$$P = (-|M| - 64) \bmod 512$$

The format of padding bits is 1 followed by the necessary number of 0's.

100000.....0000

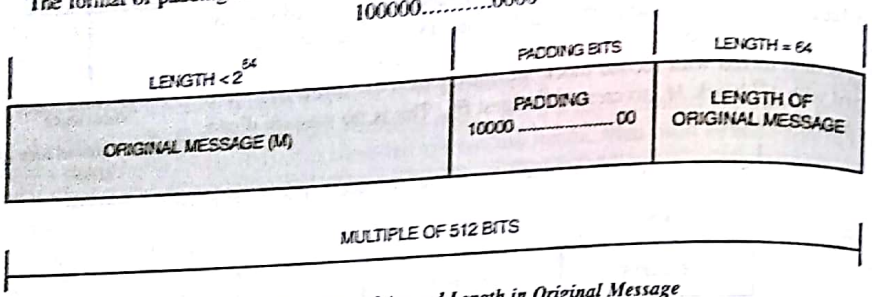


Fig. 7.9. Padding Bits and Append Length in Original Message

This will be the message whose digest we have to find out.

Step 2 : Initialize hash buffer or Initial vector (IV) : A 128-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as four registers A to D, each of 32 bits. These registers are initialized to the following 32-bit integers:

Buffer	Value (in Hexa)	Buffer	Value (in Hexa)
A	01 23 45 67	C	FE DC BA 98
B	89 AB CD EF	D	76 54 32 10

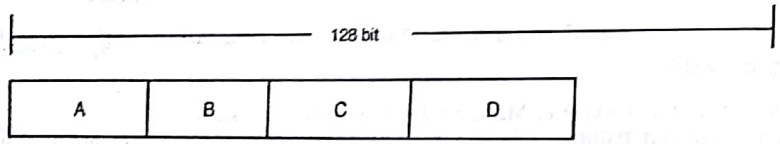


Fig. 7.10. Initialize Hash Buffer or Initial Vector (IV)

These 128 bits (32*4) value is called initial value (IV) or hash buffer.

NOTE

For first 512 bits block this will be IV, for second 512 bits block intermediate digest H_1 will be IV and for last block H_{N-1} will be IV as shown in Fig. 7.8.

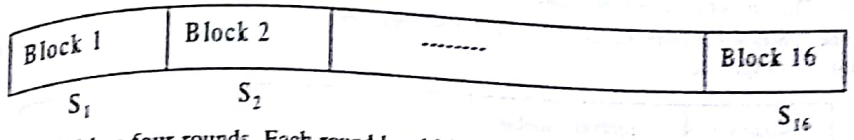
Step 3 : Divide the message (step 1) into 512 bit blocks

The message (after step 1) is now divided into blocks, each of length 512 bits (M_1, M_2 .... M_N). These blocks become the input to the processing logic (f) as shown in Fig. 7.7.

Step 4 : Process Message Blocks

Now we are having blocks of size 512 bits. Now we try to understand the process on first block M_1. Same process will be applied on other blocks, only IV will be changed. We can divide the process into following steps :

(i) Divide the current 512-bit block into 16 sub-blocks, each of 32 bits. (32*16=512). The sub blocks are S_1, S_2 ... S_{16}.



(ii) MD5 has four rounds. Each round has 16 iterations, *i.e.*, there are total 64 iterations. Each round takes three inputs as

(i) 16 sub blocks (S_1, S_2, ..., S_{16})

(ii) 128-bit hash buffer values $A - D$

(iii) Array of constants $K[t]$ (t = 1 to 64). Total 64 constants defined (in Hexadecimal)

(iii) **Structure of each iteration :** MD5 has four rounds, each round has 16 iterations. This makes total 64 iterations. The working of each iteration is shown in Fig. 7.11.

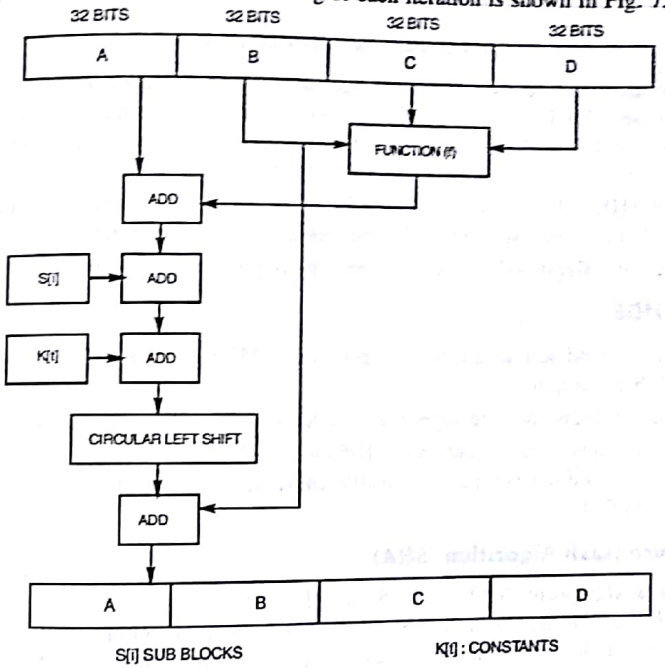


Fig. 7.11. *Working of Each Iteration in MD5*

The working of each iteration can be understand as :

1. The function f is performed on B, C and D. The function f is different for each round.
2. The output of step 1 is added with A.

3. The message sub blocks ($S[i]$) are added with output of step 2. For example, for first iteration $S[1]$ sub block will be used.
4. The constants ($K[i]$) are added with output of step 3.
5. The output of step 4 is circular left shift by s bits which is changed as iteration change.
6. At last output of step 5 is added with B.
7. The output of step 6 becomes new ABCD for next iteration.

The function f is different for each round. There are total 4 rounds, so function f for each round is shown in Fig. 7.12.

Round	Iteration Number	Function F
1	0 to 16	(B AND C) OR ((NOT B) AND D)
2	17 to 32	(B AND D) OR (C AND (NOT D))
3	33 to 48	B XOR C XOR D
4	49 to 64	C XOR (B OR (NOT D))

Fig. 7.12. Function (f) for Each Iteration

Step 5 : When four rounds on first message block M_1 (512 bits) complete then the values stores in A, B, C and D will work Initialize vector (IV) for second message block M_2 (512 bits) and after all rounds complete on N^{th} message block M_N then values stores in A, B, C and D will be final message digest for original message (M). The size of message digest will be 128 bits.

Strength of MD5 : Ron Rivest tried to add more and more complexity as much as possible. The possibility that two messages produce same message digest is in order of 2^{64} operations.

Given a message digest and to find corresponding original message can take 2^{128} operations.

Attacks on MD5

1. Dobbertin found serious attack, the operation of MD5 on two different message produce same 128-bit output.
2. Tom Berson found two messages that produce same digest for each of four rounds.
3. Den Boer showed that execution of MD5 on a single block of 512 bits will produce same output for two different values in hash buffer register ABCD.

7.12.2. Secure Hash Algorithm (SHA)

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 (FIPS 180) in 1993. It is sometimes referred to as Secure Hash Standard (SHS). A revised version was issued as FIPS 180-1 in 1995 and name is changed to SHA-1. SHA-1 is a modified version of MD5. We provide a description of SHA-1:

Fig. 7.13. General Idea of SHA-1

SHA-1 Logic

The algorithm takes as input a message with a maximum length of less than 2^{64} bits and produces as output a 160-bit message digest. The message is divided into 512 bit blocks ($M_1, M_2 \ldots M_N$). The general working of SHA-1 is shown in Fig. 7.14.

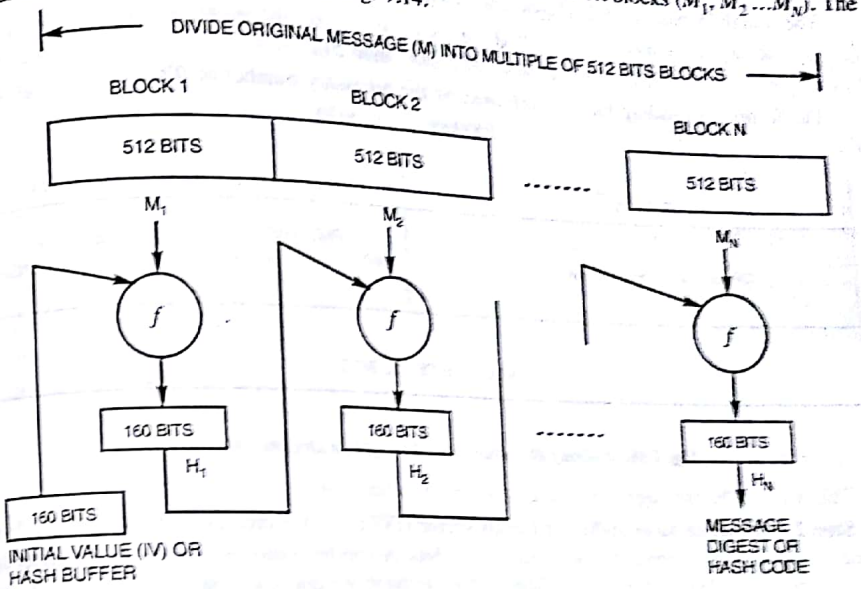


Fig. 7.14. Idea of Message Digest Generation Using SHA-1

The initial value (hash buffer or IV) is initialized to a predefined value of 160 bits. The algorithm mixes this initial value (**IV**) with the first message block value (M_1) to create intermediate digest (H_1) of 160 bits. This digest (H_1) is then mixed with second block M_2. Finally $(N-1)^{th}$ digest (H_{N-1}) is mixed with N^{th} block M_N to create N^{th} digest H_N. This is the message digest for entire message.

Working of SHA-1

To understand the working of SHA-1, we divide into following steps :

Step 1 : Append Padding bits and Append length : This step is same as in MD5.

The message is padded so that its length is 64 bits short of a multiple of 512. Padding is always added, even if message is already 64 bits short of a multiple of 512.

For example- Suppose message is of length 1200 bits, now we add a padding of 272 bits. This is because,

$1200 + 272 + 64 = 1536$ BITS [1536 is a multiple of 512, $512 * 3 = 1536$]

Fig. 7.15. Concept of Padding Bits and Length

A block of 64 bits is appended to the message. This block is treated as an unsigned 64 bits integer and contains the length of the original message. The basic purpose of these 64 bits is to store the length of original message.

The length of padding bits can be calculated as follows.

$$(|M| + |P| + 64) \bmod 512 = 0$$

$$P = (-|M| - 64) \bmod 512$$

The format of padding bits is 1 followed by the necessary number of 0's.

100000.....0000

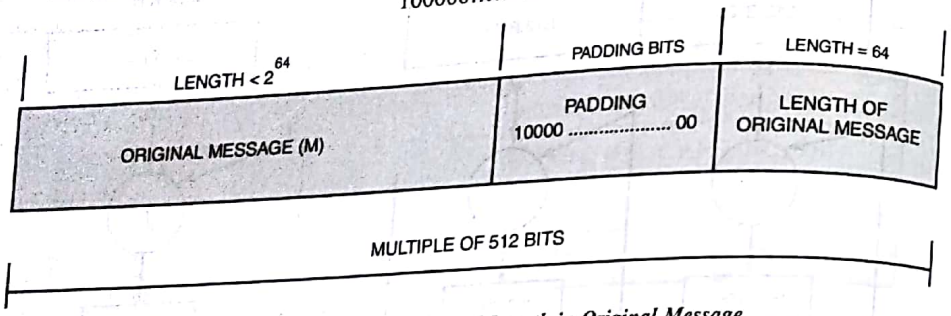


Fig. 7.16. Padding Bits and Append Length in Original Message

This will be the message whose digest we have to find out.

Step 2 : Initialize hash buffer or Initial vector (IV) : SHA-1 creates 160-bit message digest, so there is need of five registers A to E, each of 32 bits. A 160-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as four registers A to E, each of 32 bits. These registers are initialized to the following 32-bit integers:

Buffer	Value (in Hexa)	Buffer	Value (in Hexa)
A	01 23 45 67	C	FE DC BA 98
B	89 AB CD EF	D	76 54 32 10
E	C3 D2 E1 F0		

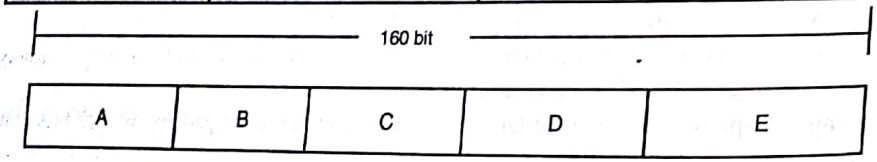


Fig. 7.17. Initialize hash buffer or Initial vector (IV)

These 128 bits (32*4) value is called initial value (IV) or hash buffer.

NOTE

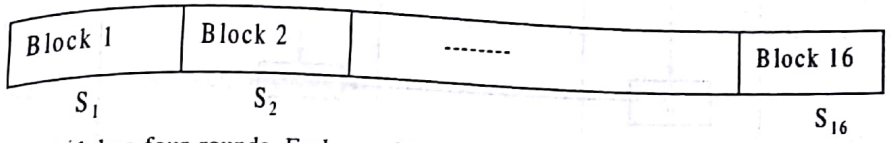
For first 512 bits block this will be IV, for second 512 bits block intermediate digest H_1 will be IV and for last block H_{N-1} will be IV as shown in Fig. 7.17.

Step 3 : Divide the message (step 1) into 512-bit blocks

This is same as in MD5. The message (after step 1) is now divided into blocks, each of length 512 bits (M_1, M_2....M_N). These blocks become the input to the processing logic (f).

Step 4 : Process Message Blocks : Now we are having blocks of size 512 bits. Now we try to understand the process on first block M_1. Same process will be applied on other blocks, only IV will be changed. We can divide the process into following steps :

(i) Divide the current 512-bit block into 16 sub-blocks, each of 32 bits. (32*16 = 512). The sub blocks are S_1, S_2 ... S_{16}.



(ii) SHA-1 has four rounds. Each round has 20 iterations, *i.e.*, there are total 80 iterations. Each round takes three inputs as

(a) 16 sub blocks (S_1, S_2, ..., S_{16})

(b) 160-bit hash buffer values A-E

(c) Array of constants $K[t]$ (t = 1 to 80). Total 4 constants defined for each round (in Hexadecimal).

For each round there are four constants defined for $K[t]$.

Round	Value of t	K[t] (in hexa)
1	0 to 20	5A 92 79 99
2	21 to 40	6E D9 EB A1
3	41 to 60	9E 1B BC DC
4	61 to 80	CA 62 C1 D6

Fig. 7.18. Values of Constants K[t]

***(iii)* Structure of each iteration :** SHA-1 has four rounds, each round has 20 iterations. This makes total 80 iterations. The working of each iteration is defined in figure.

The working of each iteration can be understand as :

1. The function f is performed on B, C and D. The function f is different for each round.
2. The output of step 1 is added with E.
3. A is circular left shift by 5 bits.
4. The output of step 2 and step 3 are added.
5. The output of step 4 is added with W[t]; total 80 words are calculated.
6. The output of step 5 is added with K[t]; total 4 constants are used defined in Fig. 7.18.
7. The output of step 6 becomes new value of A (32 bits).
8. The value of A copied into B, this becomes new value of B (32 bits).

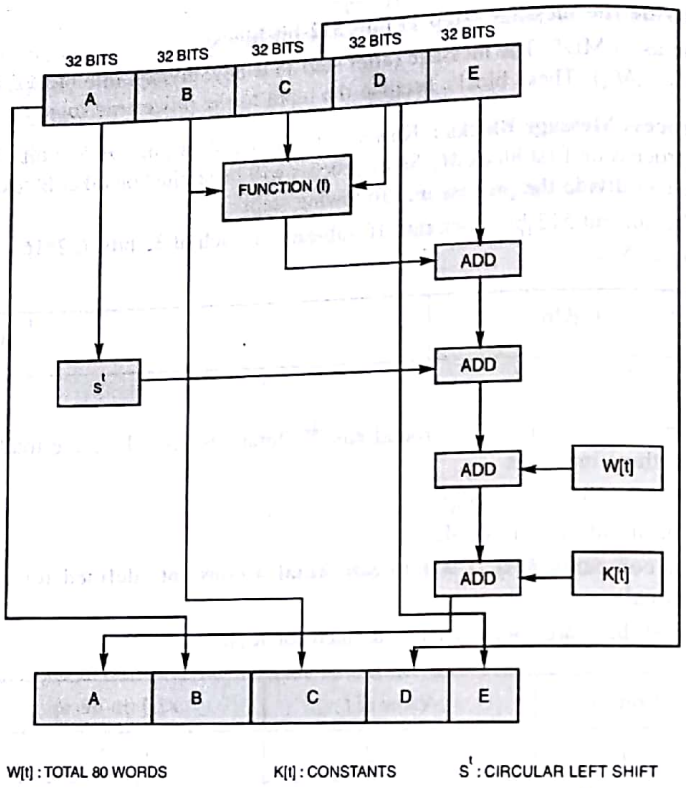


Fig. 7.19. *Working of each iteration in SHA-1*

9. The value of B copied into C, this becomes new value of C (32 bits).
10. The value of C copied into D, this becomes new value of D (32 bits).
11. The value of D copied into E, this becomes new value of E (32 bits).

The function f is different for each round. There are total 4 rounds, so function f for each round is shown in Fig. 7.20.

There is need of 80 words values. The values of $W[t]$ can be derived using following formula-

$W[t] = S[t]$, if $0 = t < 16$
$W[t] = S^1 (W[t-16]$ XOR $W[t-14]$ XOR $W[t-8]$ XOR $W[t-3])$, if $16 = t = 79$

Round	Iteration Number	Function (*f*)
1	0 to 19	(B AND C) OR ((NOT B) AND D)
2	20 to 39	B XOR C XOR D
3	40 to 59	(B AND C) OR (B AND D) OR (C AND D)
4	60 to 79	B XOR C XOR D

Fig. 7.20. *Function (f) for Each Iteration*

First 16 words of W (t = 0 to 15), the contents of the input message sub-block $S[t]$ (step 4, i) part) becomes the contents of $W[t]$. Rest 64 values are calculated using given formula.

S^1 indicates circular left shift by 1 bit.

S^t indicates circular left shift by t bits.

$K[t]$ is defined earlier.

Step 5 : When four rounds on first message block M_1 (512 bits) complete then the values stores in A, B, C, D and E will work Initialize vector (IV) for second message block M_2 (512 bits) and after all rounds complete on N^{th} message block M_N then values stores in A, B, C, D and E will be final message digest for original message (M). The size of message digest will be 160 bits.

Comparison between MD5 and SHA-1

Parameter	MD5	SHA-1
Message digest length	128 bits	160 bits
Attack on to find original message from digest	Require 2^{128} operation to break	Require 2^{160} operation to break
Attack on to find two messages with same digest	Require 2^{64} operation to break	Require 2^{80} operation to break
Speed	Faster, because 64 iteration, 128 bit hash buffer	Slower, because 80 iteration, 160 bit hash buffer
Implementation	Simple	Hard
Rounds	64	80
Space	Less	More

7.12.3. Versions of SHA

NIST produced a revised version of the standard, FIPS 180-2, that defined four new versions of SHA, known as **SHA-224, SHA-256, SHA-384**, and **SHA-512**. The table shows these versions.

Parameter	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Block size (in bits)	512	512	512	1024	1024
Word Size (in bits)	32	32	32	64	64
Message digest Size (in bits)	160	224	256	384	512
Rounds	80	64	64	80	80
Message Size (Maxi) (in bits)	$<2^{64}$	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$

7.13. SECURITY OF HASH FUNCTIONS

The attacks in hash function can be divided into two categories :

1. Brute force attack
2. Cryptanalysis

1. Brute Force Attack

The strength of hash function depends on the length of hash code created by that algorithm.

For hash code of size n

- The efforts required for birthday attack is of order of $2^{m/2}$, where m is size of hash code. For example to break 128-bit message digest required efforts are :

$$2^{128/2} = 2^{64}$$

So size of hash code should be larger.

- The effort required to break 'Weak Collision Resistance' in hash function will be of order 2^m.
- The efforts required to break 'Strong Collision Resistance' in hash function will be of order $2^{m/2}$.

Oorschot and Wiener presented a collision search machine for MD5, which has a 128 bit hash length that could find a collision in 24 days. Thus 128-bit code is inadequate.

2. Cryptanalysis

Cryptanalysis attacks on hash functions seek to exploit some prosperity of the algorithm or internal structure of hash function.

In recent years, there has been some success in developing cryptanalytic attacks on hash functions. To understand these, we need to know internal structure of hash function.

Iterated hash function proposed by Merkle and mostly hash function SHA, Whirlpool used this. It is called iterated hash function.

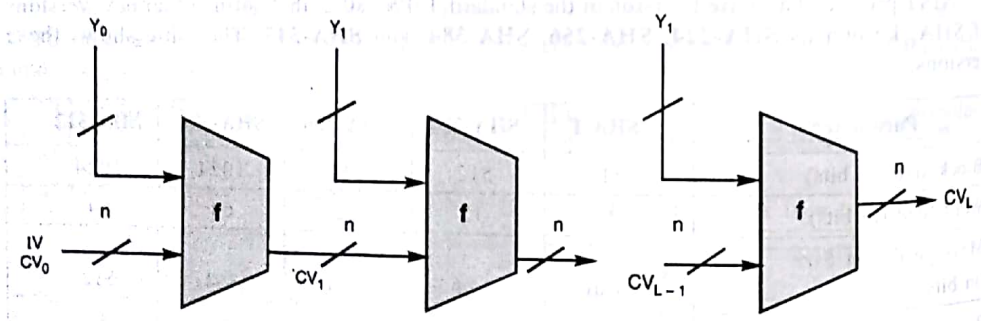


Fig. 7.21. *Structure of Secure Hash Function*

The main component of secure hash function is compression function f. If compression function is collision resistant then so is the iterated hash function. So security of hash function depends on compression function. Therefore 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 function focuses on internal structure of f and is based on attempts to find efficient techniques for producing collisions for a single execution of f.

SUMMARY

- Message integrity means message is not changed or altered.
- To check the message integrity message digest is created from message.
- Given any two messages, if we calculate their message digest, the two message digest must be different. This property is called **Collision resistance.**
- Given a message digest, it should be very difficult to find the message. This property is called **Preimage Resistance.**
- Given a specific message and its digest, it is impossible to create another message with the same digest. This property is called **Second Preimage Resistance.**
- Message Authentication means message Integrity and message Data Origin.
- A modification detection code (MDC) is a message digest that can prove the integrity of the message: that message has not been changed.
- Message Authentication is achieved by a message authentication code (MAC).
- MAC is created by applying security to MDC.
- Security of MAC depends on the security of used hash algorithm.
- HMAC is practical algorithm to implement MAC.
- HMAC is a message digest algorithm that involves encryption.
- The size of HMAC depends on message digest algorithm used.
- Hash function accepts a variable size message M as input and produces a fixed size output, referred to as a hash code H(M) or message digest.
- What is the minimum number, k, of students in a classroom such that it is likely that at least two students have the same birthday? In other words what is the minimum value of k such that the probability is greater than 0.5 that at least two people in a group of k people have same day birthday? It is called birthday paradox.
- MD5 and SHA-1 are message digest algorithms.
- The versions of MD are MD2, MD4, and MD5.
- MD5 produce 128 bits message digest from a variable length message.
- MD5 has four rounds, each round has 16 iterations.
- SHA-1 is a modified version of MD5.
- SHA-1 produce 160 bits message digest from a variable length message.
- SHA-1 has four rounds, each round has 20 iterations.
- Four new versions of SHA are **SHA-224, SHA-256, SHA-384,** and **SHA-512.**

Digital Signature

8.1. INTRODUCTION

Inside This Chapter

In chapter 1, we study some security services confidentiality, integrity, authentication and non-repudiation. Using symmetric or asymmetric key cryptosystem we can achieve confidentiality. Digital signature is used to achieve integrity, authentication and non-repudiation.

In chapter 7, we study message authentication using message authentication code (MAC). Message authentication protects sender and receiver from attacker. Till here we are thinking about only attacker (Third party).

Suppose *A* send an authenticated message (*i.e.,* using MAC) to *B*. Now think the following two cases :

1. *A* can deny sending the message.
2. *B* can forge a different message and claim that it came from *A*. Simply *B* create a message and create authentication code using secret key.

In such situations when there is no trust between sender and receiver, something more than authentication is required. The most suitable solution is use of digital signature.

Definition

A digital signature is an authentication mechanism that enables the creator of a message to attach a code that acts as a signature. The signature is formed by taking the hash of the message and encrypting the message with the creator's private key. The signature guarantees the source and integrity of the message.

8.2. PROPERTIES OF DIGITAL SIGNATURE

It must have following properties :

- It must verify the author and the date and time of the signature.

- It must be authenticate the contents at the time of the signature.
- It must be verifiable by third parties, to resolve disputes.

8.3. PROCESS OF DIGITAL SIGNATURE

We have study public key cryptosystem. Digital signature is based on public key cryptosystem. Both uses public key. The difference between asymmetric key cryptosystem and digital signature is :

In digital signature public and private key of sender is used. But in asymmetric key cryptosystem public and private key of receiver is used.

The general idea of digital signature can be understand as shown in Fig. 8.1.

The sender (A) uses her private key, applied to a signing algorithm, to sign a message. On the other hand, the receiver (B) uses the public key of the sender, applied to the verifying algorithm, to verify the document.

NOTE

When a document is signed, anyone, including receiver can verify it because everyone has access to sender's public key. Sender must not use her public key to sign the document because then anyone could forge her signature.

Drawback : We know that digital signature uses asymmetric key cryptosystem. Asymmetric key system is slow when message is long, so digital signature process also be.

One solution of this drawback is to sign the message digest rather than signing message.

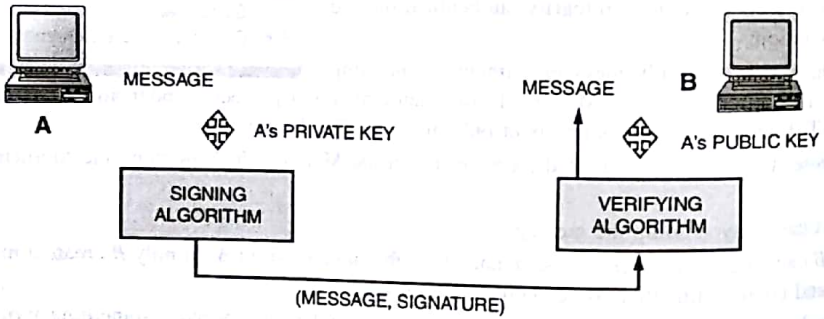


Fig. 8.1. *Digital Signature Process*

8.4. PROCESS OF SIGNING THE DIGEST

When message is too long then sign the message digest. The Fig. 8.2 shows the process. The message is passed through a hash algorithm (*e.g.*, SHA) to create message digest (MD). Now this message digest is input for signing algorithm. Suppose signature created is S.

At receiver end, message is again passed through hash algorithm to create message digest. Now from message digest signature will be created, suppose S'. Now checked

$$S = S'$$

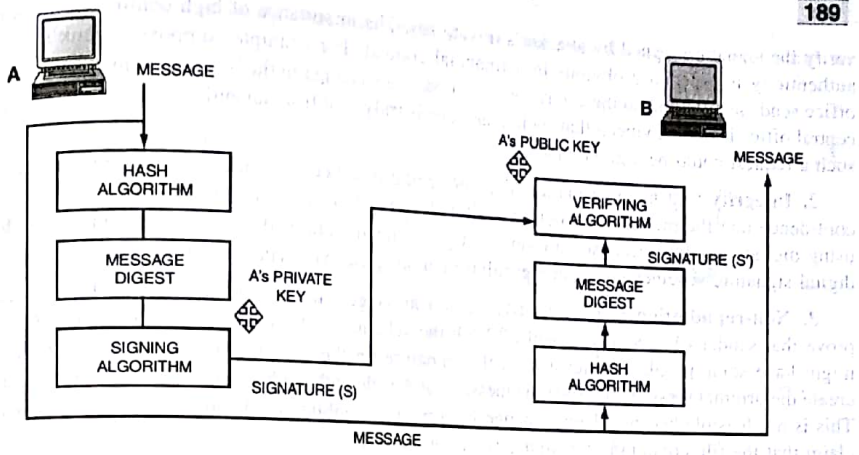


Fig. 8.2. *Signing the Digest*

8.5. SERVICES OR BENEFITS OF DIGITAL SIGNATURES

Digital signature is used to provide following types of security services.

1. Authentication : Digital signature scheme can provide message authentication. Sender (A) creates digital signature and (message, signature) both are sends to receiver. Receiver can check that message came from A, because A public key is used in verification process. A's public key can't

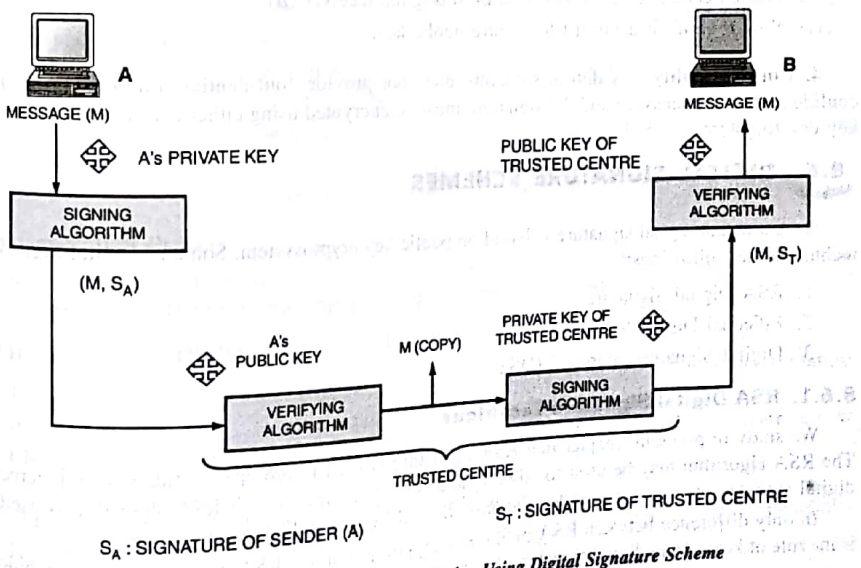


Fig. 8.3. *Process of Providing Non-repudiation Using Digital Signature Scheme*

verify the signature signed by attacker's private key. The importance of high confidence in sender authenticity is especially obvious in a financial context. For example, suppose a bank's branch office sends instructions to the central office requesting a change in the balance of an account. If the central office is not convinced that such a message is truly sent from an authorized source, acting on such a request could be a grave mistake.

2. Integrity : In many scenarios, the sender and receiver of a message may have a need for confidence that the message has not been altered during transmission. The integrity is maintained using digital signature because attacker can't get same signature, if message is changed. Today's digital signature schemes use hash algorithm in signing and verifying process.

3. Non-repudiation : If sender (A) signs a message and then denies it. Can later receiver prove that sender (A) actually signed it? With the scheme we have presented so far, receiver (B) might have some problem. B must keep the signature on the file and later use A's public key to create the original message to prove the message in the file and newly created message are the same. This is not feasible because A may change her private or public key during this time; he may also claim that the file containing the signature is not authentic.

One solution is a trusted third party. Figure 8.3 shows the concept.

Process : The process to provide non-repudiation can be understand in following steps :

(i) Sender (A) signs the message (M) using own private key. Suppose sign created is S_A.

(ii) Sender sends M and S_A to trusted centre.

(iii) Now trusted centre verify it using A's public key.

(iv) Trusted centre sign the original message (M) using own private key (trusted centre private key). Suppose sign created is S_T. Also centre keep original copy of message (M).

(v) Trusted centre sends M and S_T both to original receiver (B).

(vi) Now B verify it using trusted centre public key.

4. Confidentiality : A digital signature does not provide confidential communication. For confidentiality, the message and the signature must be encrypted using either a secret key or public key cryptosystem.

8.6. DIGITAL SIGNATURE SCHEMES

We know that digital signature is based on public key cryptosystem. Some of Digital Signature techniques are followings:

1. RSA Digital Signature
2. ElGamal Digital Signature
3. Digital Signature Standard (DSS)

8.6.1. RSA Digital Signature Technique

We study in previous chapter that RSA algorithm is used to provide confidentiality (secrecy). The RSA algorithm may be used for signing and verifying a message. If RSA algorithm is used in digital signature process, it is called the RSA Digital Signature Technique.

In only difference between RSA public key cryptosystem and RSA digital signature technique is the role of keys. As in digital signature the private and public key of the sender are used. The sender

uses her own private key to sign the document; the receiver uses the sender's public key to verify it.

The process of signing and verifying can be understood as given in Fig. 8.4.

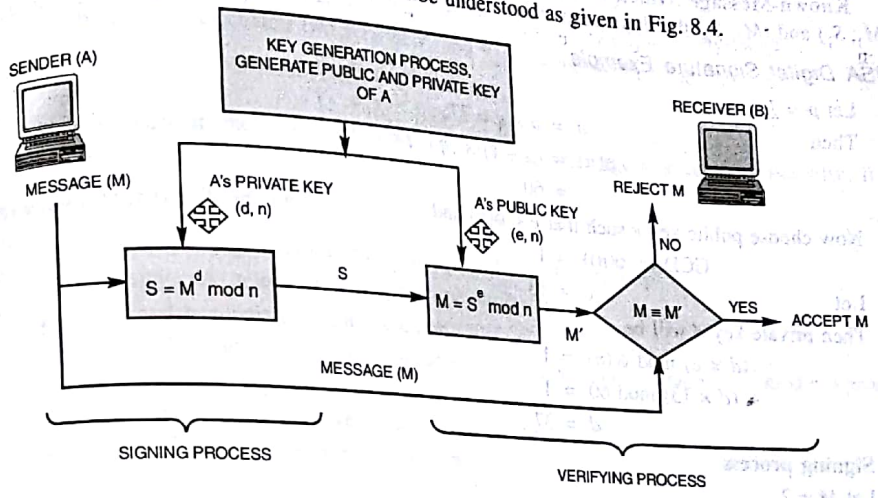


Fig. 8.4. *RSA Digital Signature Techniques*

The signing and verifying sites use the same function, but with different parameters. The verifier compares the message and the output of the function for congruence. If the result is true message is accepted.

Signing : Signer using her private key to create signature S using, $S = M^d \bmod n$ and sends the message and signature (S) to receiver.

Verifying : Receiver receives M and S and creates a copy of message

$$M = S^e \bmod n$$

Receiver compares value of M' with the value of M. If the two values r congruent, Receiver accepts the message.

$$M' \equiv M \pmod n \rightarrow S^e = M \pmod n \rightarrow M^{d*e} = M \pmod n$$

Key Generation Process

Key generation in the RSA digital signature scheme is exactly the same as key generation in the RSA cryptosystem. Here we create the public and private key of sender.

Attacks on RSA Digital Signature Scheme

There are some attacks that intruder (named as c) can apply to the RSA digital signature technique.

Chosen-Message Attack : If M_1 and M_2 are two messages, for sender and later creates a new message $M = M_1 * M_2$. Intruder can claim that sender's has signed M.

The attack is also referred to as multiplicative attack.

Key-Only Attack : Intruder (named as C) has access only to sender's public key. C intercepts the pair (M, S) and tries to create another message M' such that

$$M' = S^e \pmod n$$

This problem is difficult to solve as a discrete logarithm problem.

Known-Message Attack : Assume that intruder intercepted two message–signature pairs (M_1, S_1) and (M_2, S_2) that have been created using the same private key.

RSA Digital Signature Example

Let $p = 7$, $q = 11$

Then
$$n = p \times q = 77$$
$$\phi(n) = (p-1) \times (q-1)$$
$$= 60$$

Now choose public key e such that $e < \phi(n)$ and
$$\text{GCD}(e, \phi(n)) = 1$$

Let $e = 13$

Then private key d will be
$$(d \times e) \bmod \phi(n) = 1$$
$$(d \times 13) \bmod 60 = 1$$
$$d = 37$$

Signing process

Let $M = 2$

Then
$$S = 2^{37} \bmod 77$$
$$\boxed{S = 51}$$

Verifying Process
$$M = S^e \bmod n$$
$$M = 51^{13} \bmod 17$$
$$= 2$$

Since $S^e \bmod 77 = 2$; the signature is verified.

8.6.2. ElGamal Digital Signature Technique

In asymmetric key cryptosystem, we discussed ElGamal cryptosystem. If ElGamal is used for signing and verifying process, then it is called ElGamal digital signature technique.

Signing Process

The following are the steps to sign the message at sender side :

1. The Sender (A) generates a random number k, which is less than q. The sender has to choose a new r, each time to sign a new message.
2. The sender now calculates two signatures r and s as :
$$r = g^k \bmod p$$
$$s = (M - xr)\, k^{-1} \bmod (p-1)$$
3. The values of r and s are signatures of sender.
4. At last sender (A) sends M, r and s to receiver (B).

Verifying Process

The following are the steps to verify the message at receiver side:

1. The receiver checks to see if $0 < r < q$ and $0 < s < q$; else reject the signature.
2. The receiver calculates two verifier V_1 and V_2 using equation:
$$V_1 = g^M \bmod p$$
$$V_2 = y^r r^s \bmod p$$
3. Now receiver check, if V_1 is congruent to V_2, the message is accepted; otherwise, it is rejected.

Key Generation Process

The process of key generation can be understood in following steps :

1. The sender (A) select a very large prime number p.
2. Now sender calculate g using equation :
$g = h^{(p-1)/q} \bmod p$, where h is a number less than $(p-1)$ such that $h^{(p-1)/q} \bmod p$ is greater than 1.
3. The sender chose private key x, a number less than q.
4. Sender calculate public key y using equation
$y = g^x \bmod p$.
5. Sender public keys are (g, y, p, q).
6. Sender private key is x.

8.6.3. Digital Signature Standard (DSS)

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 was originally proposed in 1991 and revised later.

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.

NOTE ▶

DSS is a standard and DSA is actual algorithm based on DSS.

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.

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 *(PRa)* and a set of parameters known to a group of communicating principals. Consider this set to constitute a global public key *(PUG)*.The result is a signature consisting of two components, labelled 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 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 the knowledge of the private key, could have produced the valid signature.

The Digital Signature Algorithm (DSA)

In the signing process, two functions create two signatures; in the verifying process, the output of one function is compared to the first signature for verification.

Signing Process

The following are the steps to sign the message at sender side :

1. The Sender (A) generates a random number k, which is less than q. q is a 160-bit prime, chosen in key generation process. The sender has to choose a new r, each time to sign a new message.
2. The sender creates a digest of message $h(M)$, (usually SHA-1 is used).
3. The sender now calculates :

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

$$s = ((h(M) + xr)\, k^{-1}) \bmod q$$

4. The values of r and s are signatures of sender.
5. At last sender (A) sends M, r and s to receiver (B).

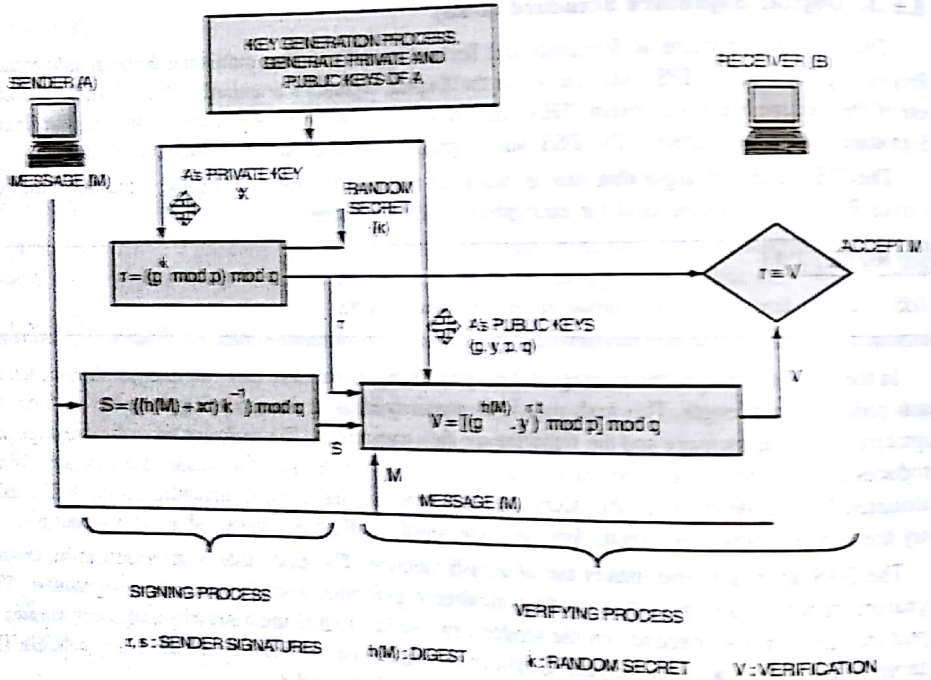


Fig. 8.5. *DSS Digital Signature Technique*

Verifying Process

The following are the steps to verify the message at receiver side:

1. The receiver checks to see if $0 < r < q$ and $0 < s < q$; else reject the signature.
2. Compute $s = S^{-1} \bmod q$.
3. The receiver calculates the verifier (V) using equation:

$$V = \left[\left(g^{h(M)} \cdot y^{r}\right) \bmod p\right] \bmod q$$

4. Now receiver check, if r is congruent to V, the message is accepted; otherwise, it is rejected.

Key Generation Process

The process of key generation can be understood in following steps:

1. The sender (A) chooses a prime p of length L. L is a multiple of 64 between 512 and 1024 bits in length. (i.e., $L = 512$ or 576 or 640 or.....).
2. The sender chooses a 160-bit prime q in such a way that it is a prime factor of $(p - 1)$.
3. Now sender calculate g using equation:
   $g = h^{(p-1)/q} \bmod p$, where h is a number less than $(p - 1)$ such that $h^{(p-1)/q} \bmod p$ is greater than 1.
4. The sender chose private key x, a number less than q.
5. Sender calculate public key y using equation

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

6. Sender public keys are (g, y, p, q).
7. Sender private key is x.

Example on DSA

Choose $p = 23$

Now choose q such that it is a prime factor of $(p - 1)$

Let $q = 11$

Calculate

$$g = h^{(p-1)/q} \bmod p$$

Let $h = 2$

$$\therefore \quad g = (2)^{(23-1)/11} \bmod 23$$

$$g = 4$$

Let private key $x = 3$

Then public key y will be

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

$$y = 4^3 \bmod 23$$

$$y = 18$$

Signing Process :

Let $k = 3 \quad (1 < k < q)$; Let $h(M) = 5 \quad (0 \le h(M < q))$

Now calculate signature of sender

$$r = (g^k \bmod p) \bmod q$$
$$r = (4^3 \bmod 23) \bmod 11$$
$$r = 7$$
$$S = ((h(M) + xr)K^{-1}) \bmod q$$
$$= ((5 + 3.7)3^{-1}) \bmod 11$$
$$= 5$$

So signature $= \{r, s\} = \{7, 5\}$

Verifying process :

$$t = S^{-1} \bmod q = 5^{-1} \bmod 11 = 9$$

Now

$$V = [(g^{h(M)} . y^r)^t \bmod p] \bmod q$$
$$= [(4^5 . 18^7)^9 \bmod 23] \bmod 11$$
$$= [(12 . 6)^9 \bmod 23] \bmod 11 = 7$$

Since $V = r = 7$ so that signature verified.

SUMMARY

- Using symmetric or asymmetric key cryptosystem we can achieve confidentiality. Digital signature is used to achieve integrity, authentication and non-repudiation.
- When there is no trust between sender and receiver, something more than authentication is required. The most suitable solution is use of digital signature.
- Digital signature is based on public key cryptosystem.
- In digital signature public and private key of sender is used. But in asymmetric key cryptosystem public and private key of receiver is used.
- We know that digital signature uses asymmetric key cryptosystem. Asymmetric key system is slow when message is long, so digital signature process also be. One solution of this drawback is to sign the message digest rather than signing message.
- Digital signature is used to provide Authentication, Integrity, Non-repudiation.
- Difference between RSA public key cryptosystem and RSA digital signature technique is the role of keys. As in digital signature the private and public key of the sender are used. The sender uses her own private key to sign the document; the receiver uses the sender's public key to verify it.
- DSS is a standard and DSA is actual algorithm based on DSS.

QUESTIONS WITH ANSWERS

Q. 1. Write the digital signature algorithm of DSS. Give reason behind the choice of various parameters of the algorithm. What is the implication, if same values of k is used to sign two different messages using DSA. **[UPTU 2010-11]**

Ans. For Answer of First part refer to Art. 8.6.3.

In DSA, the *k* value is *not* a nonce. In addition to being unique, the value must be *unpredictable* and *secret*. This makes it more like a random session key than a nonce. When an implementer gets this wrong, they expose the private key, often with only one or two signatures.

If *k* is predictable, there is a way to recover the private key from a single signature with straightforward algebra.

But what about the requirement that *k* be unique? Assume there's an email signing system that has a secure PRNG but it is rate-limited so two requests that come in quickly from the same

CHAPTER 9

Key Management and Distribution

One fundamental question arises as to why keys are necessary. (Why not just choose one encryption function and its corresponding decryption function?) Having functions associated with keys means that if the functions are revealed, then the scheme does not need to be redesigned, but simply the keys can be changed. In fact, keys are very critical to the functionality of cryptographic algorithms and it is sound cryptographic practice to change keys frequently.

Inside This Chapter

9.1. KEY MANAGEMENT

In a large network, there are many key pairs to be managed. Moreover, to guarantee security, the key should be changed frequently, perhaps for each communication session. Traditional cryptosystems (commonly known as symmetric systems or secret key systems) require the sender and the receiver to share a key known only to them. Knowledge of the key allows the decipherment of the cipher text, hence the need for secrecy. This system requires that **e = d**. But Public key cryptosystem uses two keys, one as public key and one as private key.

9.2. KEY DISTRIBUTION

In a two-party communication, the key must remain secret and must be known to both the sender and receiver before the transaction. Thus the two parties must take great care in exchanging the key so that an eavesdropper does not obtain it. There are several ways to distribute the keys:

- Public Announcement
- Publicly Available Directory

- Public key Authority
- Public key Certificates

1. Public Announcement

As shown in Fig. 9.1 in this approach participant can broadcast his/her public key or send it to any other participant also. Almost all PGP users append their public key to the messages send to public forums or internet mailing list. Though this is somewhat convenient but having a major weakness. Anyone can forge such public announcement means, any user can pretend to someone else and send public key to other users through broadcasting. Until such time the original user discovers the forgery and alerts through other participants, the forger is able to read all encrypted messages and use the forged keys for authentication.

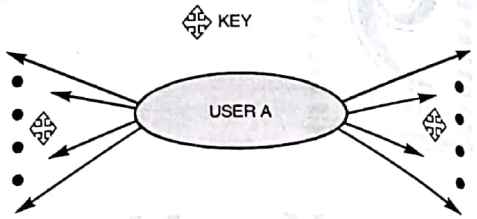


Fig. 9.1. *Public Key Announcement*

2. Publicly Available Directory

In this approach some sort of security could be achieved in key distribution scenario. Here as shown in Fig. 9.2, an authority exist which maintains a directory with an entry for each participant. Each participant registers its name and public key with the directory authority. Any time participants can replace their existing keys with new one, either because of the desire to replace a public key that has already been used for a large amount of data, or because its corresponding private key has been compromised in some way. Participants can access the directory after going through some authentication process which is mandatory to access the directory.

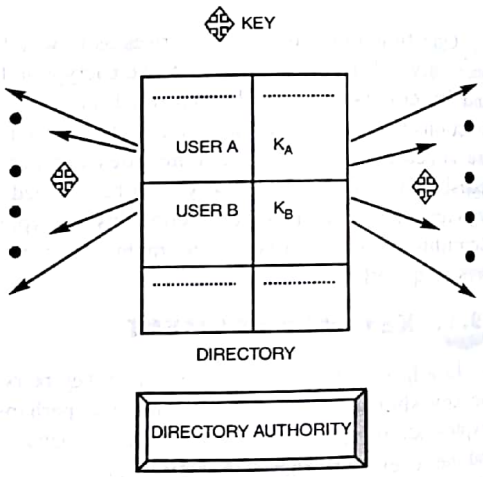


Fig. 9.2. *Publicly Available Directory*

As this scheme is more secure but having some vulnerabilities. If an opponent succeeds in obtaining or computing the private key of the directory authority, the opponent could authoritatively pass out counterfeit public keys and subsequently impersonate any participant and a message sent to the participant.

3. Public key Authorities

More better security for public key distribution can be achieved by providing tighter control over the distribution of keys by adding controls. The public key announcements can include a timestamp and be signed by an authority to prevent interception and modification of the response. If User *A* needs to know *B*'s public key, he/she can send a request to the center including B's name and timestamp.

The center responds with *B*'s public key, the original request, and the timestamp signed with the private key of center. *A* uses the public key of the center, known by all, to verify the timestamp. If the timestamp verified, he/she can extracts *B*'s public key as shown in Fig. 9.3.

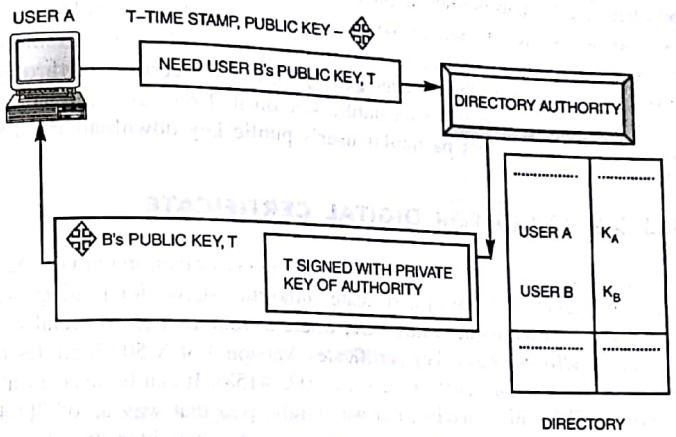


Fig. 9.3. *Publicly Key Authorities*

4. Public key Certificates

As previous approach can create a heavy load on the center, if the number of requests are large. The alternative is to create **public key certificate**. A **public key certificate** (also known as a **digital certificate** or **identity certificate**) is an electronic document which uses a digital signature to bind a public key with an identity — information such as the name of a person or an organization, their address, and so forth.

The certificate can be used to verify that a public key belongs to an individual.

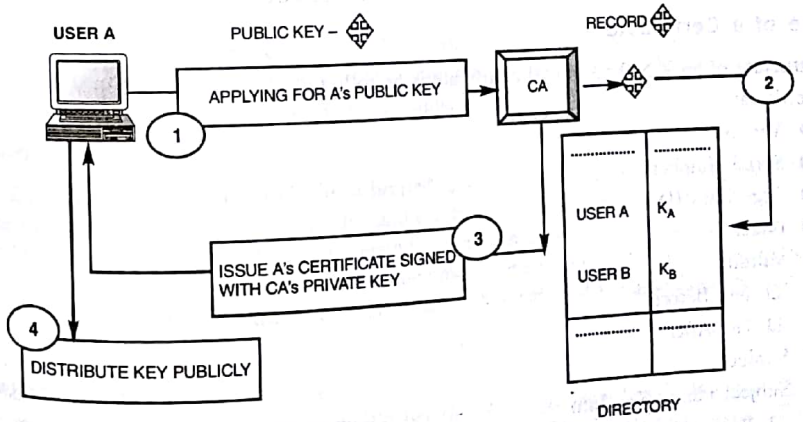


Fig. 9.4. *Publicly Key Certificates*

In this scheme as shown in Fig. 9.4, the signature will be of a certificate authority (CA). In a web of trust scheme, the signature is of either the user (a self-signed certificate) or other users ("endorsements"). In either case, the signatures on a certificate are attestations by the certificate signer that the identity information and the public key belong together.

For provable security this reliance on something external to the system has the consequence that any public key certification scheme has to rely on some special setup assumption, such as the existence of a certificate authority. After getting signed the certificate through CA which checks the identity of user and written its public key on it. Now, user can upload its signed certificate. Anyone who wants this particular user's public key download the certificate and extract its public key.

9.3. X.509 STANDARD FOR DIGITAL CERTIFICATE

X.509 was initially issued on July 3, 1988 and was begun in association with the X.500 standard. It assumes a strict hierarchical system of certificate authorities (CAs) for issuing the certificates. This contrasts with web of trust models, like PGP, where anyone (not just special CAs) may sign and thus attest to the validity of others' key certificates. Version 3 of X.509 includes the flexibility to support other topologies like bridges and meshes (RFC 4158). It can be used in a peer-to-peer, OpenPGP-like web of trustcitation needed, but was rarely used that way as of 2004. The X.500 system has only ever been implemented by sovereign nations for state identity information sharing treaty fulfillment purposes, and the IETF's Public-Key Infrastructure.

9.3.1. Certificates

In the X.509 system, a certification authority issues a certificate binding a public key to a particular *distinguished name* in the X.500 tradition, or to an alternative name such as an e-mail address or a DNS-entry.

An organization's trusted root certificates can be distributed to all employees so that they can use the company PKI system. X.509 also includes standards for certificate revocation list (CRL) implementations, an often neglected aspect of PKI systems. The IETF-approved way of checking a certificate's validity is the Online Certificate Status Protocol (OCSP).

Structure of a Certificate

The structure of an X.509 v3 digital certificate is as follows:

- Certificate
  - Version
  - Serial Number
  - Algorithm ID
  - Issuer
  - Validity
    - Not Before
    - Not After
  - Subject
  - Subject Public Key Info
    - Public Key Algorithm
    - Subject Public Key

  - Issuer Unique Identifier (optional)
  - Subject Unique Identifier (optional)
  - Extensions (optional)
    - ...
- Certificate Signature Algorithm
- Certificate Signature

Version

This identifies which version of the X.509 standard applies to this certificate, which affects what information can be specified in it. Thus far, three versions are defined.

Serial Number

The entity that created the certificate is responsible for assigning it a serial number to distinguish it from other certificates it issues. This information is used in numerous ways, for example when a certificate is revoked its serial number is placed in a Certificate Revocation List (CRL).

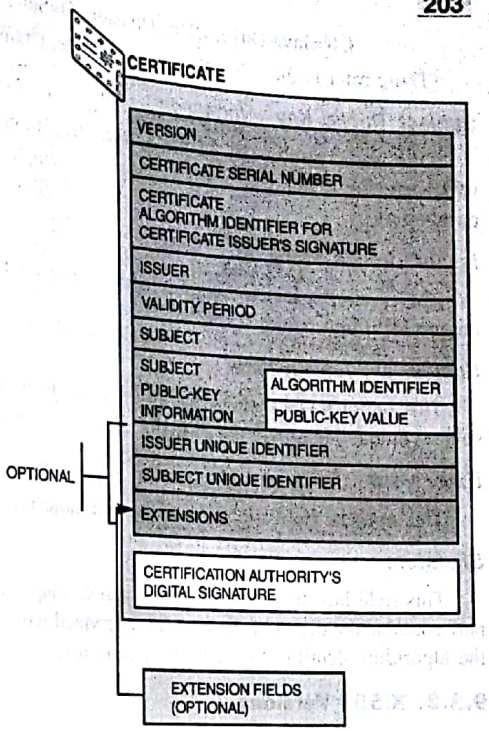


Fig. 9.5. *X.509 Certificate Format*

Signature Algorithm Identifier

This identifies the algorithm used by the CA to sign the certificate.

Issuer Name

The X.500 name of the entity that signed the certificate. This is normally a CA. Using this certificate implies trusting the entity that signed this certificate. (Note that in some cases, such as *root or top-level* CA certificates, the issuer signs its own certificate.)

Validity Period

Each certificate is valid only for a limited amount of time. This period is described by a start date and time and an end date and time, and can be as short as a few seconds or almost as long as a century. The validity period chosen depends on a number of factors, such as the strength of the private key used to sign the certificate or the amount one is willing to pay for a certificate. This is the expected period that entities can rely on the public value, if the associated private key has not been compromised.

Subject Name

The name of the entity whose public key the certificate identifies. This name uses the X.500 standard, so it is intended to be unique across the Internet. This is the Distinguished Name (DN) of the entity, for example,

CN=Java Duke, OU= Division, O=Sun Microsystems Inc, C=US

(These refer to the subject's Common Name, Organizational Unit, Organization, and Country.)

Subject Public Key Information

This is the public key of the entity being named, together with an algorithm identifier which specifies which public key crypto system this key belongs to and any associated key parameters.

Issuer Unique Identifier

This is an optional field which allows two users to have the same issuer field value, if the issuer unique identifiers are different.

Subject Unique Identifier

This optional field allows two different subjects to have the same subject field value, if the subject unique identifiers are different.

Extensions

This optional field allows issuers to add more private information to the certificate.

Signature

This field has three parts. The first part contains all other fields in the certificate. The second part contains the digest of the first part encrypted with the CA's public key. The third part contains the algorithm identifier to create the second part.

9.3.2. X.509 Versions

X.509 Version 1 has been available since 1988, is widely deployed, and is the most generic.

X.509 Version 2 introduced the concept of subject and issuer unique identifiers to handle the possibility of reuse of subject and/or issuer names over time. Most certificate profile documents strongly recommend that names not be reused, and that certificates should not make use of unique identifiers. Version 2 certificates are not widely used.

X.509 Version 3 is the most recent (1996) and supports the notion of extensions, whereby anyone can define an extension and include it in the certificate. Some common extensions in use today are: *KeyUsage* (limits the use of the keys to particular purposes such as "signing-only") and *AlternativeNames* (allows other identities to also be associated with this public key, *e.g.*, DNS names, Email addresses, IP addresses). Extensions can be marked *critical* to indicate that the extension should be checked and enforced/used. For example, if a certificate has the KeyUsage extension marked critical and set to "keyCertSign" then if this certificate is presented during SSL communication, it should be rejected, as the certificate extension indicates that the associated private key should only be used for signing certificates and not for SSL use.

9.3.3. Certificate Renewal

Each certificate has a period of validity. If there is no problem with the certificate, the CA issues a new certificate before the old one expires. The process is like the renewal of credit cards by a credit card company, the credit card holder normally receives a renewed credit card before the one expires.

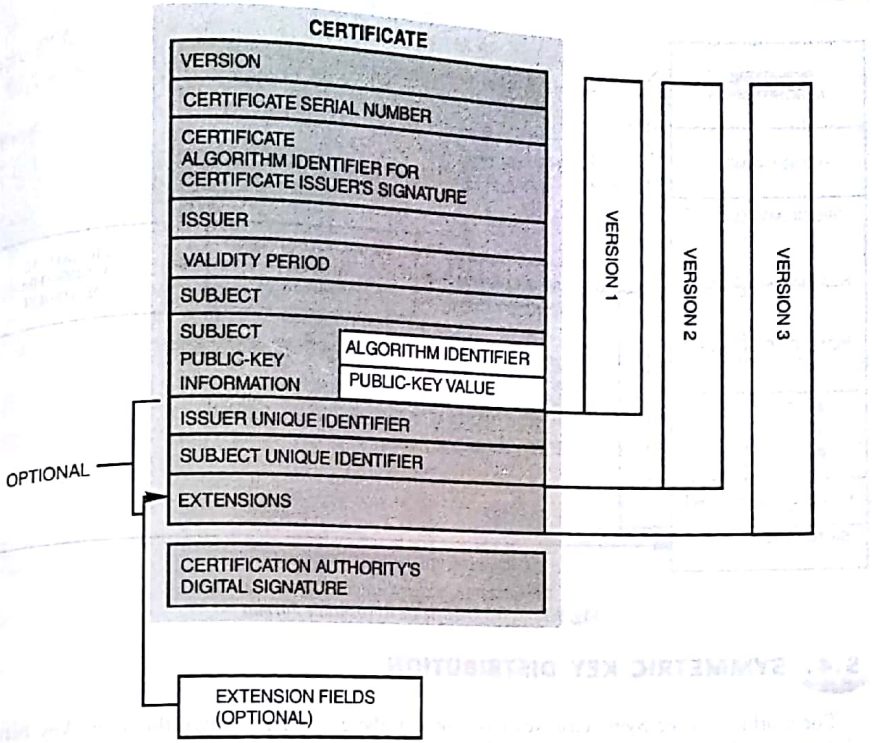


Fig. 9.6. *X.509 Version*

9.3.4. Certificate Revocation

Here below some cases are given in which certificate can be revoked before it expires.

(*a*) The user's private key might have been comprised.

(*b*) The CA is no longer willing to certify the user.

(*c*) The CA's private key, from which he can verify certificates, may have been comprised.

In all cases, the revocation is done periodically by issuing a Certificate Revocation list (CRL). When a user wants to use a certificate, he/she first needs to check the directory of the corresponding CA for the last revocation list.

A CRL has following fields:

(*a*) Signature Algorithm ID

(*b*) Issuer Name

(*c*) **This Update Date :** Release date of List.

(*d*) **Next Update Date :** Next date when the list will be released.

(*e*) **Revoked Certificate :** Repeated list of all unexpired certificates that have been revoked which contain User Certificate serial number and revoked date.

(*f*) Signature

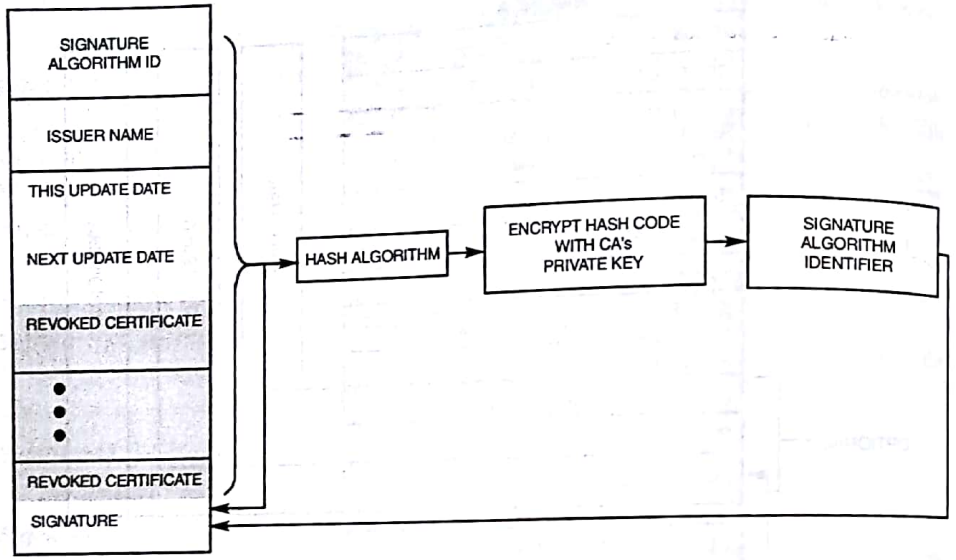


Fig. 9.7. *X.509 Certificate Revocation Format*

9.4. SYMMETRIC KEY DISTRIBUTION

For working in the Symmetric key environment, there is need to share the same key between two parties and the keys must be protected from access of others. Let's say we have a large number of people, processes, or systems that want to communicate with one another in a secure fashion. This group of people/processes/systems is not static, meaning that the individual entities may join or leave the group at any time. A simple-minded solution to this problem would consist of each party physically exchanging an encryption key with every one of the other parties. Subsequently, any two parties would be able to establish a secure communication link using the encryption key they possess for each other. This approach is obviously not feasible for large groups of people/processes/systems, especially when group membership is ever changing.

A more efficient alternative consists of providing every group member with a single key for securely communicate with a key distribution center (KDC). This key would be called a master key. When A wants to establish a secure communication link with B, A requests a session key from KDC for communicating with B.

9.4.1. Key Distribution Center (KDC)

A type of key center that implements a key-distribution protocol to provide keys (usually, session keys) to two (or more) entities that wish to communicate securely.

A secret key is established between the KDC and each member. The process of communication with KDC is as follows:

1. User A sends a request to the KDC stating that he needs a session secret key between him and user B.

2. The KDC informs user B about the A's request.
3. If B agreed, a session key is created between the two users.

This established secret key is used to authenticate user A and user B to the KDC.

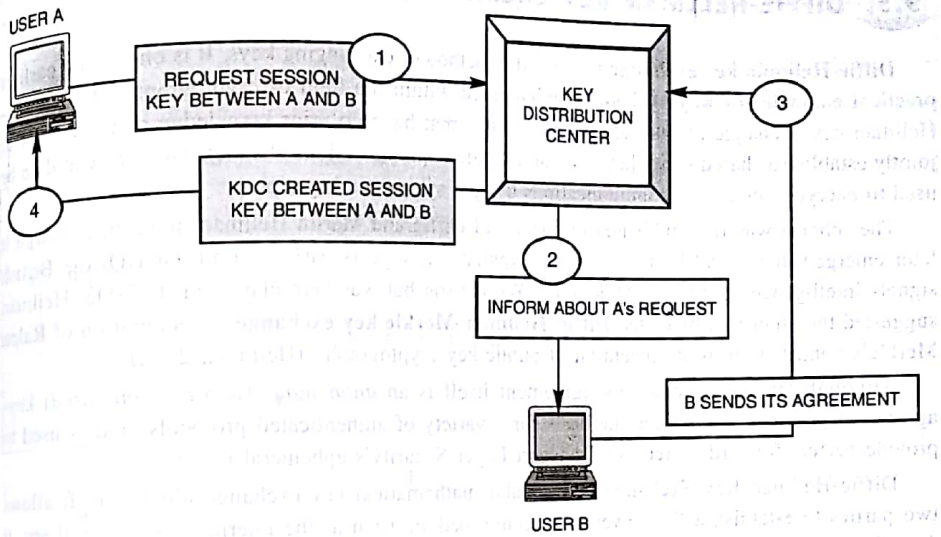


Fig. 9.8. *Key Distribution Center*

This approach must address the following issues:

1. Assuming that A is the initiator of a session-key request to KDC, when A receives a response from KDC, how can A be sure that the sending party for the response is indeed the KDC?
2. Assuming that A is the initiator of a communication link with B, how does B know that some other party is not masquerading as A?
3. How does A know that the response received from B is indeed from B and not from someone else masquerading as B?
4. What should be the lifetime of the session key acquired by A for communicating with B?

Flat Multiple KDCs

When the number of people using a KDC increases, the system become bottleneck. To solve this, we need to have multiple KDC's. A domain consist one or multiple KDC's. No, if A wants to send a confidential message to B, who belongs to another domain. A contacts his KDC, which in turn contacts the KDC in B's domain. The two KDC's can create a secret key between A and B. This scenario is called flat multiple KDCs if all KDC's are at same level.

Hierarchical Multiple KDCs

The concept of flat multiple KDC's can be extended to a hierarchical system of KDC's with one or more KDC's at the top of the hierarchy. When *A* needs to communicate with *B* who lives in another country, A sends its request to local KDC and local KDC further relays the message to

national KDC which passes the request to an International KDC. The request is then relayed all the way down to the local KDC where *B* lives.

9.5. DIFFIE-HELLMAN KEY EXCHANGE

Diffie-Hellman key exchange is a specific method of exchanging keys. It is one of the earliest practical examples of Key exchange implemented within the field of cryptography. The Diffie-Hellman key exchange method 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.

The scheme was first published by Whitfield Diffie and Martin Hellman in 1976, although it later emerged that it had been separately invented a few years earlier within GCHQ, the British signals intelligence agency, by Malcolm J. Williamson but was kept classified. In 2002, Hellman suggested the algorithm be called **Diffie–Hellman–Merkle key exchange** in recognition of Ralph Merkle's contribution to the invention of public-key cryptography (Hellman, 2002).

Although Diffie–Hellman key agreement itself is an *anonymous* (non-*authenticated*) key-agreement protocol, it provides the basis for a variety of authenticated protocols, and is used to provide perfect forward secrecy in Transport Layer Security's ephemeral modes.

Diffie-Hellman Key Exchange is a popular mathematical key exchange algorithm. It allows two parties to establish a 'key' over an insecure medium such as the internet. As you will see, it doesn't matter whether the intercepting party captures each piece of transmitted information, they will not be able to break the key in any way, other than the usual brute force method.

Process

1. Alice and Bob, using insecure communication, agree on a huge prime p and a generator g of order $p - 1$.
2. Alice chooses some large random integer $x_A < p$ and keeps it secret. Likewise Bob chooses $x_B < p$ and keeps it secret. These are their "private keys".
3. Alice computes her "public key" $y_A \equiv g^{x}{}_{A} \pmod{p}$ and sends it to Bob using insecure communication. Bob computes his public key $y_B \equiv g^{x}{}_{B}$ and sends it to Alice.

   Here $0 < y_A < p$, $0 < y_B < p$.

   As already mentioned, sending these public keys with insecure communication is safe because it would be too hard for someone to compute x_A from y_A or x_B from y_B.
4. Alice computes $z_A \equiv y_B{}^{x}{}_{A} \pmod{p}$ and Bob computes $z_B \equiv y_A{}^{x}{}_{B} \pmod{p}$. Here $z_A < p$, $z_B < p$.

But $z_A = z_B$, since $z_A \equiv y_B{}^{x}{}_{A} \equiv (g^{x}{}_{B})^{x}{}_{A} = g^{(x}{}_{A}{}^{x}{}_{B)} \pmod{p}$ and similarly $z_B \equiv (g^{x}{}_{A})^{x}{}_{B} = g^{(x}{}_{A}{}^{x}{}_{B)} \pmod{p}$. So this value is their **shared secret key**. They can use it to encrypt and decrypt the rest of their communication by some faster method.

In this calculation, notice that the step $y_B{}^{x}{}_{A} \equiv (g^{x}{}_{B})^{x}{}_{A}$ involved replacing $g^{x}{}_{B}$ by its remainder y_B, (in the reverse direction) so we were really using the "as often as you want" principle :

Alice				Bob		
Secret	Public	Calculates	Sends	Calculates	Public	Secret
a	p, g		$p, g \rightarrow$			
a	p, g, A	$g^a \bmod p = A$	$A \rightarrow$			b
a	p, g, A		$\rightarrow B$		p, g	b
a, s	p, g, A, B	$B^a \bmod p = s$		$g^b \bmod p = B$	p, g, A, B	b
				$A^b \bmod p = s$	p, g, A, B	b, s

Fig. 9.9. *Diffie Hellman Key Exchange*

Example:

1. Alice and Bob agree to use a prime number $p = 23$ and base $g = 5$.
2. Alice chooses a secret integer $a = 6$, then sends Bob $A = g^a \bmod p$
   - $A = 5^6 \bmod 23$
   - $A = 15{,}625 \bmod 23$
   - $A = 8$
3. Bob chooses a secret integer $b = 15$, then sends Alice $B = g^b \bmod p$
   - $B = 5^{15} \bmod 23$
   - $B = 30, 517, 578, 125 \bmod 23$
   - $B = 19$
4. Alice computes $s = B^a \bmod p$
   - $s = 19^6 \bmod 23$
   - $s = 47{,}045{,}881 \bmod 23$
   - $s = 2$
5. Bob computes $s = A^b \bmod p$
   - $s = 8^{15} \bmod 23$
   - $s = 35{,}184{,}372{,}088{,}832 \bmod 23$
   - $s = 2$
6. Alice and Bob now share a secret: $s = 2$. This is because 6*15 is the same as 15*6. So somebody who had known both these private integers might also have calculated s as follows:
   - $s = 5^{6*15} \bmod 23$
   - $s = 5^{15*6} \bmod 23$
   - $s = 5^{90} \bmod 23$
   - $s = 807, 793, 566, 946, 316, 088, 741, 610, 050, 849, 573, 099, 185, 363, 389, 551, 639, 556, 884, 765, 625 \bmod 23$
   - $s = 2$

Example:

1. Here $p = 11$, $g = 2$, $x_A = 9$, $x_B = 4$. So $y_A = 2^{x}{}_{A} = 2^9 \pmod{11}$.
2. You can find this most easily by finding $2^2 = 4$, $2^4 = 4^2 = 16 \equiv \pmod{11}$, $2^8 = (2^4)^2 \equiv 5^2 = 25 \equiv \pmod{11}$, and finally $2^9 = 2 \times 2^8 \equiv 2 \times 3 = 6$. So $y_A = 6$.
3. Similary, $2^{x}{}_{B} = 2^4 = 16 \equiv \pmod{11}$, so $y_B = 5$.

4. The secret shared key z_A is the remainder of $y_B^{x_A} = 5^9 \pmod{11}$. So find $5^2 = 25 \equiv \pmod{11}$, $5^4 = (5^2)^2\ 3^2 = 9 \pmod{11}$. $5^8 = (5^4)^2 \equiv 9^2 = 81 \equiv \pmod{11}$. $5^9 = 5 \times 5^8 \equiv 5 \times 4 = 20 \equiv \pmod{11}$. As a check, z_B is the remainder of $y_A^{x_B} = 6^4 \pmod{11}$. $6^2 = 36 \equiv \pmod{11}$ so $6^4 = (6^2)^2 \equiv 3^2 = 9 \pmod{11}$, which checks. So $z_A = z_B = 9$.

Attacks on Diffie Hellmann

Man-in Middle Attack

The Diffie-Hellman key exchange is vulnerable to a man-in-the-middle attack.

1. In this attack, an opponent Carol intercepts Alice's public value and sends her own public value to Bob.
2. When Bob transmits his public value, Carol substitutes it with her own and sends it to Alice.
3. Carol and Alice thus agree on one shared key and Carol and Bob agree on another shared key.
4. After this exchange, Carol simply decrypts any messages sent out by Alice or Bob, and then reads and possibly modifies them before re-encrypting with the appropriate key and transmitting them to the other party.

This vulnerability is present because Diffie-Hellman key exchange does not authenticate the participants. Possible solutions include the use of digital signatures and other protocol variants.

The authenticated Diffie-Hellman key agreement protocol, or Station-to-Station (STS) protocol, was developed by Diffie, van Oorschot, and Wiener in 1992 to defeat the man-in-the-middle attack on the Diffie-Hellman key agreement protocol. The immunity is achieved by allowing the two parties to authenticate themselves to each other by the use of digital signatures and public-key certificates which are digital documents attesting to the binding of a public key to an individual or other entity.

9.6. PUBLIC KEY INFRASTRUCTURE (PKI)

A PKI (public key infrastructure) enables users of a basically unsecure public network such as the Internet to securely and privately exchange data and money through the use of a public and a private cryptographic key pair that is obtained and shared through a trusted authority. The public key infrastructure provides for a digital certificate that can identify an individual or an organization and directory services that can store and, when necessary, revoke the certificates. Although the components of a PKI are generally understood, a number of different vendor approaches and services are emerging. Meanwhile, an Internet standard for PKI is being worked on.

The public key infrastructure assumes the use of *public key cryptography*, which is the most common method on the Internet for authenticating a message sender or encrypting a message. Traditional cryptography has usually involved the creation and sharing of a secret key for the encryption and decryption of messages. This secret or private key system has the significant flaw that if the key is discovered or intercepted by someone else, messages can easily be decrypted. For this reason, public key cryptography and the public key infrastructure is the preferred approach on the Internet. A public key infrastructure consists of:

- A certificate authority (CA) that issues and verifies digital certificate. A certificate includes the public key or information about the public key

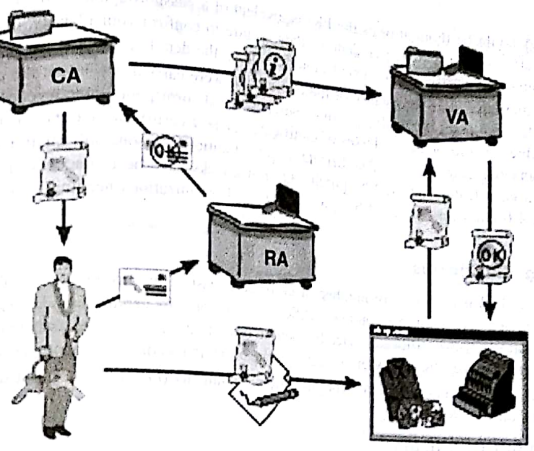


Fig. 9.10. *Components of a PKI*

- A registration authority (RA) that acts as the verifier for the certificate authority before a digital certificate is issued to a requestor
- One or more directories where the certificates (with their public keys) are held
- A certificate management system

A PKI (public key infrastructure) is created by combining a number of services and technologies:

(1) Certification Authority (CA)

A CA issues and verifies certificates (see above). The CA takes responsibility for identifying (to a stated extent) the correctness of the identity of the person asking for a certificate to be issued, and ensures that the information contained within the certificate is correct and digitally signs it.

(a) Generating key pairs

The CA may generate a public key and a private key (a key pair) or the person applying for a certificate may have to generate their own key pair and send a signed request containing their public key to the CA for validation. The person applying for a certificate may prefer to generate their own key pair so as to ensure that the private key never leaves their control and as a result is less likely to be available to anyone else.

(b) Issuing digital certificates

Unless you generate your own certificate (some applications software will enable you do this) you will generally have to purchase one from a CA. Before a CA issues you with a certificate they will make various checks to prove that you are who you say you are.

The CA could be thought of as the PKI equivalent of a passport agency—the CA issues you a certificate after you provide the credentials they require to confirm your identity, and then the CA signs (stamps) the certificate to prevent modification of the details contained in the certificate.

A CA may also state the quality of the checks that were carried out before the certificate was issued. Different classes of certificate can be purchased that correspond to the level of checks made. There are three or four general classes of certificate: Class 1 certificates can be easily acquired by supplying an email address, Class 2 certificates require additional personal information to be supplied, and Class 3 certificates can only be purchased after checks have been made as to the requestors identity. A 4th class may be used by governments and organizations needing very high levels of checking.

(c) Using Certificates

An individual may have any number of certificates issued by any number of CAs. Different Web applications may insist that you use certificates issued only by certain CAs. For example, a bank may insist that you use a certificate issued by them in order to use their services, whereas a public Web site may accept any certificate you offer (just as some allow free choice of ID and password).

The CA can be a unit within your organization, a company (*i.e.*, a bank or a post office), or an independent entity (VeriSign).

(d) Verifying Certificates

The public key certificate is signed by the CA to prevent its modification or falsification. This signature is also used when checking that the public key is still valid. The signature is validated against a list of 'Root CAs' contained within various 'PKI aware' applications (*e.g.*, your browser). Some CA certificates are called 'Root Certificates' as they form the root of all certificate validation. Certificate validation occurs automatically using the appropriate public certificate contained within the root CA list.

NOTE

PGP (Pretty Good Privacy) users normally act as their own issuing authority, so you accept their certificate on the basis that they are who they say they are without further verification. This method is called the 'Web of trust' because it is based upon people you trust rather than liability by contract.

(2) Revocation

Where a system relies upon publishing certificates so that people are able to communicate with each other, there has to be a system for letting people know when certificates are no longer valid. It can be done in one of two ways. Certificates can be deleted from the Directory or database in which they should be found. As a result, any attempt to find them to check that they still exist will fail and anyone looking for them would know that they have been revoked.

There are two problems with this approach. The first is that a denial of service attack on the Directory or database might create the appearance of a failed certificate. The second is that the Directory was designed to optimize the time to read information, so deleting information is normally avoided, as is updating. Also, deleting the record does not tell the person asking for the information why it is not there, and they may need to know why and when it was removed.

As a result, a system of revocation lists has been developed that exists outside the Directory or database. This is a list of certificates that are no longer valid (for whatever reason), equivalent to a lost or stolen ATM card list. There are currently two different methods for checking for certificate

revocation—'CRL' or 'OCSP'. Revocation lists may be publicly available even when the matching Directory or database is not. This is because certificates may have been distributed for use beyond the private network of the organization involved.

(3) Registration Authority (RA)

A CA may use a third-party Registration Authority (RA) to perform the necessary checks on the person or company requesting the certificate to ensure that they are who they say they are. That RA may appear to the certificate requestor as a CA, but they do not actually sign the certificate that is issued.

(4) Certificate Publishing Methods

One of the fundamentals of PKI systems is the need to publish certificates so that users can find them. (You must be able to get hold of the public encryption key for the recipient of encrypted information.) There are two ways of achieving this. One is to publish certificates in the equivalent of an electronic telephone directory. The other is to send your certificate out to those people you think might need it by one means or another. The commonest approaches are listed below.

(a) Directories

Directories are databases that are X.500/LDAP-compliant. This means that they contain certificates in the X.509 format, and that they provide specific search facilities as specified in the LDAP standards published by the IETF. Directories may be made publicly available or they may be private to a specific organization *i.e.*, a company may have its own directory where it holds the certificates for its users and only its users can access this directory. A Directory is kept private when it contains information that the owner does not wish to be publicly available. Public directories on the other hand can be read by anyone with access to them.

(b) Databases

A database can be configured to accept X.509 format certificates. This may be done for private systems where the search methods for locating certificates do not follow the LDAP structure. Because it is essentially proprietary, this method is not used for public systems.

(c) Email, Floppy Discs etc.

Certificates may be sent within an e-mail so that the recipient can add them to their own collection on their server or desktop, depending upon the way their security systems have been configured. They may also be put onto floppy discs, or any other medium.

(5) Certificate Management System

This term refers to the management system through which certificates are published, temporarily or permanently suspended, renewed or revoked. Certificate management systems do not normally delete certificates because it may be necessary to prove their status at a point in time, perhaps for legal reasons. A CA (and perhaps an RA) will run certificate management systems to be able to keep track of their responsibilities and liabilities.

9.7. WEB OF TRUST

An alternative approach to the problem of public authentication of public key information is the web of trust scheme, which uses self-signed certificates and third party attestations of those

certificates. The singular term Web of Trust does not imply the existence of a single web of trust, or common point of trust, but rather one of any number of potentially disjoint "webs of trust". Examples of implementations of this approach are PGP (Pretty Good Privacy) and GnuPG (an implementation of OpenPGP, the standardized specification of PGP). Because PGP and implementations allow the use of e-mail digital signatures for self-publication of public key information, it is relatively easy to implement one's own Web of Trust. One of the benefits of the Web of Trust, such as in PGP, is that it can interoperate with a PKI CA fully-trusted by all parties in a domain (such as an internal CA in a company) that is willing to guarantee certificates, as a trusted introducer. Only if the "web of trust" is completely trusted, and because of the nature of a web of trust, trusting one certificate is granting trust to all the certificates in that web. A PKI is only as valuable as the standards and practices that control the issuance of certificates and including PGP or a personally instituted web of trust could significantly degrade the trustability of that enterprise's or domain's implementation of PKI.

The web of trust concept was first put forth by PGP creator Phil Zimmermann in 1992 in the manual for PGP version 2.0:

As time goes on, you will accumulate keys from other people that you may want to designate as trusted introducers. Everyone else will each choose their own trusted introducers. And everyone will gradually accumulate and distribute with their key a collection of certifying signatures from other people, with the expectation that anyone receiving it will trust at least one or two of the signatures. This will cause the emergence of a decentralized fault-tolerant web of confidence for all public keys.

SUMMARY

- In a two-party communication the key must remain secret and must be known to both the sender and receiver before the transaction.
- There are several ways for distribution of key: Public Announcement, Publicly Available Directory, Public key Authority and Public key Certificates.
- In the X.509 system, a certification authority issues a certificate binding a public key to a particular user.
- X.509 is available in three versions.
- Each certificate has a period of validity. If there is no problem with the certificate, the CA issues a new certificate before the old one expires.
- Revocation is done periodically by issuing a Certificate Revocation list (CRL).
- In Symmetric key environment, there is need to share the same key between two parties and the keys must be protected from access of others.
- KDC is a type of key center that implements a key-distribution protocol to provide keys (usually session keys) to two (or more) entities that wish to communicate securely.
- Diffie–Hellman key exchange is a specific method of exchanging keys.
- A PKI (public key infrastructure) enables users of a basically unsecure public network such as the Internet to securely and privately exchange data and money through the use of a public and a private cryptographic key pair that is obtained and shared through a trusted authority.
- The public key infrastructure provides for a digital certificate that can identify an individual or an organization and directory services that can store and, when necessary, revoke the certificates.
- An alternative approach to the problem of public authentication of public key information is the web of trust scheme, which uses self-signed certificates and third party attestations of those certificates.

CHAPTER 10

Authentication Applications and Email Security

10.1. INTRODUCTION

Inside This Chapter

Authentication is the act of establishing identity via the presentation of information that allows the verifier to know the presenter is who or what it claims. This identity could be any number of things, including:

- People
- Systems
- Applications
- Messages

Why would one want to verify an identity in the first place? Hopefully, most people reading this recognize that as sarcastic humor. If not, here are a few common reasons:

- To control access to a system or application
- To bind some sensitive data to an individual, such as for encryption
- To establish trust between multiple parties to form some interaction with them
- To assure that a piece of information is genuine

Within an application, one or all of these aspects may apply. Kerberos is an authentication protocol, and at the same time a KDC, that has become very popular.

10.2. KERBEROS

MIT developed Kerberos to protect network services provided by Project Athena. The protocol was named after the Greek mythological character *Kerberos* (or *Cerberus*), known in Greek mythology as being the monstrous three-headed guard dog of Hades. Several versions of the protocol exist; versions 1–3 occurred only internally at MIT.

Steve Miller and Clifford Neumann, the primary designers of Kerberos version 4, published that version in the late 1980s, although they had targeted it primarily for Project Athena.

Version 5, designed by John Kohl and Clifford Neumann, appeared as RFC 1510 in 1993 (made obsolete by RFC 4120 in 2005), with the intention of overcoming the limitations and security problems of version 4.

Kerberos uses as its basis the symmetric 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 security of the protocol relies heavily on participants maintaining loosely synchronized time and on short-lived assertions of authenticity called *Kerberos tickets*.

10.2.1. Description

The following is an intuitive description. The client authenticates itself to the Authentication Server and receives a ticket (All tickets are time-stamped). It then contacts the Ticket Granting Server, and using the ticket it demonstrates its identity and asks for a service. If the client is eligible for the service, then the Ticket Granting Server sends another ticket to the client. The client then contacts the Service Server, by using this ticket which proves that it has been approved to receive the service.

A simplified and more detailed description of the protocol follows. The following abbreviations are used :

- AS = Authentication Server
- SS = Service Server
- TGS = Ticket Granting Server
- TGT = Ticket Granting Ticket

The client authenticates to the AS **once** using a long-term *shared secret* (*e.g.*, a password) and receives a TGT from the AS. Later, when the client wants to contact some SS, it can reuse this ticket to get additional tickets from TGS, for SS, without resorting to using the shared secret. These tickets can be used to prove authentication to SS.

The phases are detailed below.

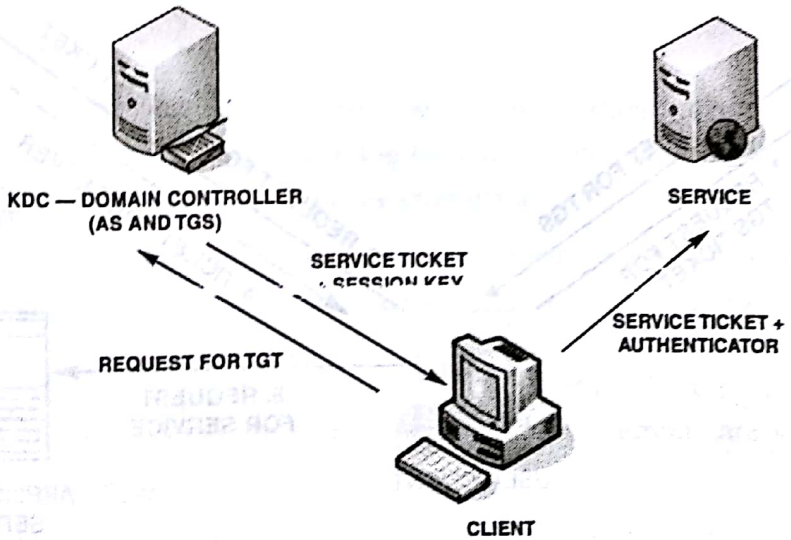


Fig. 10.1. ***Client Authentication***

User Client-Based Logon

1. A user enters a username and password on the client machine.
2. The client performs a one-way function (hash usually) on the entered password, and this becomes the secret key of the client/user.

10.2.2. Kerberos Protocol

1. The client sends a cleartext message of the user ID to the AS requesting services on behalf of the user.

NOTE

Neither the secret key nor the password is sent to the AS. The AS generates the secret key by hashing the password of the user found at the database (e.g., Active Directory in Windows Server.

2. The AS checks to see if the client is in its database. If it is, the AS sends back the following two messages to the client:
   - Message A: *Client/TGS Session Key* encrypted using the secret key of the client/user.
   - Message B: *Ticket-to Get-Ticket* (which includes the client ID, client network address, ticket validity period, and the *client/TGS session key*) encrypted using the secret key of the TGS.
3. Once the client receives messages A and B, it attempts to decrypt message A with the secret key generated from the password entered by the user. If the user entered password does not match the password in the AS database, the client's secret key will be different and thus unable to decrypt message A. With a valid password and secret key the client decrypts message A to obtain the *Client/TGS Session Key*. This session key is used for further communications with the TGS. (Note: The client cannot decrypt Message B, as it is encrypted using TGS's secret key.) At this point, the client has enough information to authenticate itself to the TGS.

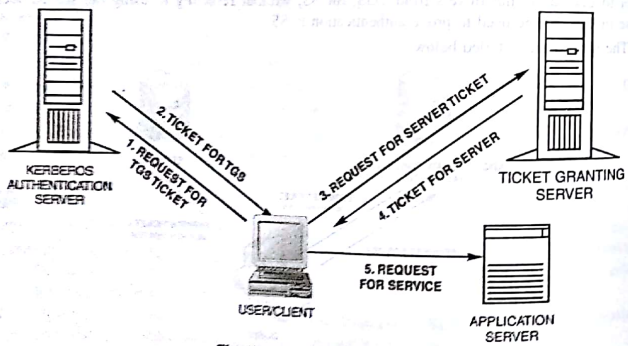


Fig. 10.2. Kerberos Description

Client Service Authorization

1. When requesting services, the client sends the following two messages to the TGS:
   - Message C: Composed of the TGT from message B and the ID of the requested service.
   - Message D: Authenticator (which is composed of the client ID and the timestamp), encrypted using the *Client/TGS Session Key*.
2. Upon receiving messages C and D, the TGS retrieves message B out of message C. It decrypts message B using the TGS secret key. This gives it the "client/TGS session key". Using this key, the TGS decrypts message D (Authenticator) and sends the following two messages to the client:
   - Message E: *Client-to-server ticket* (which includes the client ID, client network address, validity period and *Client/Server Session Key*) encrypted using the service server's secret key.
   - Message F: *Client/server session key* encrypted with the *Client/TGS Session Key*.

Client Service Request

1. Upon receiving messages E and F from TGS, the client has enough information to authenticate itself to the SS. The client connects to the SS and sends the following two messages:
   - Message E from the previous step (the *client-to-server ticket*, encrypted using service's secret key).
   - Message G: A new Authenticator, which includes the client ID, timestamp and is encrypted using *client/server session key*.
2. The SS decrypts the ticket using its own secret key to retrieve the *Client/Server Session Key*. Using the sessions key, SS decrypts the Authenticator and sends the following message to the client to confirm its true identity and willingness to serve the client:
   - Message H: The timestamp found in client's Authenticator plus 1, encrypted using the *Client/Server Session Key*.
3. The client decrypts the confirmation using the *Client/Server Session Key* and checks whether the timestamp is correctly updated. If so, then the client can trust the server and can start issuing service requests to the server.
4. The server provides the requested services to the client.

10.2.3. Drawbacks

- **Single point of failure :** It requires continuous availability of a central server. When the Kerberos server is down, no one can log in. This can be mitigated by using multiple Kerberos servers and fallback authentication mechanisms.
- Kerberos has strict time requirements, which means the clocks of the involved hosts must be synchronized within configured limits. The tickets have a time availability period and if the host clock is not synchronized with the Kerberos server clock, the authentication will fail. The default configuration per MIT requires that clock times are no more than five minutes apart. In practice Network Time Protocol daemons are usually used to keep the host clocks synchronized.

- The administration protocol is not standardized and differs between server implementations. Password changes are described in RFC 3244.

Since all authentication is controlled by a centralized KDC, compromise of this authentication infrastructure will allow an attacker to impersonate any user.

10.2.4. Kerberos Versions

The version 5 of Kerberos is to address the limitation of version 4 in following two areas:

1. Environmental Shortcoming
2. Technical Deficiencies

1. Environmental Shortcoming

As Kerberos version 4 was developed for Project Athena environment, it is not fully address the need of general purpose which results in some environmental shortcomings :

(a) Encryption System dependence

Kerberos Version 4 need DES for encryption while in Version 5, the ciphertext is tagged with an encryption type identifies so that any encryption technique can be used.

(b) Internet Protocol dependence

Kerberos Version 4 require the use of IP addresses. Other address types, such as ISO network address are accommodated while in Version 5 Network address are tagged with type and length, allowing any network address type to be used.

(c) Message byte Ordering

In version 4, the sender of message employ a byte ordering of its own choosing and tags the message to indicate byte in lowest address or most significant byte in lowest address while in version 5 all the messages are defined using Abstract System Notation one (ASN-1) and Basic Encoding Rule (BER) which provides an unambiguous byte ordering.

(d) Ticket Life Time

Life time values in version 4 s are encoded in an 8-bit quantity in units of five minutes. Thus, the maximum life time that can be expressed to $2^8 \times 5 = 1280$ min or over 21 hours while version 5 tickets include an explicit start time and end time, allowing tickets with arbitrary life times.

(e) Authentication Forwarding

Version 4 of Kerberos does not allow credentials issued to one client to be forwarded to some other host and used by some other client while version 5 provide this capability.

(f) Inter-realm Authentication

In version 4, interoperability among N realms requires on the order of N^2 Kerberos to Kerberos relationship while version 5 supports a method that require fewer relationship.

2. Technical Deficiencies

Here, some technical deficiencies of Version4 are given and version 5 attempts to address these.

(a) Double Encryption

In Version 4 ticket provide to clients are encrypted twice. Once with the secret key of TS (Target Server) and again with the secret key known to the client. The second encryption is not necessary and it is computationally wasteful.

(b) PCBC Encryption

In version 4 Encryption makes use of non-standard mode of DES known as propagating block chaining (PCBC). It provides an integrity check as part of the encryption operation. Version 5 provides explicit integrity mechanism, allowing the standard CBC mode to be used for encryption and a checksum or hash code is attached to the message prior to CBC encryption.

(c) Session Keys

Each ticket includes a session key that is used by the client to encrypt the authenticator sent to the service associated with client. However same ticket may be used repeatedly to gain service, there is risk of replay messages from an old session. In version 5, it is possible for a client and server to negotiate a subsession key, which is to be used only for that one connection. A new access by the client would result in the use of a new session key.

(d) Password Attack

Both versions are vulnerable to a password attack. The message from authentication server to the client includes material encrypted with a key based on the client's password.

10.3. E-MAIL ARCHITECTURE

E-mail, short form of electronic mail and often abbreviated to e-mail, email or simply mail, is a store and forward method of composing, sending and storing messages over electronic communication systems. The term "e-mail" applies both to the Internet e-mail system based on the Simple Mail Transfer Protocol (SMTP) and to X.400 systems, and to intranet systems allowing users within one organization to e-mail each other. Intranets may use the Internet protocols or X.400 protocols for internal e-mail service supporting workgroup collaboration. E-mail is often used to deliver bulk unsolicited messages, or "spam", but filter programs exist which can automatically delete some or most of these, depending on the situation.

E-mail Architecture addresses those information systems that allow the electronic exchange of messages, documents, and other attachments. E-mail Architecture addresses those technologies used to enable the electronic delivery of messages and documents to one or more recipients. E-mail Architecture must support the reliable, secure, and efficient delivery and storage of electronic messages, documents and other attachments.

A typical email-architecture contains four elements:

1. **Post Offices :** Where outgoing messages are temporarily buffered (stored) before transmission and where incoming messages are stored. The post office runs the server software capable of routing messages (a message transfer agent) and maintaining the post office database.
2. **Message Transfer Agents (MTA) :** For forwarding messages between post offices and to the destination clients. The software can either reside on the local post office or on a physically separate server.
3. **Gateways :** Which provide parts of the message transfer agent functionally. They translate

between different e-mail systems, different e-mail addressing schemes and messaging protocols.

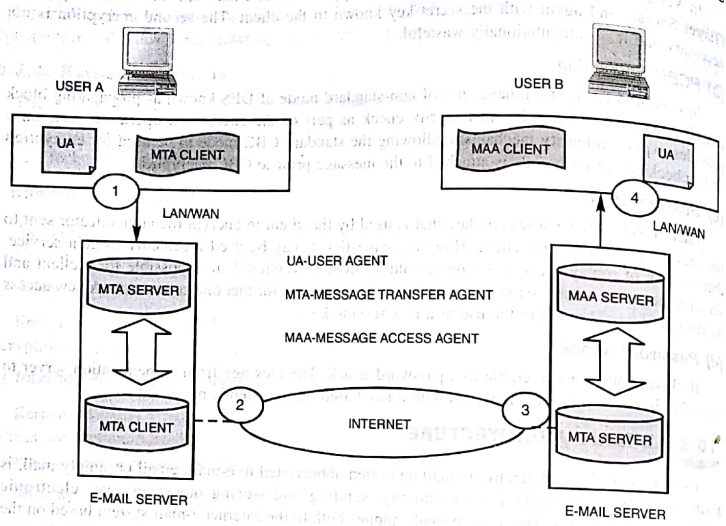


Fig. 10.3. *E-mail Architecture*

4. **E-Mail Clients :** Normally the computer which connects to the post office. It contains three parts:

   (*a*) E-mail Application Program Interface (API) such as MAPI, VIM, MHS and CMC.

   (*b*) Messaging protocol. The main messaging protocols are SMTP or X.400.STMP is defined in RFC 822 and RFC 821, Where as X.400 is an OSI-defined e-mail message delivery standard.

   (*c*) Network transport protocol, such as Ethernet, FDDI, and so on.

Assume that user A working in an organization that runs an e-mail server, every employee is connected to the e-mail server through LAN. The administrator of the e-mail server at the A's site has created a queuing system that sends e-mail to the internet one-by-one. The administrator of the e-mail at the B's site has created a mailbox for every user connected to the server, the mail box holds received messages until they retrieved by the recipient.

When user A needs to send message to the user B, A invokes a USER AGENT (UA) program to prepare the message. A then uses another program, a message transfer agent (MTA), to send the program at mail server at its site.

The message received at the mail server at A's site is queued with all other messages, each goes to it at corresponding destination. In A's case, his messages go to the mail server at B's site. A client/

server MTA is responsible for the e-mail transfer between the two servers. When the message arrives at the destination main server, it is stored in B's mailbox, a special field that holds messages until it is received by B.

10.4. EMAIL SECURITY

Sending an e-mail is one time activity. In IPSec or SSL, we assume that the two parties create a session between themselves and exchange data in both directions. In e-mail, there is no session. Alice and Bob cannot create a session. Alice sends a message to Bob, sometime later; Bob reads the message and may or may not send a reply.

In E-mail security, the sender of the message needs to include the name or identifiers of the algorithms used in the message.

In E-mail security, the encryption/decryption is done using a symmetric key algorithm, but the secret key to decrypt the message is encrypted with the public key of the receiver and is sent with the message.

The most important characteristics of security are confidentiality and integrity. There are three protocols providing security services for e-mails.

1. Pretty Good Privacy (PGP)
2. Multipurpose Internet Mail Extension (MIME)
3. Secure/Multipurpose Internet Mail Extension (S/MIME)

10.4.1. Pretty Good Privacy (PGP)

PGP, or Pretty Good Privacy, is a security software application used for the encryption and decryption of data. PGP is a computer program that provides cryptographic privacy and authentication. PGP is often used for signing, encrypting and decrypting e-mails to increase reliability for e-mail communications.

In 1991, Philip R. Zimmermann wrote PGP for the purpose of sending secured data across an insecure network, such as the internet. Individuals, businesses, and governments use strong cryptography programs such as PGP to secure networks, emails, documents, and stored data.

PGP was originally designed as a combination of RSA encryption and a symmetric key cipher known as Bass-O-Matic.

Bass-O-Matic is a conventional (often referred to as symmetric) key algorithm. Bass-O-Matic was later replaced by another conventional key algorithm known as IDEA, which enabled more powerful encryption technology.

The combination of both public and conventional key cryptology makes PGP a hybrid cryptosystem. This allows for users of PGP to be able to securely exchange keys and still have a speedy transaction of secured data. PGP automatically provides data confidentiality, data integrity, and origin authentication with the option of non-repudiation.

(i) Confidentiality

To protect the messages from eavesdropping, PGP encrypts the message using the public key of the recipient. So only the intended recipient can read the message after decrypting the message with his private key.

between different e-mail systems, different e-mail addressing schemes and messaging protocols.

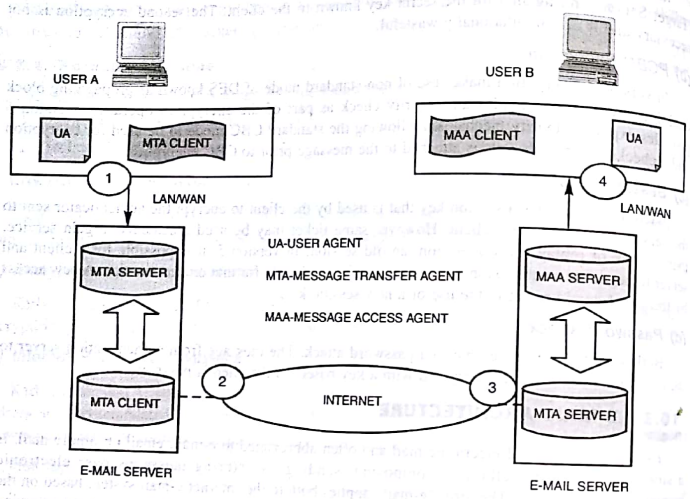


Fig. 10.3. *E-mail Architecture*

4. **E-Mail Clients :** Normally the computer which connects to the post office. It contains three parts:
   (*a*) E-mail Application Program Interface (API) such as MAPI, VIM, MHS and CMC.
   (*b*) Messaging protocol. The main messaging protocols are SMTP or X.400.STMP is defined in RFC 822 and RFC 821, Where as X.400 is an OSI-defined e-mail message delivery standard.
   (*c*) Network transport protocol, such as Ethernet, FDDI, and so on.

Assume that user A working in an organization that runs an e-mail server, every employee is connected to the e-mail server through LAN. The administrator of the e-mail server at the A's site has created a queuing system that sends e-mail to the internet one-by-one. The administrator of the e-mail at the B's site has created a mailbox for every user connected to the server, the mail box holds received messages until they retrieved by the recipient.

When user A needs to send message to the user B, A invokes a USER AGENT (UA) program to prepare the message. A then uses another program, a message transfer agent (MTA), to send the program at mail server at its site.

The message received at the mail server at A's site is queued with all other messages, each goes to it at corresponding destination. In A's case, his messages go to the mail server at B's site. A client/

server MTA is responsible for the e-mail transfer between the two servers. When the message arrives at the destination main server, it is stored in B's mailbox, a special field that holds messages until it is received by B.

10.4. EMAIL SECURITY

Sending an e-mail is one time activity. In IPSec or SSL, we assume that the two parties create a session between themselves and exchange data in both directions. In e-mail, there is no session. Alice and Bob cannot create a session. Alice sends a message to Bob, sometime later; Bob reads the message and may or may not send a reply.

In E-mail security, the sender of the message needs to include the name or identifiers of the algorithms used in the message.

In E-mail security, the encryption/decryption is done using a symmetric key algorithm, but the secret key to decrypt the message is encrypted with the public key of the receiver and is sent with the message.

The most important characteristics of security are confidentiality and integrity. There are three protocols providing security services for e-mails.

1. Pretty Good Privacy (PGP)
2. Multipurpose Internet Mail Extension (MIME)
3. Secure/Multipurpose Internet Mail Extension (S/MIME)

10.4.1. Pretty Good Privacy (PGP)

PGP, or Pretty Good Privacy, is a security software application used for the encryption and decryption of data. PGP is a computer program that provides cryptographic privacy and authentication. PGP is often used for signing, encrypting and decrypting e-mails to increase reliability for e-mail communications.

In 1991, Philip R. Zimmermann wrote PGP for the purpose of sending secured data across an insecure network, such as the internet. Individuals, businesses, and governments use strong cryptography programs such as PGP to secure networks, emails, documents, and stored data.

PGP was originally designed as a combination of RSA encryption and a symmetric key cipher known as Bass-O-Matic.

Bass-O-Matic is a conventional (often referred to as symmetric) key algorithm. Bass-O-Matic was later replaced by another conventional key algorithm known as IDEA, which enabled more powerful encryption technology.

The combination of both public and conventional key cryptology makes PGP a hybrid cryptosystem. This allows for users of PGP to be able to securely exchange keys and still have a speedy transaction of secured data. PGP automatically provides data confidentiality, data integrity, and origin authentication with the option of non-repudiation.

(i) Confidentiality

To protect the messages from eavesdropping, PGP encrypts the message using the public key of the recipient. So only the intended recipient can read the message after decrypting the message with his private key.

(ii) Data Origin Authentication

PGP vouches for the authenticity of the originator of the message by appending the originator's signature to the message. The signature is generated using the private key of the sender of the message. The private key of the sender is supposed to be known only to him. Hence the signature uniquely identifies the sender of the message.

(iii) Message Integrity

PGP also provides the recipient a means to check that the message reached him intact without any tampering/modification on the transit. This is done by means of sending a message digest of the original message to the recipient.

General Working

PGP follows a simple process when encrypting plaintext into cipher text. PGP first compresses the document desired for encryption. This saves modem transmission time and strengthens the cryptographic security of the plaintext.

Next, PGP creates a session key. The key is a number correlating to the random movements of the user's mouse and the keys that are typed. The key then works with a cryptographic algorithm to encrypt the plaintext.

A cryptographic algorithm is a mathematical function in which a computable set of steps must be followed to achieve a desired result. The strength of this encryption is dependent on the strength of the algorithm.

After the data has been encrypted into cipher text, PGP encrypts the session key. The session key is encrypted to the recipient's public key.

PGP uses digital certificates to prove the identity of a public key. The cipher text and encrypted session key are then transmitted to the recipient. When the recipient receives the data, PGP uses the user's private key to decrypt the session key. When PGP has recovered the session key, it can be used to decrypt the cipher text.

Though the plaintext has been recovered, there is still a question of authentication. PGP uses digital signatures to provide the recipient of an encryption with an origin and identification. Digital signatures are created in the opposite way a public cryptography system works. The sender encrypts a digital signature with their private key and attaches it to the rest of the data transmitted. When the digital signature is received, PGP decrypts it with the sender's public key. Through this process, PGP is able to determine the authenticity of the signature.

PGP then creates a digital signature with the message digest and the user's private key. The hash function helps to prove the authenticity of the encryption. If the encryption is changed after this process takes place, an entirely new message digest is created. This allows for PGP to detect encryption tampering.

Following diagram will definitely clear a view of PGP, and also how broadly it works in reality.

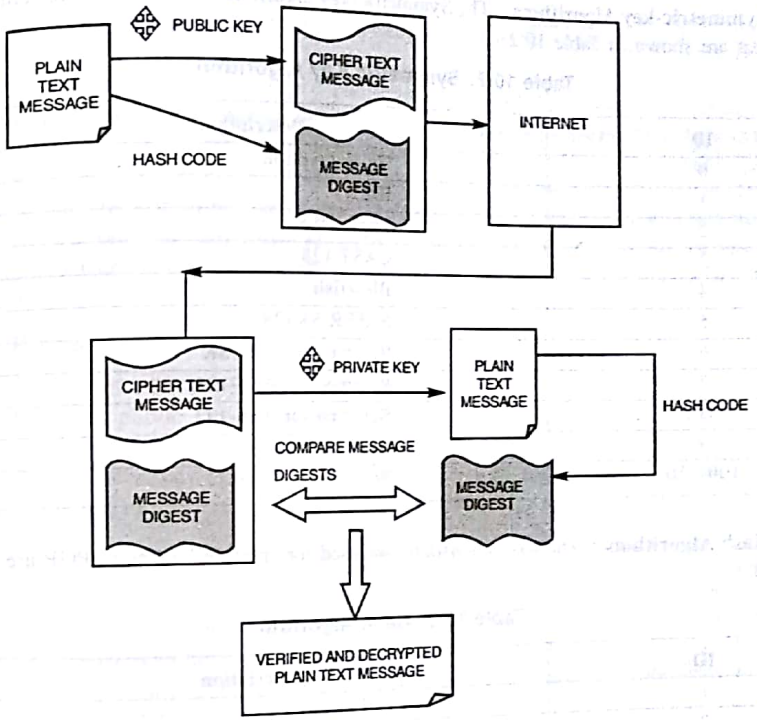


Fig. 10.4. *Working of PGP*

PGP Algorithm

Following algorithms are used in PGP:

1. Public-key Algorithms : The public-key algorithms are used for signing the digests or encrypting the messages are listed in Table 10.1.

Table 10.1. Public Key Algorithm

ID	Description
1	RSA (encryption or signing)
2	RSA (for encryption only)
3	RSA (for signing only)
16	ElGamal (encryption only)
17	DSS
18	Reserved for elliptic curve
19	Reserved for ECDSA
20	ElGamal (for encryption or signing)
21	Reserved for Diffie-Hellman
100-110	Private algorithms

2. Symmetric-key Algorithms : The Symmetric-key algorithms that are used for conventional encrypting are shown in Table 10.2.

Table 10.2. Symmetric Key Algorithm

ID	Description
0	No Encryption
1	IDEA
2	Triple DES
3	CAST-128
4	Blowfish
5	SAFER-SK128
6	Reserved for DES/SK
7	Reserved for AES-128
8	Reserved for AES-192
9	Reserved for AES-256
100-110	Private algorithms

3. Hash Algorithms : The hash algorithms are used for creating hashes in PGP are shown in Table 10.3.

Table 10.3. Hash Algorithm

ID	Description
1	MD5
2	SHA-1
3	RIPE-MD/160
4	Reserved for double-width SHA
5	MD2
6	TIGER/192
7	Reserved for HAVAL
7	Private algorithms

4. Compression Algorithms : The compression algorithms that are used for compressing text are shown in Table 10.4.

Table 10.4. Compression Algorithm

ID	Description
0	Uncompressed
1	ZIP
2	ZLIP
100-110	Private algorithms

PGP Certificates

Like other protocols PGP uses certificates to authenticate public keys. However, the process is totally different.

In PGP, anyone in the ring can sign a certificate for anyone else in the ring. Bob can sign a certificate for Ted, John, Anne, and so on. There is no hierarchy of trust in PGP; there is no tree. The lack of hierarchical structure may result in the fact that Ted may have one certificate from Bob and another certificate from Liz. If Alice wants to follow the line of certificates for Ted, there are two paths: one starts from Bob and one starts from Liz. An interesting point is that Alice may fully trust Bob, but only partially trust Liz. There can be multiple paths in the line of trust from a fully or partially trusted authority to a certificate. In PGP, the issuer of a certificate is usually called an introducer.

PGP Key Rings

The real power of PGP is not the encryption of files, but the encryption of electronic mail messages. PGP uses public key cryptography, which allows anybody to create a message and encrypt it using your public key. After the message is encrypted, no one can decrypt it unless someone has your secret key. (Ideally, nobody other than you should have a copy of your key.) PGP also allows you to electronically "sign" a document with a digital signature, which other people can verify.

PGP stores keys in two files on your hard disk; one for public keys and one for private keys. These files are called key rings. These two types of key rings are the principal method of storing and managing public and secret keys. Rather than keep individual keys in separate key files, they are collected in key rings to facilitate the automatic lookup of keys either by key ID or by user ID. Each user keeps his own pair of key rings. An individual public key is temporarily kept in a separate file long enough to send to your friend who will then add it to her key ring.

NOTE ▶

Suppose there are n users who want to communicate with each other then each user have (n – 1) public keys for n users and one own secret key for all in PGP key ring.

To make use of these features, you will first need to create a public key for yourself and distribute it among your correspondents.

The **passphrase** is used to encrypt the secret key that is stored on your computer. In this manner, if somebody breaks into your account or steals your computer, they won't be able to read your encrypted messages.

After you've generated your key, you should do two things with it immediately:

1. Sign it yourself. You should always sign your own key right away. There are some obscure ways that your key might be abused if it is circulated without a signature in place, so be sure that you sign it yourself.

2. Generate a revocation certificate and store it offline somewhere. *Don't send it to anyone!* The idea behind generating the revocation right now is that you still remember the passphrase and have the secret key available. If something should happen to your stored key, or you forget the passphrase, the public/private key pair becomes useless. Having the revocation certificate ready in advance allows you to send it out if that should ever happen.

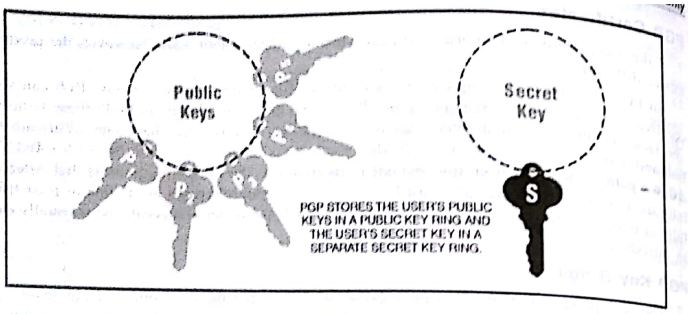


Fig. 10.5. PGP key Rings

PGP Detached Signatures

PGP has the ability to store digital signatures in a separate file from the original document. Such a signature is called a *detached signature*. Detached signatures are recommended for binary files, such as programs, because the signature will not change the data.

For the UNIX system administrator, one of the truly valuable things that you can do with PGP is to create detached signatures of your critical system files. These signatures will be signed by you, the system administrator. You (or other users on your system) can then use these signatures to detect unauthorized modification in the critical system files: if the files that you sign are ever modified, the signature will no longer validate.

PGP Trust Model

In relation to the cryptographic and legal issue of PGP, How PGP automatically derives. "Trustworthiness" is important, as wrongful underlying assumptions can undetermine its effectiveness. In PGP trust model, there are different levels of trust, complete trust, marginal trust, and no trust.

These levels correspond to how much the user thinks the owner of this public key can be trusted to be an 'introducer' to another trustworthy public-key certificate. Trust levels can be one of these:

- **Complete Trust :** Here, public-key is fully trusted to introduce another public-key.
- **Marginal Trust :** Here, public-key can be trusted to introduce another public-key, but, it is uncertain whether it is fully competent to do that.
- **Untrustworthy:** Here, public-key should not be trusted to introduce another, therefore any occurrence of this key as a signature on another public-key should be ignored.

A big problem with all these public key stuff is knowing that a public key really came from the person whose name is on it. Other advocates of public key technology are proposing complex hierarchies in which your key is registered with some big organization that signs your key. The big organization's key is signed by some bigger agency and so on up to some super-dooper master certifying agency.

The Web of Trust — which has nothing to do with the Internet's World Wide Web, by the way — works by having people sign the keys of people they know. If you have enough signers, and I have enough signers, and all the signers have enough signers, the chances are good that we may have a signer in common. If so, we can be pretty confident that we each are who we say we are. For example, Bill knows Bob who knows Sally. Bill also know Marko who knows Irena who knows Ofer. So Sally can know who Ofer is. The Web of Trust is a nice concept, and no one can revoke your keys because you didn't pay your parking tickets.

10.4.2. Multipurpose Internet Mail Extension (MIME)

The basic Internet e-mail transmission protocol, SMTP, supports only 7-bit ASCII characters. This effectively limits Internet e-mail to messages which, *when transmitted*, include only the characters sufficient for writing a small number of languages, primarily English.

MIME defines mechanisms for sending other kinds of information in e-mail. These include text in languages other than English using character encodings other than ASCII, and 8-bit binary content such as files containing images, sounds, movies, and computer programs. MIME is also a fundamental component of communication protocols such as HTTP, which requires that data be transmitted in the context of e-mail-like messages even though the data might not fit this context. Mapping messages into and out of MIME format is typically done automatically by an e-mail client or by mail servers when sending or receiving Internet (SMTP/MIME) e-mail.

RFC 822 : The basic format of Internet e-mail is defined in RFC 2822, which is an updated version of RFC 822. These standards specify the familiar formats for text e-mail headers and body and rules pertaining to commonly used header fields such as "To:", "Subject:", "From:", and "Date:". MIME defines a collection of e-mail headers for specifying additional attributes of a message including *content type*, and defines a set of *transfer encodings* which can be used to represent 8-bit binary data using characters from the 7-bit ASCII character set. MIME also specifies rules for encoding non-ASCII characters in e-mail message headers, such as "Subject:", allowing these header fields to contain non-English characters.

The goals of the MIME definition included requiring no changes to extant e-mail servers, and allowing plain text e-mail to function in both directions with extant clients. These goals were achieved by using additional RFC 822-style headers for all MIME message attributes and by making the MIME headers optional with default values ensuring a non-MIME message is interpreted correctly by a MIME-capable client. A simple MIME text message is therefore likely to be interpreted correctly by a non-MIME client although it has e-mail headers the non-MIME client won't know how to interpret. Similarly, if the quoted printable transfer encoding (see below) is used, the ASCII part of the message will be intelligible to users with non-MIME clients.

MIME is an extension to the RCF 822 framework that is intended to address some of the problems and limitations of the use of SMTP (Simple Mail Transfer Protocol). There is some following limitation of the SMTP/822 scheme:

1. SMTP cannot transmit executable files or other binary object.
2. SMTP cannot transmit text data that includes national language characters because these are represented by 8-bit codes with the values of 128 decimal or higher, and SMTP is limited to 7-bit ASCII.
3. SMTP server may reject mail message over a certain size.
4. SMTP gateways that translate between ASCII and the character code EBCDIC do not use a consistent set of mappings, resulting in translation problems.

5. SMTP gateways to X.400 electronic mail network cannot handle nontextual data included in X.400 messages.
6. Some SMTP implementation does not adhere completely to the SMTP standards defined in RFC821. common problems are include:
   - Deletion, addition, or reordering of carriage return and linefeed
   - Truncating or wrapping lines longer than 76 characters
   - Removal of trailing white space (tab and space characters)
   - Padding of lines in a message to the same length
   - Conversion of tab characters into multiple space characters

MIME is intended to resolve these problems in a manner that is compatible with existing RFC822 implementations. The specification is provided in RFCs 2045 through 2049.

The MIME specification includes the following elements:

1. Five new message header fields are defined, which may include in an RFC 822 headers. These fields provide information about the body of the message.
2. A number of content formats are defined, thus standardizing representations that support multimedia electronic mail.
3. Transfer encoding are defined that enable the conversion of any content formats into a form that protected from alteration by the mail system.

MIME defines five headers that can be added to the original e-mail header section to define the transformation parameters.

(i) MIME-Version

This header defines the version of MIME used. The current version is 1.0. This field indicate that the message conforms to RFCs 2045 and 2046.

(ii) Content-Type

This header defines the type of data used in the body of the message. MIME allows seven different types of data.

Text : The original message is in 7-bit ASCII format and no transformation by MIME is needed. There are two sub type currently used, plain and HTML.

Multipart : The body contain multiple, independent parts. The multipart header needs to define the boundary between each part. A parameter is used for this purpose. The parameter is a string broken that comes before each part; it is on separate line by itself and is preceded by two hyphens. The body is terminated using the boundary token, again preceded by two hyphens, and then terminated with two hyphen.

Four subtype are defined for this type: mixed. Parallel, digest, and alternative. In the mixed subtype, the parts must be presented to the recipient in the exact order as in the message.

Message : In the message type, the body is itself an entire mail message, a part of a mail message, or a pointer to a message. Three subtype are currently used: RFC822, partial, and external-body.

- The subtypes RFC822 are used if the body is encapsulating another message.
- The partial subtype is used if the original message has been fragmented into different mail messages and this mail message is one of the fragments. Three parameter must be added: id, number, and the total. The id identifies the message and is present in all the fragments.

The number defines the sequence order of the fragment. The total defines the number of fragments that comprise the original message

```
Content-type: message/partial;
Id="challenger.atc.fhad.edu";
Number=1;
Total=3;
------..
------.
```

- The subtype external-body indicates that the body does not contain the actual message but is only a reference to the original message.

```
Content-type: message/ external-body;
Name="report.txt";
Site="fhad.edu";
Access-type="ftp";
----------
----------
```

Image: The original message is a stationary image, indicating that there is no animation. The two currently used subtypes are Joint photographic Expert group (JPEG). Which uses image compression, and Graphic Interchange Format (GIF).

Video : The original message is a time varying image (animation). The only subtype is Moving Picture Expert Group (MPEG). If the animated image contains sounds, it must be sent separately using the audio content type.

Audio : The original message is sound. The only subtype is basic, which uses 8 kHz standard audio data.

Application : The original message is a type of data not previously defined. There are only two subtypes used currently: PostScript and Octet-stream. PostScript is used when the data are in Adobe PostScript format. Octet-stream is used when the data must be interpreted as a sequence of 8-bit or 1 byte.

(iii) Content-Transfer-Encoding

In June 1992, MIME (RFC 1341, since obsolete by RFC 2045) defined a set of methods for representing binary data in ASCII text format. The *content-transfer-encoding:* MIME header has 2-sided significance:

1. It indicates whether or not a binary-to-text encoding scheme has been used on top of the original encoding as specified within the Content-Type header, and
2. If such a binary-to-text encoding method has been used it states which one.

Suitable for use with normal SMTP :

- **7 bit** — up to 998 octets per line of the code range 1.127 with CR and LF (codes 13 and 10 respectively) only allowed to appear as part of a CRLF line ending. This is the default value.
- **Quoted-printable :** It used to encode arbitrary octet sequences into a form that satisfies the rules of 7 bit. Designed to be efficient and mostly human readable when used for text data consisting primarily of US-ASCII characters but also containing a small proportion of bytes with values outside that range.

- **Base 64** — It is used to encode arbitrary octet sequences into a form that satisfies the rules of 7bit. Designed to be efficient for non-text 8 bit data. Sometimes used for text data that frequently uses non-US-ASCII characters.
- **8-bit** : Up to 998 octets per line with CR and LF (codes 13 and 10 respectively) only allowed appearing as part of a CRLF line ending.
- **Binary :** This is 8-bit encoding. Non-ASCII character can be sent, and the length of the line can exceed 1,000 characters can. MIME does not perform any encoding here; the underlying SMTP protocol must be able to transfer binary data.

(iv) Content-Id

This header uniquely identifies the whole message in a multiple message environment.

```
Content-id: id=< content-id >
```

(v) Content-Description

This header defines whether the body is image, audio, or video.

```
Content-Description:<description>
```

10.4.3. Secure/Multipurpose Internet Mail Extension (S/MIME)

S/MIME adds some new content type to include security service to the MIME. All of these new type include the parameter "application/pkcs 7-mime", in which "pkcs" defines "public key cryptography specification".

Cryptographic Message Syntax (CMS)

To define how security services, such as confidentiality or integrity, can be added to MIME content type, S/MIME has defined Cryptographic message syntax (CMS). The syntax in each case defined the exact encoding scheme for each content type. The following describe the type of message and different subtype of message and different subtypes that are created this message. For details, the reader is referred to RFC3369 and 3370.

Data Content Type

This is an arbitrary string the object created is called data.

(i) **Signed-Data Content Type :** This type provides only integrity of data. It contains any type and zero or more signature values. The encoded result is an object called signed Data. The following are the steps in the process:

- For each signer, a message digest is created from the content using the specific hash algorithm chosen by that signer.
- Each message digest is signed with the private key of the signer.
- The content, signature values, certificate, and algorithms are then collected to create the signed Data object.

(ii) **Enveloped-Data Content Type :** This type is used to provide privacy for the message. It contains any type and zero or more encrypted keys and certificates. The encoded result is an object called enveloped Data.

- A pseudorandom session key is created for the symmetric-key algorithms to be used.

- For each recipient, a copy of the session key is encrypted with the public key of each recipient.
- The encrypted contents, encrypted session keys, algorithm used, and certificates are encoded using Radix-64.

(iii) **Digested-Data Content Type :** This type is used to provide integrity for the message. The result is normally used as the content for the enveloped-data content type. The encoded result is an object called digested Data.

- A message digest is calculated from the content.
- The message digest, the algorithm, and the content are added together to create the digested Data object.

(iv) **Encrypted-Data Content Type :** This type is used to create an encrypted version of any content type. Although this looks like the enveloped-data content type, the encrypted-data content type has no recipient. It can be used to store the encrypted data instead of transmitting it. The process is very simple; the user employs any key and any algorithm to encrypt the content. The encrypted content is stored without including the key or the algorithm. The object created is called encrypted Data.

(v) **Authenticated-Data Content Type :** This type is used to provide authentication of the data. The object is called authenticated Data.

- Using a pseudorandom generator, a MAC key is generated for each recipient.
- The MAC key is encrypted with the public key of the recipient.
- A MAC is created for the content.
- The content, MAC algorithms, and other information are collected together to form the authenticated data object.

Key Management

The key management is S/MIME is a combination of key management used by X.509 and PGP. S/MIME uses public-key certificates signed by the certificate authorities defined by X.509. However, the user is responsible to maintain the web of trust to verify signatures as defined by PGP.

Cryptographic Algorithms

S/MIME defines several Cryptographic algorithms as in following table. The term "must" means an absolute requirement; the term "should" means recommendation.

Must : The definition is an absolute requirement of the specification. An implementation must include this feature or function to 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.

The following rules, in the following order, should be followed by sending agent :

1. If the sending agent has a list of preferred decrypting capabilities from an intended recipient, it SHOULD choose the first 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 message from the recipient, then 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, and then the sending agent SHOULD use triple DES.
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, and then the sending agent MUST use RC2/40.

Table 10.5. S/MIME Algorithm

Algorithm	Sender must support	Receiver must support	Sender should support	Receiver should support
Content-encryption algorithm	Triple DES	Triple DES		1. AES 2. RC2/40
Session key encryption algorithm	RSA	RSA	Diffie-Hellman	Diffie-hellman
Hash algorithm	SHA-1	SHA-1		MD5
Digest-encryption algorithm	DSS	DSS	RSA	RSA
Message-authentication algorithm				

S/MIME Certificate Processing

S/MIME uses public key certificate that conform to version 3 of X.509. The key-management scheme used by S/MIME is in some ways a hybrid between a strict X.509 certification hierarchy and PGP's web of trust.

User Agent Role

An S/MIME user has several key-management functions to perform:

- **Key generation :** The user of some related administrative utility MUST be capable of generating separate Diffie-Hellman and DSS key pair and SHOULD be capable of generating RSA key pairs. Each key pair MUST be generated from a good source of non-deterministic random input and be protected in a secure fashion. A user agent SHOULD generate RSA key pairs with a length in the range of 768 to 1024 bits and MUST NOT generates a length of less than 512 bits.
- **Registration :** A user's public key must be registered with a certification authority in order to receive an X.509 public-key certificate.
- **Certificate storage and retrieval :** A user requires access to a local list certificates in order to verify incoming signatures and to encrypt outgoing messages. Such a list could be maintained by the user or by some local administrative entity on behalf of a number of users.

Verisign Certificates

There are several companies that provide certification authority (CA). Verisign provides a CA service that is intended to be compatible with S/MIME and a Varity of other application. Verisign issues X.509 certificates with the product name Verisign digital ID.

The information contained in a Digital ID developed on the type of digital ID and its use. At a minimum, each digital ID contains

- Owner's public key
- Owner's name or alias
- Expiration date of the digital ID
- Serial number of the digital ID
- Name of the signature certification authority that issued the digital ID

Digital IDs can also contain other user's supplied information, including:

- Address
- E-mail address
- Basic registration information (country, zip code, age, and gender)

Verisign provides three levels, or classes of security for public-key certificates, the following procedure are used:

- **For class 1** digital IDs, Verisign confirm the user's e-mail address by sending a PIN and digital ID pick-up information to the e-mail address provided in the application.
- **For class 2** digital IDs, Verisign verifies the information in the application through an automated comparison with consumer data base in addition to performing all of the checking associated with a class 1 digital ID. Finally, confirmation is sent to the specified postal address alerting the user that a digital ID has been issued on his or her name.
- **For class 3** digital IDs, Verisign requires a higher level of identify assurance. An individual must prove his or her identity by providing notarized credentials or applying in person.

Enhanced Security Services

There are three Enhanced security services have been proposed in an internet draft. The three services are as follows:

- **Signed receipts :** A signed receipt may be requested in a signed Data 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.
- **Security labels :** A security label may be include in the authenticated attributes of a signed data object. A security label is asset of security information regarding the sensitivity of the content that is protected by S/MIME encapsulation.
- **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).

SUMMARY

- Authentication is the act of establishing identity via the presentation of information that allows the verifier to know the presenter is who or what it claims.
- Kerberos is both an authentication protocol and a KDC.
- Kerberos protocol involve: Authentication Server, Ticket Granting Server and Application Server.
- Kerberos has two versions: Version 4 and Version 5.
- The version 5 of Kerberos is to address the limitation of version 4 in following two areas: Environmental Shortcoming and Technical Deficiencies.
- In version 5, an encrypted message is tagged with an encryption algorithm identifier.
- It gives the user the option of another algorithm
- It supports a technique known as authentication forwarding.
- It allows credentials issued to one client to be forwarded to some other host and used by some other client. (Version 4 does not support)

CHAPTER 11

IP Security

11.1. IP-SECURITY OVERVIEW

Inside This Chapter

In 1994, the Internet Architecture Board (IAB) issued a report entitled "Security in the Internet Architecture" (RFC 1636). The report stated the general consensus that the Internet needs more and better security, and it identified key areas for security mechanisms. In response to these issues, the IAB included authentication and encryption as necessary security features in the next-generation IP, which has been issued as IPv6. IPsec implementation is a mandatory part of IPv6, but is not an integral part of IPv4. However, because of the slow uptake of IPv6, IPsec is most commonly used to secure IPv4 traffic.

IP Security (IPSec) protocols operate at the network layer, layer 3 of the TCP/IP model as shown in figure. IPSec is a collection of protocols designed by the Internet Engineering Task Force (IETF) to provide security for a packet at the network level. The network layer in the internet is often referred to as the internet protocol or IP layer. IP-level security encompasses three functional areas: Authentication, Confidentiality and Key management. IPsec was introduced to provide security services such as the following :

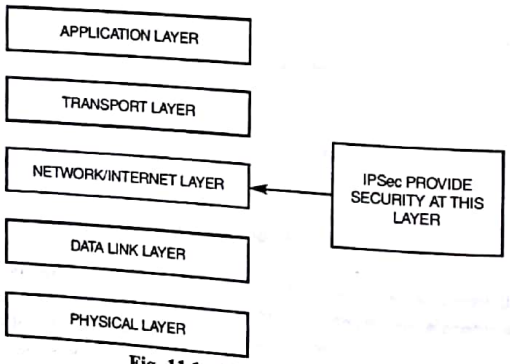


Fig. 11.1. *Idea of IPSec*

- Encrypting traffic (so it cannot be read by parties other than those for whom it is intended).
- Integrity validation (ensuring traffic has not been modified along its path).
- Authenticating the peers (ensuring that traffic is from a trusted party).
- Anti-replay (protecting against replay of the secure session).

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 parameter.
- IPSec in a firewall is resistant to bypass.
- IPSec is below the transport layer (TCP, UDP) and so it is transparent to applications. There is no need to change the software on a user or server system when IPSec is implemented in the firewall or router.
- IPSec is transparent to end users. There is no need to train users.
- IPSec can provide security for individual users.

Services provided by IPSec

The security services provided by IPSec are as follows :

- Access control
- Connectionless integrity
- Data origin authentication
- Confidentiality
- Anti replay service.

11.2. IPSec MODES

IPSec operates in one of the two modes as shown in Fig. 11.2.

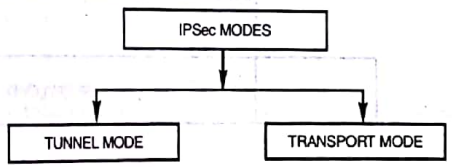


Fig. 11.2. *IPSec Modes*

11.2.1. Tunnel Mode

In tunnel mode, IPSec protects the entire IP layer packet (IP packet). The process of tunnel mode can be understood as shown in Fig. 11.3.

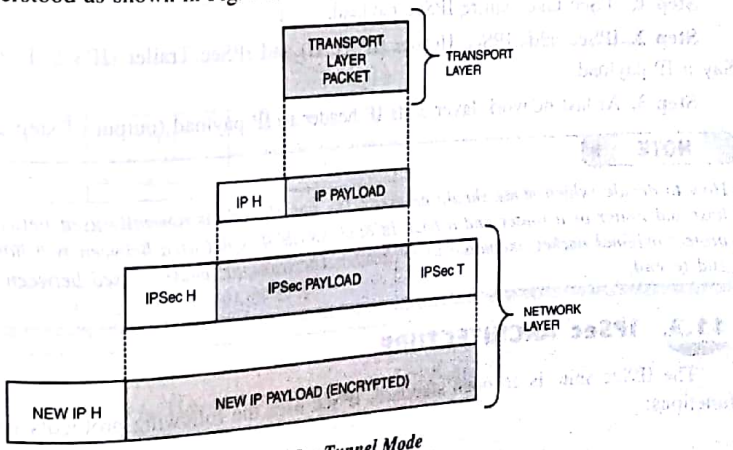


Fig. 11.3. *IPSec Tunnel Mode*

Network layer adds IP header to the payload received from transport layer as network layer usually did. It is called IPSec payload. IPSec performs following steps on the IPSec payload in tunnel mode.

Step 1. IPSec takes entire IPSec payload.

Step 2. IPSec adds IPSec Header (IPSec H) and IPSec Trailer (IPSec T) to IPSec payload.

Step 3. Now it encrypts the whole packet (after step2). Say it new IP payload.

Step 4. At last network layer adds IP header to new IP payload (output of step3).

11.2.2. Transport Mode

In transport mode, IPSec protects the entire packet received from transport layer. The process of transport mode can be understood as shown in Fig. 11.4.

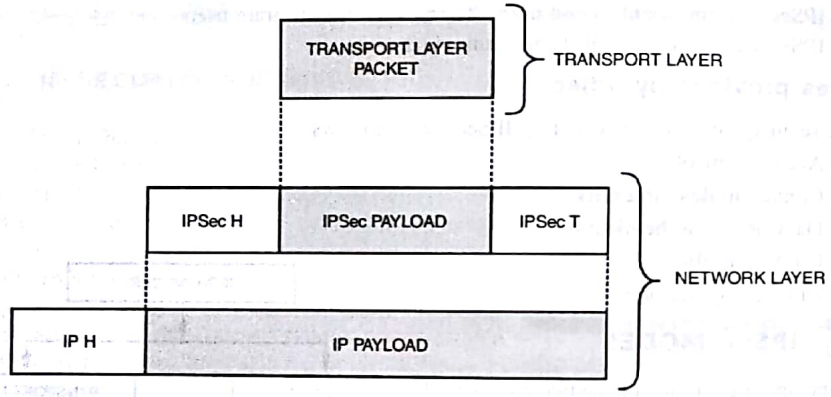


Fig. 11.4. *IPSec Transport Mode*

The packet received from transport layer is called IPSec Payload. Now IPSec performs following steps on the IPSec payload.

Step 1. IPSec takes entire IPSec payload.

Step 2. IPSec adds IPSec Header (IPSec H) and IPSec Trailer (IPSec T) to IPSec payload. Say it IP payload.

Step 3. At last network layer adds IP header to IP payload (output of step 2).

NOTE

How to decide, which mode should be used? The tunnel mode is normally used between two routers, a host and router or a router and a host. In other words it is not used between two hosts. The idea is to protect original packet, including its IP header. The transport mode is used between host to host (i.e., end-to-end).

11.3. IPSec ARCHITECTURE

The IPSec suite is an open standard. IPSec uses the following protocols to perform various functions:

- Authentication Headers (AH) provide connectionless integrity and data origin authentication for IP datagram and provides protection against replay attacks.
- Encapsulating Security Payloads (ESP) provide confidentiality, data origin authentication, connectionless integrity, an anti-replay service (a form of partial sequence integrity), and limited traffic flow confidentiality.
- Security associations (SA) provide the bundle of algorithms and data that provide the parameters necessary to operate the AH and/or ESP operations. The Internet Security Association and Key Management Protocol (ISAKMP) provides a framework for authentication and key exchange, with actual authenticated keying material provided either by manual configuration with pre-shared keys.

IPSec defines two types of protocols.

1. Authentication Header (AH)
2. Encapsulating Security Payload (ESP)

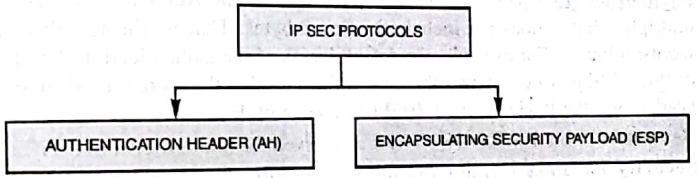


Fig. 11.5. *Types of IPSec Protocols*

11.3.1. Authentication Header (AH)

The Authentication Header (AH) Protocol is used to ensure the integrity and to authenticate the source host of the payload carried in the IP packet. It does not provide privacy.

The protocol creates a message digest using symmetric key; the digest is inserted in the authentication header. The AH is then placed in the appropriate location, based on the mode (transport or tunnel). The format of an AH packet in transport mode is shown in Fig. 11.6.

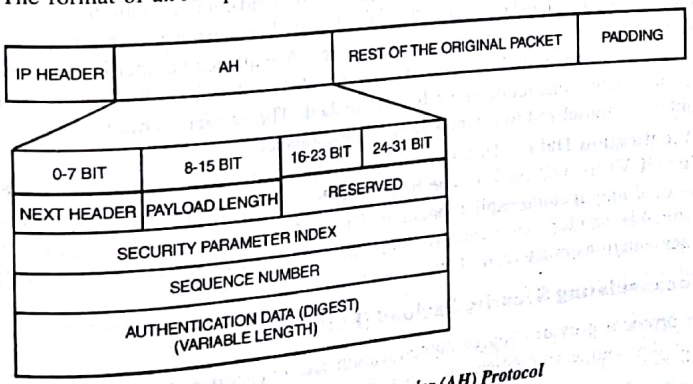


Fig. 11.6. *Authentication Header (AH) Protocol*

The Authentication Header (AH) Protocol follows the following steps :

1. An Authentication Header is added to the payload with the authentication data field set to 0.
2. Padding may be added to make the total length even for a particular hashing algorithm.
3. Hashing is based on the total packet. However, only those fields of the IP header that do not change during transmission are included in the calculation of the message digest (authentication data).
4. The authentication data are inserted in the authentication header.
5. The IP header is added after changing the value of the protocol field to 51.

The Authentication Header consists of the following fields :

(*i*) **Next Header (8 bits) :** The Next Header is an 8-bit field that identifies the type of the payload (such as TCP, UDP, and ICMP etc.). The value of protocol field in new IP datagram is now 51 to show that packet carries an authentication header.

(*ii*) **Payload length :** This field defines the length of the Authentication Header in 4-byte multiples, but it does not include the first 8 bytes. That is, the length of AH 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.

(*iii*) **Reserved (16 bits) :** This 16-bit field is reserved for future use. It MUST be set to "zero."

(*iv*) **Security Parameter Index (32 bits) :** The SPI is an arbitrary 32-bit value that, in combination with the destination IP address and security protocol (AH), uniquely identifies the Security Association for this datagram. The set of SPI values in the range 1 through 255 are reserved by the Internet Assigned Numbers Authority (IANA) for future use. The SPI value of zero (0) is reserved for local, implementation- specific use and MUST NOT be sent on the wire. For example, a key management implementation MAY use the zero SPI value to mean "No Security Association Exists" during the period when the IPsec implementation has requested that its key management entity establish a new SA, but the SA has not yet been established.

(*v*) **Sequence Number (32 bits) :** This unsigned 32-bit field contains a monotonically increasing counter value (sequence number).This field provide ordering information for a sequence of datagrams. the sequence prevents a playback. A sequence number is not repeated even if a packet is retransmitted. A sequence number does not wrap around even after it reaches 2^{32}; but a new connection must be established. The sender's counter and the receiver's counter are initialized to 0 when an SA is established.

(*vi*) **Authentication Data :** This is a variable-length field that contains the Integrity Check Value (ICV) for this packet. The field must be an integral multiple of 32 bits in length. This field may include explicit padding. This padding is included to ensure that the length of the AH header is an integral multiple of 32 bits (IPv4) or 64 bits (IPv6). All implementations must support such padding.

11.3.2. Encapsulating Security Payload (ESP)

The AH protocol provides source authentication and integrity, not privacy. So IPSec defined ESP that provides source authentication, integrity and privacy.

The Encapsulating Security Payload (ESP) header is designed to provide a mix of security services in IPv4 and IPv6. ESP may be applied alone, in combination with the IP Authentication Header (AH).The ESP header is inserted after the IP header and before the upper layer protocol header (transport mode) or before an encapsulated IP header (tunnel mode). ESP is used to provide confidentiality, data origin authentication, connectionless integrity, an anti-replay service (a form of partial sequence integrity), and limited traffic flow confidentiality. The set of services provided depends on options selected at the time of Security Association establishment and on the placement of the implementation. The format of an ESP packet is shown below.

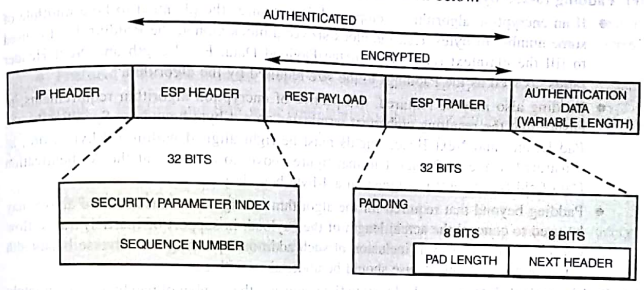


Fig. 11.7. *Encapsulating Security Payload (ESP) Protocol*

The ESP Protocol follows the following steps:

1. An ESP trailer is added to the payload.
2. The payload and the trailer are encrypted.
3. The ESP header is added.
4. The ESP header, payload, and ESP trailer are used to create the authentication data.
5. The authentication data are added to the end of the ESP trailer.
6. The IP header is added after changing the protocol value to 50.

The ESP consists of the following fields :

(*i*) **Security Parameters Index (32 bits) :** The SPI is an arbitrary 32-bit value that, in combination with the destination IP address and security protocol (ESP), uniquely identifies the Security Association for this datagram. The set of SPI values in the range 1 through 255 are reserved by the Internet Assigned Numbers Authority (IANA) for future use. The SPI value of zero (0) is reserved for local, implementation-specific use and must not be sent on the wire. For example, a key management implementation may use the zero SPI value to mean "No Security Association Exists" during the period when the IPsec implementation has requested that its key management entity establish a new SA, but the SA has not yet been established.

(*ii*) **Sequence Number (32 bits) :** This unsigned 32-bit field contains a monotonically increasing counter value (sequence number). This field provide ordering information for a sequence of datagram. The sequence prevents a playback. A sequence number is not repeated even if a packet is retransmitted. A sequence number does not wrap around even after it reaches

2^{32}; but a new connection must be established. The sender's counter and the receiver's counter are initialized to 0 when an SA is established.

(*iii*) **Payload Data (variable) :** Payload Data is a variable-length field containing data described by the Next Header field. The Payload Data field is mandatory and is an integral number of bytes in length. If the algorithm used to encrypt the payload requires cryptographic synchronization data, *e.g.*, an Initialization Vector (IV), then this data may be carried explicitly in the Payload field.

(*iv*) **Padding (0-255 bytes) :** The padding fields serves several purposes :

- If an encryption algorithm is employed that requires the plaintext to be a multiple of some number of bytes, *e.g.*, the block size of a block cipher, the Padding field is used to fill the plaintext (consisting of the Payload Data, Pad Length and Next Header fields, as well as the Padding) to the size required by the algorithm.
- Padding also may be required, irrespective of encryption algorithm requirements, to ensure that the resulting ciphertext terminates on a 4-byte boundary. Specifically, the Pad Length and Next Header fields must be right aligned within a 4-byte word, as illustrated in the ESP packet format figure above, to ensure that the Authentication Data field (if present) is aligned on a 4-byte boundary.
- Padding beyond that required for the algorithm or alignment reasons cited above may be used to conceal the actual length of the payload, in support of (partial) traffic flow confidentiality. However, inclusion of such additional padding has adverse bandwidth implications and thus its use should be undertaken with care.

(*v*) **Pad Length (8 bits) :** The Pad Length field indicates the number of pad bytes immediately preceding it. The range of valid values is 0-255, where a value of zero indicates that no Padding bytes are present. The Pad Length field is mandatory.

(*vi*) **Next Header (8 bits) :** The Next Header is an 8-bit field that identifies the type of data contained in the Payload Data field, *e.g.*, an extension header in IPv6 or an upper layer protocol identifier. The value of this field is chosen from the set of IP Protocol Numbers defined in the most recent "Assigned Numbers" RFC from the Internet Assigned Numbers Authority (IANA). The Next Header field is mandatory.

(*vii*) **Authentication Data (variable) :** The Authentication Data is a variable-length field containing an Integrity Check Value (ICV) computed over the ESP packet minus the Authentication Data. The length of the field is specified by the authentication function selected. The Authentication Data field is optional, and is included only if the authentication service has been selected for the SA in question.

11.4. SECURITY ASSOCIATION (SA)

Security association is a very important aspect of IPSec. IPSec requires a logical relationship, called a Security Association (SA).

SA is an agreement between the hosts about following factors :

- IPSec protocol version number
- Mode of operation (transport or tunnel mode)
- Cryptographic Algorithms
- Cryptographic Keys
- Lifetime of keys, etc.

A Security Association (SA) is a simplex "connection" that affords security services to the traffic carried by it. Security services are afforded to an SA by the use of AH, or ESP, but not both. If both AH and ESP protection is applied to a traffic stream, then two (or more) SAs are created to afford protection to the traffic stream.

A security association is uniquely identified by three parameters:

- **Security Parameters Index (SPI) :** A bit string assigned to this SA and having local significance only. The SPI is carried in AH and ESP headers to enable the receiving system to select the SA under which a received packet will be processed.
- **Destination Address (DA) :** Currently, only unicast addresses are allowed: this is the address of the destination endpoint of the SA, which may be an end user system or a network system such as a firewall or router.
- **Protocol (P) :** This indicates whether the association is an AH or ESP security association.

Example : If Alice and Bob want confidentiality then they can share secret key between themselves. We can say that there are two Security Associations (SA) between Alice and Bob. Two SA are

- One outbound SA
- One inbound SA

Each of these SA stores the value of key in a variable and name of encryption/decryption algorithm in another. Figure 11.8 shows the concept of SA.

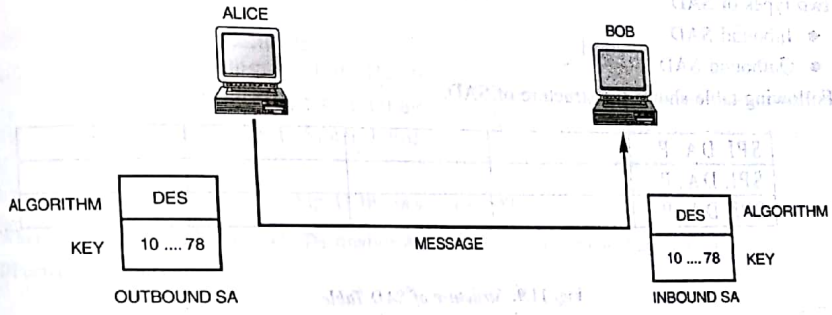


Fig. 11.8. *Concept of SA*

Two types of SAs modes are defined:

- Transport mode SA
- Tunnel mode SA

A transport mode SA is a security association between two hosts. In IPv4, a transport mode security protocol header appears immediately after the IP header and any options, and before any higher layer protocols (*e.g.*, TCP or UDP). In IPv6, the security protocol header appears after the base IP header and extensions, but may appear before or after destination options, and before higher layer protocols. In the case of ESP, a transport mode SA provides security services only for these higher layer protocols, not for the IP header or any extension headers preceding the ESP header. In the case of AH, the protection is also extended to selected portions of the IP header, selected portions of extension headers, and selected options (contained in the IPv4 header, IPv6 Hop-by-Hop extension header, or IPv6 Destination extension headers).

A tunnel mode SA is essentially an SA applied to an IP tunnel. Whenever either end of a security association is a security gateway, the SA must be tunnel mode. Thus an SA between two security gateways is always a tunnel mode SA, as is an SA between a host and a security gateway. Note that for the case where traffic is destined for a security gateway, *e.g.*, SNMP commands, the security gateway is acting as a host and transport mode is allowed. But in that case, the security gateway is not acting as a gateway, *i.e.*, not transiting traffic. Two hosts may establish a tunnel mode SA between themselves. The requirement for any (transit traffic) SA involving a security gateway to be a tunnel SA arises due to the need to avoid potential problems with regard to fragmentation and reassembly of IPsec packets, and in circumstances where multiple paths (*e.g.*, via different security gateways) exist to the same destination behind the security gateways.

In summary,

(*a*) A host must support both transport and tunnel mode.

(*b*) A security gateway is required to support only tunnel mode. If it supports transport mode, that should be used only when the security gateway is acting as a host, *e.g.*, for network management.

Security Association Database (SAD)

If sender wants to send message to many people or receiver needs to receive messages from many people. So we need a set of SA that can be stored in a database. This database is called Security Association Database (SAD). It can be 2-D table where each row for a single SA. There are two types of SAD

- Inbound SAD
- Outbound SAD

Following table shows the structure of SAD.

SPI, DA, P				
SPI, DA, P				
SPI, DA, P				

Fig. 11.9. Structure of SAD Table

Where SPI- Security Parameter Index, DA- Destination Address and P- Protocol

Inbound SAD : When a host receives a packet that carries an IPSec header, host need to find corresponding entry in inbound SAD to find the information for checking the security of the packet.

Outbound SAD : When a host need to send a packet that must carry an IPSec header, the host needs to find entry in outbound SAD to find the information for applying security to the packet.

The entry for each row is called SA parameters. Typical parameters are shown below.

- **Sequence Number Counter :** A 32-bit value used to generate the Sequnce Number field in AH or ESP headers.
- **Sequence Counter Overflow :** A flag indicating whether overflow of the Sequence Number Counter should generate an auditable event and prevent further transmission of packets on this SA.

- **Anti Reply Window :** Used to determine whether an inbound AH or ESP packet is a reply.
- **AH Information :** Authentication algorithms, keys, keys lifetimes, and related parameters being used with AH.
- **ESP Information :** Encryption and authentication algorithm, keys, initialization values, keys lifetimes, and related parameters being used with ESP.
- **Lifetime of SA :** A time interval or byte count after which an SA must be replaced with a new SA or terminated, plus an indication of which of these actions should occur.
- **IPSec Protocol Mode :** Tunnel, transport, or wildcard. These modes are discussed later in this section.
- **Path Maximum Transmission Unit (MTU) :** Any observed path maximum transmission unit and aging variables.

Security Policy Database (SPD)

It defines type of security applied to a packet when it is to be sent or when it has arrived. Each host that is using IPSec protocol needs to keep SPD. There are two types of SPD.

- Inbound SPD
- Outbound SPD

The structure of SPD table is shown below.

Index	Policy
SA, DA, Name, P, Sport, DPort	
SA, DA, Name, P, Sport, DPort	
SA, DA, Name, P, Sport, DPort	

Fig. 11.10. Structure of SPD Table

Where SA- Source Address, DA- Destination Address, P- Protocol, SPort-Source Port DPort-Destination Port

11.5. COMBINING SECURITY ASSOCIATIONS

The IP datagrams transmitted over an individual SA are afforded protection by exactly one security protocol, either AH or ESP, but not both. Sometimes a security policy may call for a combination of services for a particular traffic flow that is not achievable with a single SA. In such instances it will be necessary to employ multiple SAs to implement the required security policy. The term "security association bundle" or "SA bundle" is applied to a sequence of SAs through which traffic must be processed to satisfy a security policy.

The order of the sequence is defined by the policy. (Note that the SAs that comprise a bundle may terminate at different endpoints. For example, one SA may extend between a mobile host and a security gateway and a second, nested SA may extend to a host behind the gateway.)

Security associations may be combined into bundles in two ways:

1. Transport Adjacency
2. Iterated Tunneling

1. Transport Adjacency : Transport adjacency refers to applying more than one security protocol to the same IP datagram, without invoking tunneling. This approach to combining AH and ESP allows for only one level of combination; further nesting yields no added benefit assuming use of adequately strong algorithms in each protocol) since the processing is performed at one IPsec instance at the (ultimate) destination.

2. Iterated Tunneling : Iterated tunneling refers to the application of multiple layers of security protocols affected through IP tunneling. This approach allows for multiple levels of nesting, since each tunnel can originate or terminate at a different IPsec site along the path.

There are 3 basic cases of iterated tunneling — support is required only for cases 2 and 3.:

Case 1 : Both endpoints for the SAs are the same — The inner and outer tunnels could each be either AH or ESP, though it is unlikely that Host 1 would specify both to be the same, *i.e.*, AH inside of AH or ESP inside of ESP.

Case 2 : One endpoint of the SAs is the same — The inner and outer tunnels could each be either AH or ESP.

Case 3 : Neither endpoint is the same — The inner and outer tunnels could each be either AH or ESP.

These two approaches also can be combined, *e.g.*, an SA bundle could be constructed from one tunnel mode SA and one or two transport mode SAs, applied in sequence.

NOTE

Nested tunnels can also occur where neither the source nor the destination endpoints of any of the tunnels are the same. In that case, there would be no host or security gateway with a bundle corresponding to the nested tunnels.

For transport mode SAs, only one ordering of security protocols seems appropriate. AH is applied to both the upper layer protocols and (parts of) the IP header. Thus if AH is used in a transport mode, in conjunction with ESP, AH should appear as the first header after IP, prior to the appearance of ESP. In that context, AH is applied to the ciphertext output of ESP. In contrast, for tunnel mode SAs, one can imagine uses for various orderings of AH and ESP.

11.6. INTERNET KEY EXCHANGE (IKE)

Internet key exchange (IKE) is a protocol used for IPSec key management. IKE ia a protocol designed to create both inbound and outbound SA. IKE creates SA for IPSec.

IKE is used to negotiate the cryptographic algorithm to be later used by AH and ESP. IKE is the initial phase of IPSec, where algorithm and keys are decided. After IKE phase, AH and ESP protocol take over. The process is shown below.

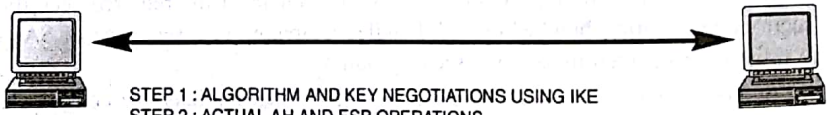


Fig. 11.11. *Steps in IPSec*

So without proper key management IPSec cannot exist. IKE is based on three protocols as shown in Fig. 11.12. Three protocols are given below :

1. ISAKMP (Internet Security Association and Key Management Protocol)
2. Oakley
3. SKEME

1. ISAKMP (Internet Security Association and Key Management Protocol)

It is a protocol for establishing Security Associations (SA) and cryptographic keys in an Internet environment. ISAKMP provides a framework for authentication and key exchange. It is designed to be key exchange independent. Authenticated keying material for use with ISAKMP is provided by protocols such as Internet Key Exchange.

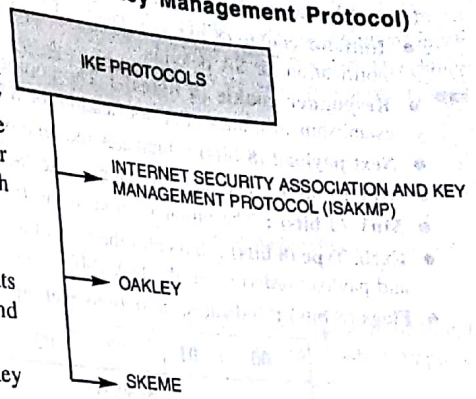


Fig. 11.12. *Types of IKE Protocols*

It defines the followings.

- Defines procedures and packet formats to establish, negotiate, modify and delete Security Associations (SAs)
- Defines payloads for exchanging key generation and authentication data
- Provides a consistent framework for transferring key and authentication data which is independent of the key generation technique, encryption algorithm and authentication mechanism

Each message consists of a header that is followed by at least one payload.

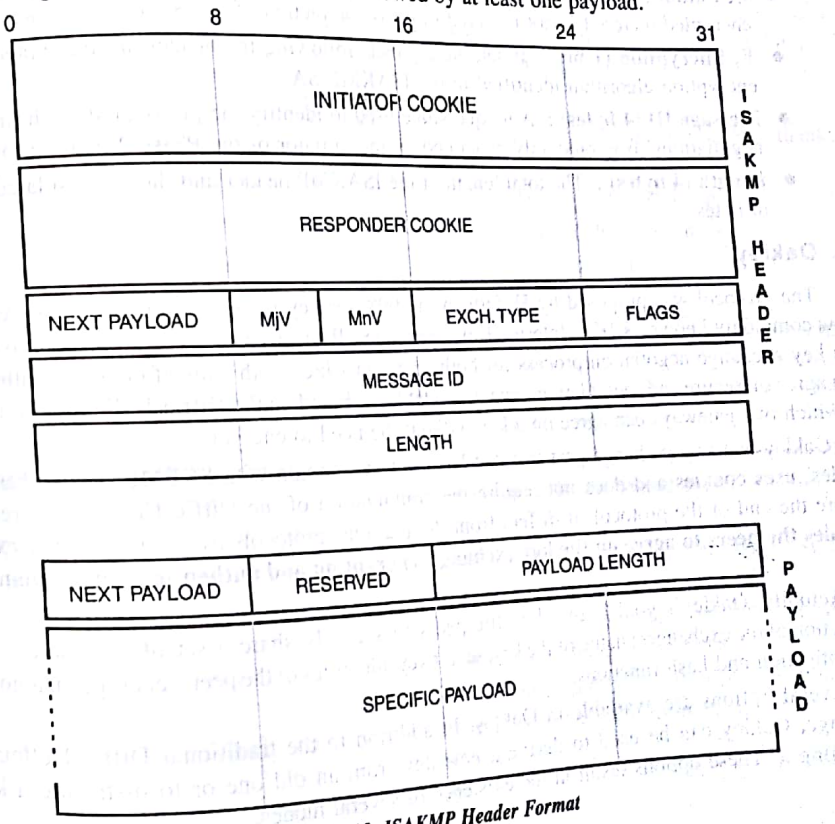


Fig. 11.13. *ISAKMP Header Format*

ISAKMP have the following fields. Description of each field is given below.

- **Initiator cookie (8 bytes) :** The cookie of the entity that initiated SA establishment, SA notification, or SA deletion.
- **Responder cookie (8 bytes) :** The cookie of the entity that is responding to an SA establishment request, SA notification, or SA deletion.
- **Next payload (8 bits) :** Indicates the type of the first payload in the message.
- **MjV (4 bits) :** The major version of the ISAKMP protocol in use.
- **MnV (4 bits) :** The minor version of the ISAKMP protocol in use.
- **Exch. Type (8 bits) :** Indicates the type of exchange being used. This dictates the message and payload orderings in the ISAKMP exchanges.
- **Flags** (8 bits) **:** Indicates the options that are set for the ISAKMP exchange.

00	01	02	03	04	05	06	07
Reserved					A	C	E

- **A, Authentication only (1-bit) :** Intended for use with the Informational Exchange with a Notify payload and will allow the transmission of information with integrity checking, but no encryption.
- **C, Commit (1 bit) :** Used to signal key exchange synchronization. It is used to ensure that encrypted material is not received prior to completion of the SA establishment.
- **E, Encryption (1 bit) :** If set, all payloads following the header are encrypted using the encryption algorithm identified in the ISAKMP SA.
- **Message ID (4 bytes) :** A unique value used to identify the protocol state during Phase 2 negotiations. It is randomly generated by the initiator of the Phase 2 negotiation.
- **Length (4 bytes) :** The total length of the ISAKMP header and the encapsulated payloads in bytes.

2. Oakley

The protocol was proposed by H. Orman in 1998. Oakley is used along side ISAKMP, and is now commonly known as IKE (Internet Key Exchange). Basically Oakley is a protocol to carry out the key exchange negotiation process for both peers, in which both ends after being authenticated can agree on secure and secret keying material. Oakley is based on the Diffie-Hellman key algorithm in which two gateways can agree on a key without the need to encrypt.

Oakley is a key exchange protocol that has a lot in common with SKEME: it also has several modes, uses cookies and does not require the computation of the Diffie-Hellman shared secret before the end of the protocol. It differs from the previous protocols by the fact that **it explicitly enables the peers to agree on the key exchange, encryption and authentication mechanisms to use.**

Actually, Oakley's goal is to allow the peers to securely share a set of parameters for the protection of the exchanges: name of the key, secret key, identities of the peers, encryption algorithms, authentication and hash functions.

Several options are available in Oakley. In addition to the traditional Diffie-Hellman key exchange, Oakley can be used to derive a new key from an old one or to distribute a key by encrypting it. These options result in the existence of several modes.

The main principle of Oakley is that the initiator of the exchanges starts by specifying as much information than he wishes in his first message. His interlocutor answers by also providing as much information than he wishes. The conversation goes on until the required state is reached. The choice of the quantity of information to include in each message depends on the selected options (use of stateless cookies, identity protection, PFS, non-repudiation ...).

The three components of the protocol are:

1. Cookies exchange (optionally stateless),
2. Diffie-Hellman public values exchange (optional),
3. Authentication (options: anonymity, PFS on the identities, non-repudiation).

3. SKEME

It is developed specifically for IPsec, SKEME is an extension of Photuris suggested in 1996 by Hugo Krawczyk of the IBM T. J. Watson Research Center. Contrary to Photuris, which imposes a precise course for the protocol, SKEME provides various modes of key exchange.

Like STS and Photuris, SKEME's basic mode is based on the use of public keys and a Diffie-Hellman shared secret generation. However, SKEME is not restricted to the use of public keys, but also allows the use of a pre-shared key. This key can be obtained by manual distribution or by the intermediatory of a key distribution center (KDC) such as Kerberos. The KDC enables the communicating peers to share a secret by the intermediatory of a trusted third party. The use of this secret for the authentication of the Diffie-Hellman secret and not directly as a session key decreases the necessity of trust in the KDC. Lastly, SKEME also makes it possible to carry out faster exchanges by not using Diffie-Hellman when perfect forward secrecy is not required.

SKEME contains four distinct modes:

- Basic mode, which provides a key exchange based on public keys and ensures PFS thanks to Diffie-Hellman.
- A key exchange based on the use of public keys, but without Diffie-Hellman.
- A key exchange based on the use of a pre-shared key and on Diffie-Hellman.
- A mechanism of fast rekeying based only on symmetrical algorithms.

SKEME is composed of three phases: SHARE, EXCH and AUTH.

- During the SHARE phase, the peers exchange half-keys, encrypted with their respective public keys. These two half-keys are used to compute a secret key K. If anonymity is wanted, the identities of the two peers are also encrypted. If a shared secret already exists, this phase is skipped.
- The exchange phase (EXCH) is used, depending on the selected mode, to exchange either Diffie-Hellman public values or nonces. The Diffie-Hellman shared secret will only be computed after the end of the exchanges.
- The public values or nonces are authenticated during the authentication phase (AUTH), using the secret key established during the SHARE phase.

The messages from these three phases do not necessarily follow the order described above; in actual practice they are combined to minimize the number of exchanged messages.

Another phase, known as the COOKIES phase, can be added before the SHARE phase to provide protection against denial of service attacks thanks to the cookies mechanism introduces by Photuris.

CHAPTER 12

Web Security

12.1. INTRODUCTION

Inside This Chapter

Transport layer provide end-to-end service, so that there is need to secure transport layer data. Suppose a customer use online shopping then he need entity authentication, message integrity and confidentiality.

Two protocols can be used to provide security at transport layer. These are called web security protocols.

(*a*) Secure Sockets Layer (SSL) Protocol

(*b*) Transport layer Security (TLS) Protocol

These protocols provide server and client authentication, data confidentiality and data integrity. If client and server are SSL enabled then use https:// instead of http://. If you write https:// then packet will wrapped in SSL or TLS. For example, credit card number can be transferred via the Internet for online shopping.

12.2. SECURE SOCKET LAYER (SSL)

12.2.1. Introduction

SSL protocol is used for secure exchange of information between a web browser and web server. It provide logical pipe between browser and server. Netscape developed it in 1994. SSL has becomes most popular web security protocol. SSL comes in three version; 2, 3 and 3.1.

Typically, SSL can received data from any application layer protocol, but usually the protocol is HTTP. The data received from the application layer is compressed sign and encrypted. The data is then pass to a reliable protocol such as TCP.

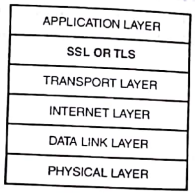


Fig. 12.1. *Location of SSL and TLS*

SSL Connection and SSL Session

Two important SSL concept are the SSL Session and SSL Connection which are define in the specification as follow :

- **Connection :** A connection is a transport that provides suitable type of service. For SSL, such connection is peer-to-peer relationship. The connection is transient. Every connection is associated with one session.
- **Session :** An SSL session is an association between a client and a server. Session is created by the Handshake Protocol. Session define a set of cryptographic security parameters, which can be shared among multiple connections. Sessions are used to avoid the expansive negotiations of new security parameters for each connection.

After a session is establish, the two parties has common information such as the session identifier, the certificate authenticating each of them, the compression method, cipher suit, and a master secret that is used to create keys for message authentication encryption.

A session can consist of many connections. A connection between two parties can be terminated and reestablished within the same session. When a connection is terminated, the two parties can also terminate the session. A session can be suspended and resumed again.

To create a new session, the two parties need to go through a negotiation process. To resume an old session they need to create only a new connection, the two parties escape negotiation process part.

NOTE

In a connection both parties have equal roles, they are peers. In a session, one party has the role of a client and other role of a server.

Session State

A session is define by a session state, a set of parameters establish between server and client. The list of parameters is as follows :

- **Session Identifier :** An arbitrary byte sequence chosen by the server to identify an active or resumable session state.
- **Peer Certificate :** An X.509 version3 certificate of the peer. This element of the state may be NULL.
- **Compression Method :** The algorithm used to compressed data prior to encryption.
- **Cipher Spec :** Specifies the bulk data encryption algorithm and a hash algorithm use for MAC calculation. It also defines cryptographic attributes such as the hash size.
- **Master Secret :** 48 byte secret shared between the client and server.
- **Is resumable :** A flag indicating whether the session can be used to initiate new connection.

Connection State

A connection is define by a connection state, set of parameters established between two peers. A connection state is defined by the following parameters:

- **Server and client random :** Byte sequences that are chosen by the server and client for each connection.
- **Server write MAC secret :** The secret key used in MAC operations on data sent by the server.

- **Client Write MAC Secret :** The secret key used in MAC operations on data sent by the client.
- **Server Write Key :** The conventional encryption key for data encrypted by the server and decrypted by the client.
- **Client Write Key :** The conventional encryption key for data encrypted by the client and decrypted by the server.
- **Initialization vectors :** When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key. This field is first initialized by the SSL Handshake Protocol. Their after the final cipher text block from each record is preserved for use as the IV with the following record.
- **Sequence number :** Each party maintains separate sequence numbers for transmitted and for receive messages for each connection. When a party send or receives a change cipher SPEC message, the appropriate sequence number is set to zero. Sequence number may not exceed $2^{64} - 1$.

Working of SSL

SSL is not a single protocol but rather two layer of protocols, the SSL **Record Protocol** provide basic security services to various higher layer protocols. Three higher layer protocols are define as part of SSL: **Handshake protocol**, **Change Cipher Spec Protocol** and **Alert protocol**. These SSL specific protocols are used in the management of SSL exchanges. The work of each protocol is as follows.

- Messages from the record protocol are pay loads to the transport layer, normally TCP.
- The Handshake protocol provides security parameters for the record protocol. It establishes the cipher set and provides keys and security parameters. It also authenticates the server to the client and the client to the server if needed.
- The ChangeCipherSpec protocol is used for signaling the readiness of cryptographic secrets.
- The alert protocol is used to record abnormal conditions.

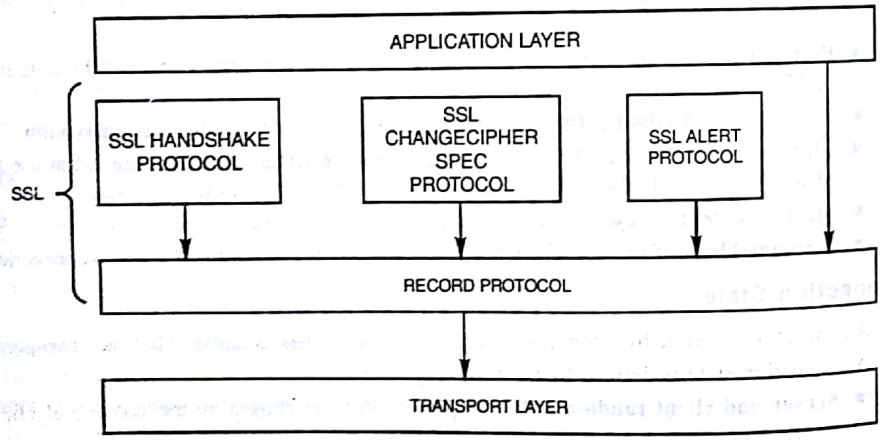


Fig. 12.2. *SSL Protocols Stack*

A. Handshake Protocol

The handshake protocol of SSL is used by client and server to communicate using SSL enabled connection. There are possible message exchange between client and server in handshake protocol as shown in Fig. 12.3.

TYPE	MESSAGE
0	HelloRequest
1	ClientHello
2	ServerHello
11	Certificate
12	ServerKeyExchange
13	CertificateRequest
14	ServerHelloDone
15	CertificateVerify
16	ClientKeyExchange
20	Finished

Fig. 12.3. *SSL Handshake Protocol Message Types*

The Handshake protocol is made up of four phases, as shown in Fig. 12.4.

Phase 1. Establish security capabilities

Phase 2. Server authentication and key exchange

Phase 3. Client authentication and key exchange

Phase 4. Finish or Finalizing handshake protocol

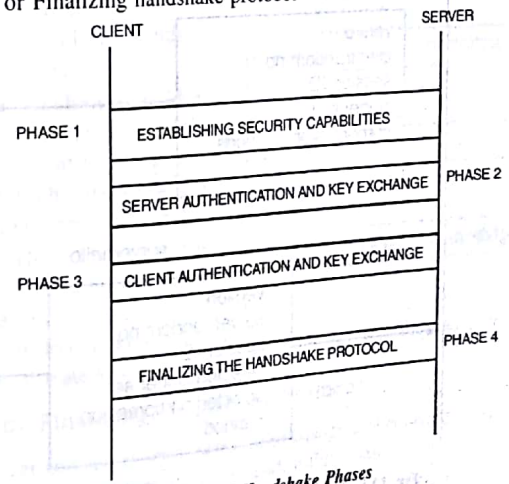


Fig. 12.4. *SSL Handshake Phases*

Now we try to understand each phases.

Phase 1 : Establishing Security Capabilities

In phase 1, the client and server announces their security capabilities and choose those that are convenient for both. In this phase, a session ID is established and the cipher suite is chosen. The parties agreed upon a particular compression method. Finally two random numbers are selected, one by the client and one by server, to be used creating a master secret. This consist of two messages, Client Hello and Server Hello.

Client Hello

The client sends the client hello message. It contains the following:

- The highest SSL version number the client can support.
- A 32-byte random number that will be used for master secret generation.
- A session ID that defines the session.
- A cipher suite that defines the list of algorithms that the client can support.
- A list of compression methods that the client can support.

Server Hello

The server responds to the client with a server hello message. It contains the following :

- An SSL version number. This number is the lower of two version numbers. The highest supported by the client and the highest supposed by the server.
- A 32 byte random number that will be used for master secret generation.
- A session ID that defines the session.
- The selected cipher set from the client list.
- The selected compression method from client list.

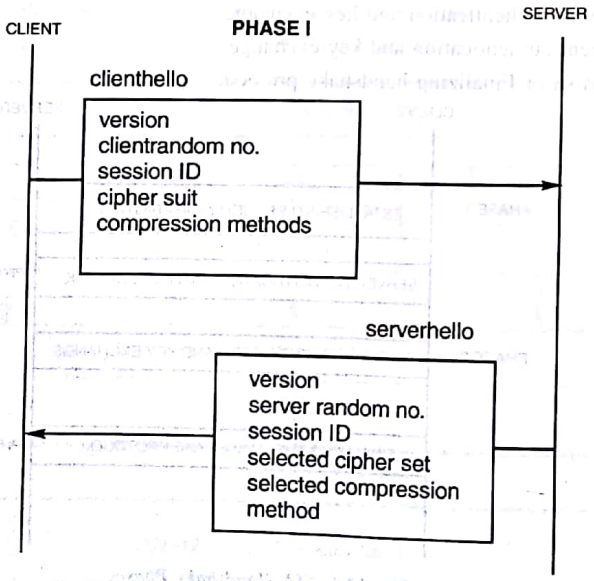


Fig. 12.5. *Details of Handshake Protocol Phase 1*

Phase 2: Server Authentication and Key Exchange

In phase 2, the server authentication itself if needed. The sender may send its certificate, its public key, and may also request certificates from the client at the end, the server announces that the server hello process is done.

It contains the followings :

- **Certificate :** If it is required, the server sends a certificate message to authenticate it.
- **Server Key Exchange :** After the certificate message, the server sends a server key exchange message that include its contribution to the pre master secret.
- **Certificate Request :** The server may require the client to authenticate itself.

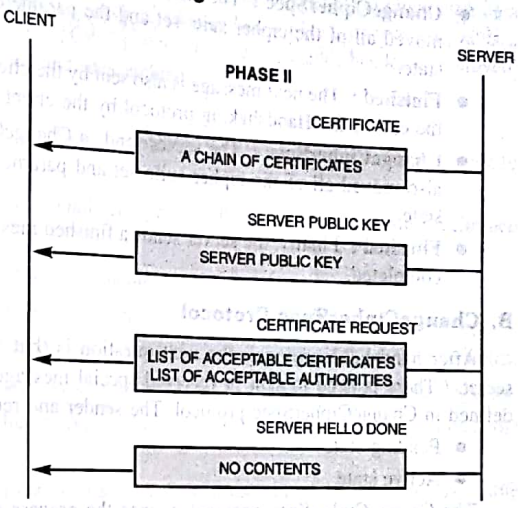


Fig. 12.6. *Details of Handshake Protocol Phase 2*

- **Server Hello Done :** The last message in phase 2 is the server hello done message, which is a signal to the client that phase 2 is over and that the client needs to start phase 3.

Phase 3: Client Authentication and Key Exchange

Phase 3 is designed to authenticate the client. Up to three message can be sent from the client to the server.

- **Certificate :** To certify itself to the server, the client sends a certificate message.
- **Client Key Exchange :** After sending the certificate message, the client send a client key exchange message, which includes its contribution to the pre master secret.
- **Certificate Verify :** If the client has sent a certificate declaring that is own the public key in the certificate, it needs to prove that it knows the corresponding private key.

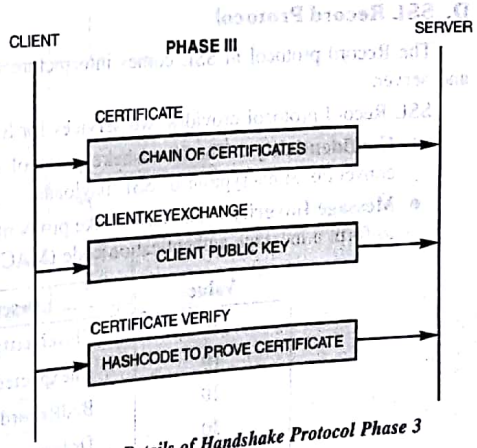


Fig. 12.7. *Details of Handshake Protocol Phase 3*

Phase 4: Finish or Finalizing Handshake Protocol

In phase 4 the client and server send messages to change cipher specification and to finish the Handshaking protocol. Four messages are exchanged in this phase.

- **ChangeCipherSpec :** The client sends a ChangeCipherSpec message to show that is has moved all of the cipher suite set and the parameters from the pending state to the active state.
- **Finished :** The next message is also sent by the client. It is finished message that announces the end of the Handshaking protocol by the client.
- **ChangeCipherSpec :** The server sends a ChangeCipherSpec message to show that it has also moved all of the cipher suite set and parameter from the pending state to the active state.
- **Finished :** Finally, the server sends a finished message to show that Handshaking is totally completed.

B. ChangeCipherSpec Protocol

After handshake protocol, the next question is that when two parties use these parameters secrets? There is need to sent or received special message, the ChangeCipherSpec message. It is defined in ChangeCipherSpec protocol. The sender and receiver need two states.

- Pending state
- Active state

The ChangeCipherSpec protocol defines the process of moving values between pending and active states.

C. Alert Protocol

SSL uses the alert protocol for reporting errors and abnormal conditions. It has only one message type, the Alert message, that describes the problem and its level. Types of Alert messages are shown in Fig. 12.8.

D. SSL Record Protocol

The Record protocol in SSL comes into picture after handshake is completed between client and server.

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).

Value	Description
0	CloseNotify
10	UnexpectedMessage
20	BadRecordMAC
30	DecompressionFailure
40	HandshakeFailure
41	NoCertificate
42	BadCertificate
43	UnsupportedCertificate

Fig. 12.8. *Types of Alerts in SSL*

The record protocol takes an application message to be transmitted. First it fragments the data into manageable blocks, optionally compresses each block, applies a MAC, encrypts it, adds a header, and transmits the resulting unit in a TCP segment. At receiver's end, header is removed; block is decrypted, verified, decompressed and reassembled.

The steps can be understood as follows.

Fragmentation : Original message is broken into blocks. Size of block is less than or equal to 2^{14} bytes.

Compression : Fragmented block is optionally compressed. For compression lossless technique is used.

Addition of MAC : MAC for each block is calculated.

Encryption : The output of previous step is encrypted.

Append Header: At last header is added to encrypted block. It consists of the following fields.

- **Content Type(8 bits) :** The higher layer protocol used to process the enclosed fragment.
- **Major Version (8 bits) :** It indicates major version of SSL in use. For SSL version3, the value is 3.
- **Minor Version (8 bits) :** It indicates minor version in use. For SSL version3 the value is 0.
- **Compressed Length (16 bits) :** The length in bytes of the plaintext fragments. The maximum value is $2^{14} + 2048$.

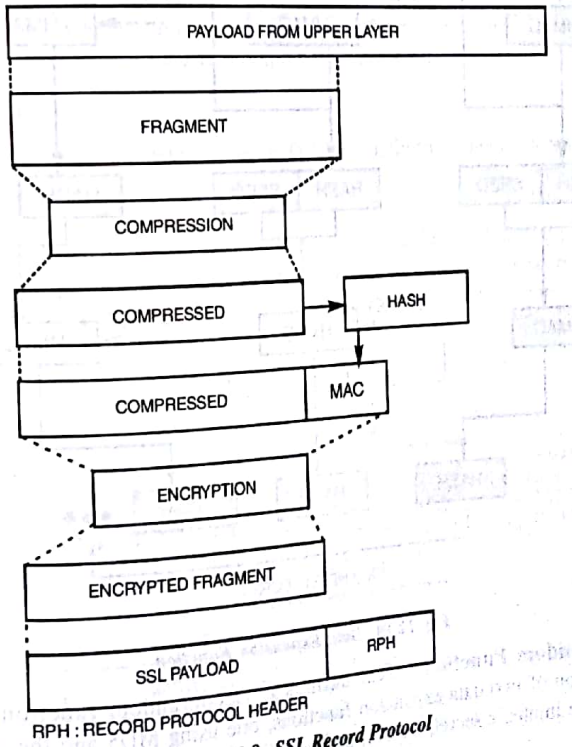


Fig. 12.9. *SSL Record Protocol*

12.3. TRANSPORT LAYER SECURITY (TLS)

The transport layer security (TLS) protocol is the Internet Engineering Task Force (IETF) standard version of the SSL protocol. It is defined in RFC 2246. TLS is quite similar to SSL V3, with minor differences.

Features of TLS

Version : The current version of SSL is 3.0; the current version of TLS is 1.0.

Cipher Suite : TLS does not support FORTEZZA for key exchange or for encryption / decryption.

Generation of Cryptographic Secrets : The generation of secrets is more complex in TLS that in SSL. TLS first defines two functions: the data expansion function and the pseudorandom function. Let us discuss these two functions.

- **Data-Expansion Function :** The data expansion function uses a predefined HMAC to expand a secret into a longer one. This function can be considered a multiple section function, where each section creates one hash value.

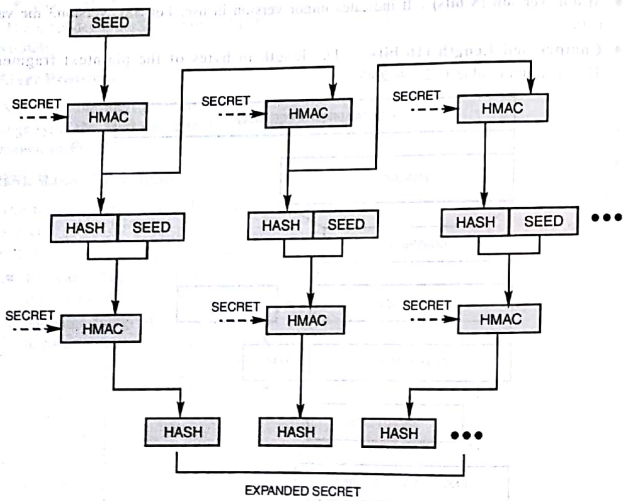


Fig. 12.10. *Data Expansion Function*

- **Pseudorandom Function :** TLS defines a pseudorandom function (PRF) to be the combination of two data expansion functions, one using MD5 and the other SHA-1. PRF takes three inputs, a secret, a label, and seed.

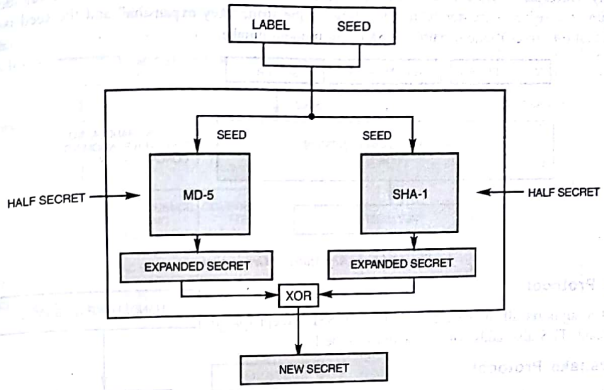


Fig. 12.11. *Pseudorandom Function*

Hashes created from MD5 and SHA-1 are of different sizes, extra sections of MD5-based function must be created to make the two outputs the same size.

Pre Master Secret

The generation of the pre master secret in TLS is exactly the same as in SSL.

Master Secret

TLS uses the PRF function to create the master secret from the pre master secret. This is achieved by using the pre master secret as the secret, the string "master secret as the label, and concatenation of the client random number and server random number as he seed".

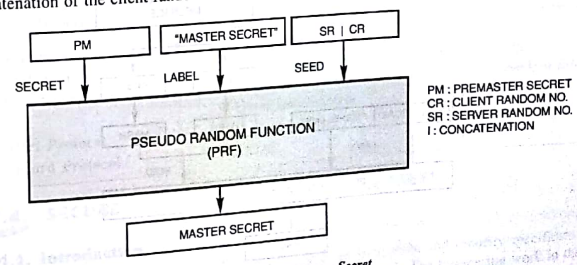


Fig. 12.12. *Master Secret*

Key Material : TLS uses the PRF function to create the key material from the master secret. This time the secret is master secret. The label is the string "key expansion" and the seed is the concatenation server random number and client random number.

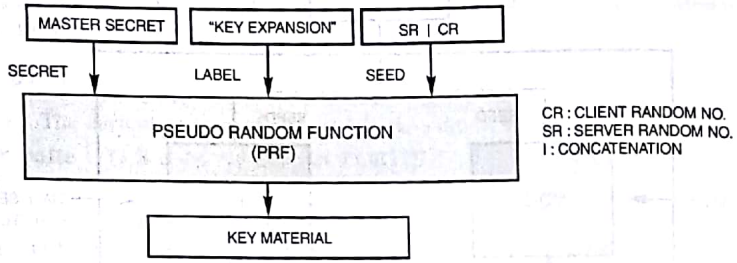


Fig. 12.13(*a*). *Key Material Generation*

Alert Protocol

TLS supports all of the alerts defined in SSL except for no certificate. TLS also adds some new ones to the list.

Handshake Protocol

TLS has made some changes in the Handshake protocol. specifically the detail of the certificate verify message and the finished message have been changed.

Certificate Verify Message

TLS has simplified the process. TLS hash in the TLS is only over the Handshake messages.

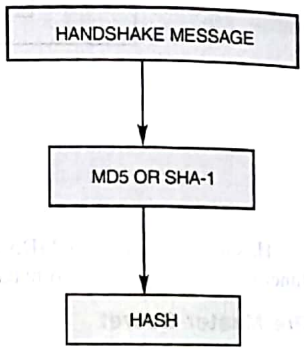


Fig. 12.13(*b*). *Certificate Verify Message*

Finished Message

The calculation of the hash for the finished message has also been changed. TLS uses the PRF to calculate two hashes used for the finished message.

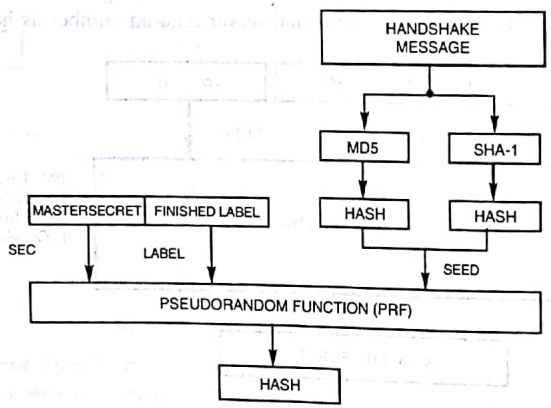


Fig. 12.14. *Finished Message*

Record Protocol : The only change in the record protocol is the use of HMAC for signing the message. TLS is a MAC to create the HMAC. TLS also adds the protocol version to the text to be sign.

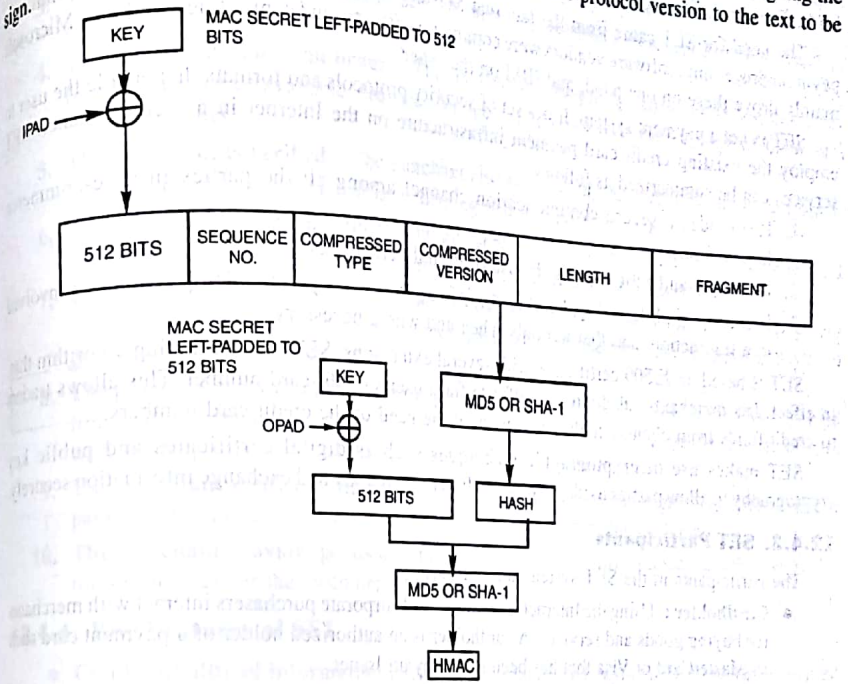


Fig. 12.15. *Record Protocol*

Difference between SSL and TLS

We can summarize the differences between SSL and TLS as shown in table.

Table 12.1. Difference between SSL and TLS

Property	SSL	TLS
Version	3.0	1.0
Cipher Suite	Support algorithm Fortezza	Does not support
Cryptography Secret	Not use pseudorandom function to create master secret	Use pseudorandom function to create master secret
Alert Protocol	Less alert messages	More alert messages
Record Protocol	Use MAC	Use HMAC

12.4. SECURE ELECTRONIC TRANSACTION (SET)

12.4.1. Introduction

The Secure Electronic Transaction (SET) is an open encryption and security specification that is designed for protecting credit card transactions on the internet. The pioneering work in this area

was done in 1996 by MasterCard and Visa jointly. They were joined by IBM, Microsoft, Netscape, RSA, Terisa and VeriSign.

The need for SET came from the fact that MasterCard and Visa realized that for e-commerce payment processing, software vendors were coming up with new and conflicting standards. Microsoft mainly drove these on one hand, and IBM on the other.

SET is not a payment system. It is a set of security protocols and formats that enable the user to employ the existing credit card payment infrastructure on the Internet in a secure manner. SET services can be summarized as follows:

1. It provides a secure communication channel among all the parties in an e-commerce transaction.
2. It provide authentication by the use of digital certificates.
3. It ensures confidentiality, because the information is only available to the parties involved in a transaction, and that too only when and where necessary.

SET is based on X.509 certificates with several extensions. SET uses a blinding algorithm that, in effect, lets merchants substitute a certificate for a user's credit-card number. This allows traders to credit funds from clients' credit cards without the need of the credit card numbers.

SET makes use of cryptographic techniques such as digital certificates and public key cryptography to allow parties to identify themselves to each other and exchange information securely.

12.4.2. SET Participants

The participants in the SET system are as follows:

- **Cardholder :** Using the Internet, consumers and corporate purchasers interact with merchants for buying goods and services. A cardholder is an authorized holder of a payment card such as MasterCard or Visa that has been issued by an Issuer.
- **Merchant :** A merchant is a person or an organization that wants to sell goods or services to cardholders. A merchant must have a relationship with an Acquirer for accepting payment on the internet.
- **Issuer :** The issuer is a financial institution that provides a payment card to a cardholder. The most critical point is that the issuer is the ultimately responsible for the payment of the cardholder's debit.
- **Acquirer :** This is a financial institution that has a relationship with merchants for processing payment card authorizations and payments. The reason for having acquirers is that merchants accept credit cards of more than one brand, but are not interested in dealing with so many bankcard organizations or issuers.
- **Payment Gateway :** This is a task can be taken up by the acquirer or it can be taken up by the organization as a dedicated function. The payment gateway processes the payment messages on behalf of the merchant.
- **Certification Authority (CA) :** This is an authority that is trusted to provide public key certificates to cardholders, merchants and payment gateways. Infact, CAs is very crucial to the success of SET.

12.4.3. SET Process

1. **The customer opens an account :** The customer opens an account with a bank that supports electronic payment mechanisms and the SET protocol.

2. **The customer receives a certificate :** After the customer's identity is verified, the customer receives a digital certificate from a CA (Certificate Authority).
3. **The merchant receives a certificate :** A merchant that wants to accept a certain brand of credit cards must possess a digital certificate.
4. **The customer places an order :** This is a typical shopping cart wherein the customer browses the list of available, searches for specific items, selects one or more of them, and places the order.
5. **The merchant is verified :** The merchant also sends its digital certificate to the customer. This assures the customer that he is dealing with a valid merchant.
6. **The order and payment and details are sent :** The customer sends both the order and payment details to the merchant along with the customer's digital certificate.
7. **The merchant requests payment authorization :** The merchant forwards the payment details sent by the customer to the payment gateway via the acquirer and request the payment gateway to authorize the payment
8. **The payment gateway authorizes the payment :** Using the credit card information received from the merchant, the payment gateway verifies the details of the customer's credit card with the help of issuer, and either authorizes or rejects the payment.
9. **The merchant confirms the order :** Assuming that the payment gateway authorizes the payment, the merchant sends a confirmation of the order of the customer.
10. **The merchant provides goods or services :** The merchant now ships the goods or provides the services as per the customer's order.

12.4.4. 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.
- **Integrity of data :** Payment information sent from cardholder's to merchants includes order information, personnel data and payment instructions. SET guarantees that these message contents are not altered in transit.
- **Cardholder account authentication :** SET enables merchant to verify that a cardholder is a legitimate user of a valid card number. SET uses X.509 V3 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.

12.4.5. SET Objectives

The main concern with online payment mechanisms is that they demand that the customer sends his credit card details to the merchant. There are two aspects of this. One is that the credit card number travels in clear text, which provide an intruder with an opportunity to know that number and use it with malicious intensions (for instance, to make his own payments using that credit card number). The second issue is that the credit card number is available to the merchant who can misuse it.

The way SET hides the cardholder's credit card details from the merchant is very interesting. For this, SET relies on the concept of a digital envelope. The following steps illustrate the idea.

1. The SET software prepares the Payment Information (PI) on the cardholder's computer exactly in the way as it happens in any Web-based payment system.
2. However what is specific to SET is that the cardholder's computer now creates a one time session key.
3. Using this one-time session key, the cardholder's computer now encrypts the payment information.
4. The cardholder's now wraps the one-time session key with the public key of the payment gateway to form a digital envelope.
5. It then sends the encrypted payment information (step 3) and the digital envelope (step 5) together to the merchant.

12.4.6. Working of SET

Let us discuss the major transactions supported by SET. They are three transactions namely.

1. Purchase Request
2. Payment Authorization
3. Payment Capture

1. Purchase Request

Before the Purchase Request exchange begins, the cardholder has completed browsing, selecting, and ordering. The purchase request exchange consists of four messages: Initiate Request, Initiate Response, Purchase Request, and Purchase Response.

(*a*) **Initiate request :** 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 nonce used to ensure timeliness.

(*b*) **Initiate response :** The action is initiated by merchant. Merchant prepares a message and find digital signature of message. Message also contains transaction id created by merchant. This message also includes the merchants signature certificate and the payment gateway's key exchange certificate.

(*c*) **Purchase request :** This is main in SET. The cardholder ensures the identity of the merchant and that of payment gateway. For this the cardholder verifies the digital certificates of merchant and that of payment gateway. The cardholder creates two blocks of information: Order Information (OI) and payment Information (PI).

OI = order information (e.g. which item to be purchased)

PI = Credit card details + expiry + purchase amount.

The message now includes the following :

(*i*) **Purchase related information :** This information will be forwarded to the payment gateway by the merchant and consist of

- The PI.
- The digital signature, calculated over the PI and OI.
- The OI message digest (OIMD).

All these are encrypted with key K. Finally digital envelope is created by encrypting K with payment gateway's public key.

(*ii*) **Order related information :** This information is needed by the merchant and consist 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).

(*iii*) **Cardholder certificate :** This contains the cardholder's public signature key. It is needed by the merchant and by the payment gateway.

Here is an important concept of dual signature. The process is shown in Fig. 12.16.

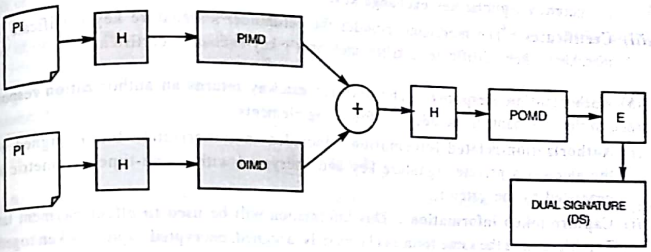


Fig. 12.16. *Dual Signature Process*

(*d*) **Purchase response :** When merchant receives the purchase request. It does the following:

(*i*) Verifies the cardholder certificates by means of CA signatures.

(*ii*) Verifies the signature created over PI and OI using carholder's public key.

(*iii*) Processes the order and forwards the payment information (PI) to payment gateway for authorization.

(*iv*) Sends a purchase response back to cardholder.

2. Payment Authorization

The payment authorization ensures that the transaction was approved by the issuer. During the processing of an order from a cardholder, the merchant authorizes the transaction with the payment gateway. For this merchant sends the payment details to payment gateway. The gateway verifies these details and authorizes the payment. The payment authorization consists of two messages : Authorization Request and Authorization Response.

(*a*) **Authorization Request :** It is prepared by merchant and sent to gateway, which consist of following details.

(*i*) **Purchase related information :** This information was obtained from the customer and consist of:

- The PI.
- The dual signature, calculated over the PI and OI, signed with the customer's private signature key.

13

CHAPTER

System Security

13.1. INTRUDERS

Intruder tries to intrude into the privacy of a network. Whether the network itself is private or public does not matter. What matter is the intent of the attacker, of trying to intrude? Two most widely threats to security are intruders and viruses. Intruder can be of three types as follows.

Inside This Chapter

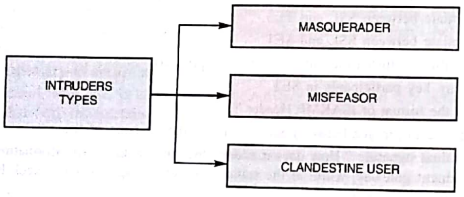


Fig. 13.1. *Types of Intruders*

(1) **Masquerader :** A user who is not authorized to use the computer, but penetrates a system's access control to legitimate user's account.

(2) **Misfeasor :** There are two cases when a user can be misfeasor

- A legitimate user who accesses data, programs, or resources for which such access is not authorized.
- A legitimate user, who has access to some applications, data or resources and misuses these privileges.

(3) **Clandestine User :** An internal or external user who tries to work using the privileges of a supervisor user to avoid auditing information being captured and recorded is called as a clandestine user.

NOTE

The masquerader is likely to be an outsider; the misfeasor is an insider and the clandestine can be insider or outsider.

13.2. INTRUSION DETECTION

Intrusion prevention is almost impossible to achieve at all times. A system's second line of defense is intrusion detection. Intrusion detection is based on assumption that the behaviour of the intruder differs from that of a legitimate user in ways that can be quantified.

Intrusion detection is very important aspect of protecting the cyber infrastructure from terrorist attack or from hackers. Intrusion prevention technique such as firewall, filtering router policies fails to stop much type of attacks. Therefore, no matter how secure you try to make your system. Intrusion still happens and so they must be detected. Intrusion detection systems are becoming an important part of your computer system, and security.

An intrusion detection system is used to detect several types of malicious behaviours that can compromise the security and trust of a computer system. This includes network attacks against vulnerable services, data driven attacks on applications, host based attacks such as privilege escalation, unauthorized logins and access to sensitive files, and malware (viruses, trojan horses and worms).

13.3. INTRUSION DETECTION TECHNIQUES

Intrusion means some one attempting to break or misuse the system. Any set of actions that attempts to compromise the integrity, confidentiality, and availability of resources are that can damage your system or may access the confidential data. Any abuse might have quite different view related to attaching: what is the aim (objective) of attacking? What damages or other consequences are likely to be done? Any intruder can take the advantages of weak point of your system or vulnerabilities exist in targeted (your) system.

There are many techniques of intrusion detection. Some of main approaches are described below.

1. Anomaly Detection
2. Misuse Detection or Signature Detection
3. Combined Anomaly/Misuse detection
4. Pattern Recognition
5. Network Monitoring

1. Anomaly Detection

This technique has been developed to detect abnormal or unusual behaviour (anomalies) on a host or network. They function on the assumption that attacks are different from normal (legitimate) activity and can therefore be detected by systems that identify these differences.

For Example : If a normal user usually logs on to his account 2 times a day then, if in one day he logs 20 times, the system will treat this as an abnormal and considers it as an attack.

These can be of two types :

(*i*) Static (*ii*) Dynamic

(*i*) **Static :** Static means a portion of the system remain constant, *e.g.*, data integrity, tripwire, virus checkers.

(*ii*) **Dynamic :** A profile consists of a set of observed measures of behaviour for each of a set of dimensions. Frequently used dimensions include:

- Preferred choices, *e.g.*, log-in time, log-in location, and favourite editor.
- Resources consumed cumulatively or per unit time.
- Representative sequences of actions.
- Program profiles: system call sequence.

Advantages

- Can detect unusual behaviour and thus have the ability to detect symptoms of attacks without specific knowledge of details.
- Can produce information that can in turn be used to define signatures for misuse detectors.

Disadvantages

- Usually produce a large number of false alarms due to the unpredictable behaviours of users and networks.
- Often require extensive "training sets" of system event records in order to characterize normal behaviour patterns.

2. Misuse Detection or Signature Detection

The second general approach to intrusion detection is misuse detection. This technique involves the comparison of a user's activities with the known behaviours of attackers attempting to penetrate a system. Misuse detection also utilizes a knowledge base of information. The misuse knowledge bases include specific metrics on the various techniques employed by attackers when the knowledge base was created.

While anomaly detection typically utilizes threshold monitoring to indicate when a certain established metric has been reached, misuse detection techniques frequently utilize a rule-based approach. When applied to misuse detection, the rules become scenarios for network attacks. The intrusion detection mechanism identifies a potential attack if a user's activities are found to be consistent with the established rules. The use of comprehensive rules is critical in the application of expert systems for intrusion detection.

Advantage

- Very effective at detecting attacks without generating an overwhelming number of false alarms.

Disadvantages

- Can only detect those attacks they know about—therefore they must be constanly updated with signatures of new attacks.
- Many misuse detectors are designed to use tighly defined signatures that prevent them from detecting variants of common attacks.

3. Combined Anomaly/Misuse detection

Research has also been conducted into intrusion detection methodologies which combine the anomaly detection approach and the misuse detection approach. These techniques seek to incorporate the benefits of both of the standard approaches to intrusion detection. The combined approach permits a single intrusion detection system to monitor for indications of external and internal attacks.

While a significant advantage over the singular use of either method separately, the use of a combined anomaly/misuse detection mechanism does possess some disadvantages.

Disadvantages

- The use of two knowledge bases for the intrusion detection system will increase the amount of system resources which must be dedicated to the system. Additional disk space will be required for the storage of the profiles, and increased memory requirements will be encountered as the mechanism compares user activities with information in the dual knowledge bases.
- In addition, the technique will share the disadvantage of either method individually in it's inability to detect collaborative or extended attack scenarios.

4. Pattern Recognition

One of the few intrusion detection methodologies which have departed from the established use of anomaly and misuse detection profiles is pattern recognition. In this approach, a series of penetration scenarios are coded into the system.

Advantages

- Pattern recognition possesses a distinct advantage over anomaly and misuse detection methods. It is capable of identifying attacks which may occur over an extended period of time, a series of user sessions, or by multiple attackers working in concert.
- This approach is effective in reducing the need to review a potentially large amount of audit data.

Disadvantages

- The key disadvantage of pattern-recognition technique is the reliance of the system on predefined intrusion scenarios. If attack characteristics do not match one which has been coded into the system, the intrusion may not be detected. As a result, pattern-recognition mechanisms are still dependent on a statistical-type of intrusion detection approach to be a truly effective security mechanism.

5. Network Monitoring

A final method of detecting system intrusions which is currently in use is the use of various network monitoring techniques. These methodologies passively monitor network activity for indications of attacks.

Network monitoring offers several advantages over traditional audit-based intrusion detection systems. Because many intrusions occur over network at some point, and because networks are increasingly becoming the targets of attack, these techniques are an excellent method of detecting many attacks which may be missed by audit-based intrusion detection mechanisms.

The greatest advantage of network monitoring mechanisms is their independence from reliance on audit data. Because these methods do not require input from any operating system's audit trail

they can use standard network protocols to monitor heterogeneous sets of operating systems and hosts.

Independence from audit trails also frees network monitoring systems from possessing an inherent weakness caused by the vulnerability of the audit trail to attack. Intruder actions which interfere with audit functions or which modify audit data can lead to the prevention of intrusion detection or the inability to identify the nature of an attack. Network monitors are able to avoid attracting the attention of intruders by passively observing network activity and reporting unusual occurrences.

Another significant advantage of detecting intrusions without relying on audit data is the improvement of system performance which results from the removal of the overhead imposed by audit trails. The process of analyzing audit trails increases the performance degradation of the system. In addition, techniques which move the audit data across network connections reduce the bandwidth available to other functions.

Network monitoring techniques can increase performance of networks by 5 to 20 percent compared to audit based systems.

13.4. INTRUSION DETECTION SYSTEM (IDS)

An IDS does not include preventing the intrusion from occurring, only detecting it and reporting the intrusion to an operator.

An Intrusion detection system monitors the activities of a given environment and decides whether these activities are malicious (intrusive) or legitimate (normal) based on system integrity, confidentiality and the availability of information resources.

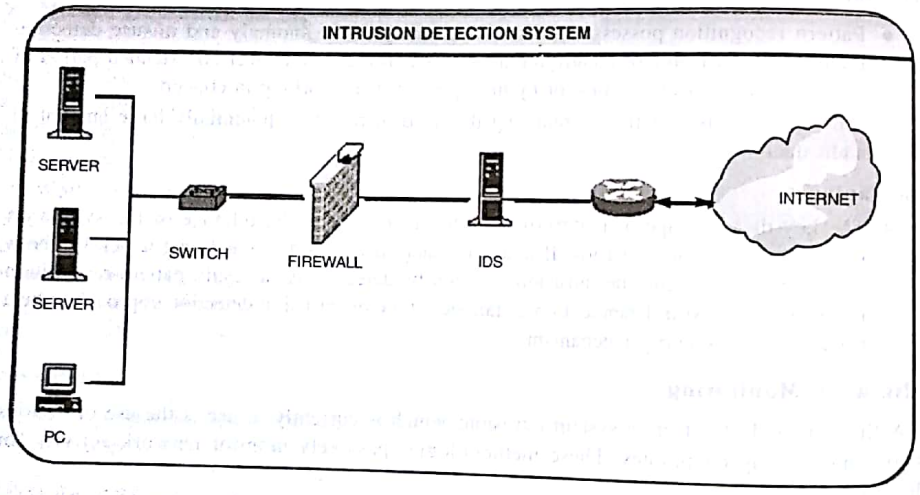


Fig 13.2. *Intrusion Detection System*

In order to properly response to an asset-attack it must first be detected and then all of its characteristics will uncovered and documented. Once the attack or misuse recognized the response can be automatic or manual and can be include termination of connection, or host of response aimed at apprehending the attacker. Any technique such as firewalls, identification, authentication

procedures and encryption are not design with any of above security issues. These particular security issues are the province of Intrusion Detection System.

NOTE

An IDS is a computer program that attempts to perform identification by either misuse or anomaly or combination of techniques.

13.4.1. Terminology

Following terminology are used in IDS.

Alert/Alarm : A signal suggesting that a system has been or is being attacked. These can be of four types:

(*i*) **True Positive :** A legitimate attack which triggers an IDS to produce an alarm.

(*ii*) **False Positive :** An event signaling an IDS to produce an alarm when no attack has taken place.

(*iii*) **False Negative :** A failure of an IDS to detect an actual attack.

(*iv*) **True Negative :** When no attack has taken place and no alarm is raised.

Noise : Data or interference that can trigger a false positive.

Site policy : Guidelines within an organization that control the rules and configurations of IDS.

Site policy awareness : An IDS's ability to dynamically change its rules and configurations in response to changing environmental activity.

Confidence value : A value an organization places on an IDS based on past performance and analysis to help determine its ability to effectively identify an attack.

Alarm filtering : The process of categorizing attack alerts produced from an IDS in order to distinguish false positives from actual attacks.

13.4.2. Types of IDS

For general purpose, the Intrusion Detection System (IDS) has evolved into two major architectures, according to Anomaly or signature, scope:

1. Network-Based Intrusion Detection System (NIDS)

Network based IDS are best suited for alert generation of intrusion from outside the perimeter of the enterprise. The network based IDS are inserted at various points on LAN and observe packets traffic on the Network information is assembled into packets and transmitted on LAN or Internet. N-B IDS are valuable if they placed just outside the firewalls, thereby altering personal to incoming packets that might circumvent the firewall.

Some Network-Based IDS take or allows taking input of "Custom signatures" taken from user's security policy, which permits limited detection security policy violation. This limitation is due to packets traffic information that does not work well today in switched and encrypted environments packets analysis is weak in detecting attacking or misuse originating from authorized Network users.

More detail approach is needed to perform damage assessments and to secure information assets from insider misuse and Intrusion Audit Log Analysis; the audit data generated by OS and application provide solution.

Network-Based Intrusion Detection Systems (IDS) use raw network packets as the data source. The IDS typically uses a network adapter in promiscuous mode that listens and analyses all traffic

in real-time as it travels across the network. A first level filter is usually applied to determine which traffic will be discarded or passed on to an attack recognition module. This first level filter helps performance and accuracy by allowing known un-malicious traffic to be filtered out. An example of this would be if an event for suspicious SNMP get was detected, and a known SNMP management station generated this event. Using Filters SNMP traffic from this machine could be filtered out of the examined traffic. Caution must be taken when using filters as traffic can be spoofed, and misconfigurations can cause more traffic to be filtered than desired. At the attack recognition module, typically on of three methodologies are used for attack signatures; pattern, frequency, or anomaly based detection. Once an attack is detected a response module provides a variety of options to notify, alert, and take action in regards to the attack at hand.

Disadvantages

- NIDS may have difficult processing all packets in a large or busy network and therefore, may fail to recognize an attack launched during periods of high traffic.
- Modern switch-based networks make NIDS more difficult: Switches subdivide networks into many small segments and provide dedicated links between hosts serviced by the same switch. Most switches do not provide universal monitoring ports.
- NIDS cannot analyze encrypted information.
- Most NIDS cannot tell whether or not an attack was successful.

2. Host-based IDS (HIDS)

Host-based IDS placed monitoring "Sensors" also known as "agents" on network resources nodes to monitor audit logs that are generated by Network Operating System or application program. Audit logs contain records for events and activities taking place at individual Network resources. Because this Host-Based IDS can detect attacks that cannot be seen by Network-based IDS. Such as Intrusion and misuse and misuse by trusted insider. Host-based system utilize "Signature rule base", which is derived from site-specific security policy. Host-Based can overcome the problems associated with N/W based IDS immediately after alarming the security personnel can locate the source provided by site–security policy.

Host-based IDS can also verify if any attack was unsuccessful, either because of immediate response to alarm or any other reason but this is not available at packet level. Host-Based IDS can also maintain user login and user logoff action and all activity that generates audit records. Amount of info originating from large no. of sources is time consuming and slower process is the disadvantage of H/B IDS and also Host-Based audit data analysis time consuming. Rapid detection analysis critical for alert generation and effective countermeasure which interrupt attacks in progress and reduce actual damage.

Advantages

- Can detect attacks that cannot be seen by NIDS.
- Can operate in an environment in which network traffic is encrypted.
- Unaffected by switched networks.
- Can help detect Trojan horse or other attacks that involve software integrity breaches.

Disadvantages

- Since at least the information sources reside on the host targeted by attacks, the IDS may be attacked and disabled as port of the attack.

- Are not well suited by detecting network scans or other such surveillance that targets an entire network.
- Since they use the computing resources of the hosts they are monitoring, therefore inflicting a performance cost on the monitored systems.

The drawbacks of Host Intrusion Detection Systems (HIDS) are :

- Difficult to analyse the intrusion attempts on multiple computers.
- Host Intrusion Detection Systems (HIDS) can be very difficult to maintain in large networks with different operating systems and configurations
- Host Intrusion Detection Systems (HIDS) can be disabled by attackers after the system is compromised.

Difference between Host IDS and Network IDS

The Key difference between Host-based and Network-based IDS is that—N/W based IDS although run on single host, is responsible for entire network segment while Host-based IDS is only responsible for host on which it resides. Other differences are shown in table.

Table 13.1. Difference between Host IDS and Network IDS

Benefit	Host	Network
Deterrence	Strong deterrence for insiders.	Strong deterrence for outstiders.
Detection	Strong insider detection. Weak outsider detection.	Strong outsider detection. Weak insider detection.
Response	Weak real-time response. Good for long-term attacks.	Strong response against outsiders attacks.
Damage Assessment	Excellent for determining extent of compromise.	Very weak damage assessment capabilities.
Attack Anticipation	Good at trending and detecting suspicious behaviour patterns.	None.
Prosecution Support	Strong prosecution support capabilities.	Very weak because there is no data source integrity

3. Active IDS

An active Intrusion Detection Systems (IDS) is also known as Intrusion Detection and Prevention System (IDPS). Intrusion Detection and Prevention System (IDPS) is configured to automatically block suspected attacks without any intervention required by an operator. Intrusion Detection and Prevention System (IDPS) has the advantage of providing real-time corrective action in response to an attack.

4. Passive IDS

A passive IDS is a system that's configured to only monitor and analyze network traffic activity and alert an operator to potential vulnerabilities and attacks. A passive IDS is not capable of performing any protective or corrective functions on its own.

5. Knowledge-based (Signature-based) IDS

A knowledge-based (Signature-based) Intrusion Detection Systems (IDS) references a database of previous attack signatures and known system vulnerabilities. The meaning of word signature, when we talk about Intrusion Detection Systems (IDS) is recorded evidence of an intrusion or attack. Each intrusion leaves a footprint behind (*e.g.*, nature of data packets, failed attempt to run an application, failed logins, file and folder access etc.). These footprints are called signatures and can be used to identify and prevent the same attacks in the future. Based on these signatures Knowledge-based (Signature-based) IDS identify intrusion attempts.

Disadvantages

- Signature database must be continually updated and maintained.
- It may fail to identify unique attacks.

6. Behaviour-based (Anomaly-based) IDS

A Behavior-based (Anomaly-based) Intrusion Detection Systems (IDS) references a baseline or learned pattern of normal system activity to identify active intrusion attempts. Deviations from this baseline or pattern cause an alarm to be triggered. Higher false alarms are often related with Behaviour-based Intrusion Detection Systems (IDS).

13.4.3. IDS Strengths and Limitations

Up Side

- Detect an ever-growing number of serious problems
- New signatures are added.
- New methods are being developed.

Down Side

- IDs look for known weaknesses (patterns or normal behaviour)
- False positive

13.5. VIRUSES AND RELATED THREATS

The most types of threats to computer systems are presented by program. The program which threats the vulnerabilities in computing system is called malicious program or Malware.

13.5.1. Malicious Programs or Malware

These threats can be divided into two categories:

- Needs host program
- Independent

Need host program : These types of programs can not exist independently. There is need of host program to invoke.

Independent : These are self-contained programs that can be scheduled and run by operating system. When executed, may produce one or more copies of itself.

Types of malicious program or Malware are shown in Fig. 13.3.

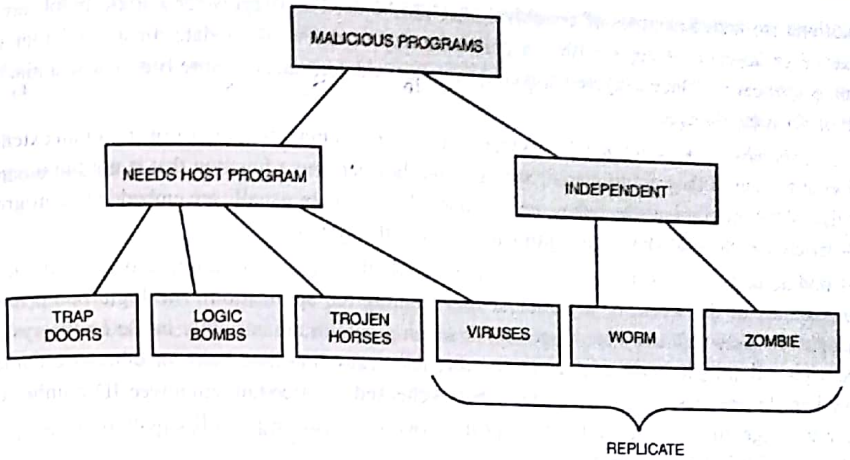


Fig. 13.3. *Malicious Programs/Malware*

1. Trap doors/Back doors

Back doors, also called trap doors. A trap door is a secret entry point into a program that allow someone that is aware of the trapdoor to gain access without going through the usual security access procedures. Trap doors have been used legitimately for many years by programmers to debug and test programs. Trap doors become threats when they are used by unscrupulous programmers to gain unauthorized access. It is difficult to implement operating system controls for trap doors. Security measures must focus on the program development and software update activities.

Most back doors are inserted into applications that require lengthy authentication procedures, or long setups, requiring a user to enter many different values to run the application. When debugging the program, the developer may wish to gain special privileges, or to avoid all the necessary setup and authentication steps. The programmer also may 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 back door is code that either recognizes some special sequence of input, or is triggered by being run from a certain user ID. It then grants special access.

Back doors become threats when they're used by unscrupulous programmers to gain unauthorized access. They are also a problem when the initial application developer forgets to remove a back door after the system has been debugged and some other individual discovers the door's existence.

Perhaps the most famous UNIX back door was the DEBUG option of the sendmail program, exploited by the Internet worm program in November of 1988. The DEBUG option was added for debugging sendmail. Unfortunately, the DEBUG option also had a back door in it, which allowed remote access of the computer over the network without an initial login. The DEBUG option was accidentally left enabled in the version of the program that was distributed by Sun Microsystems, Digital Equipment Corporation, and others.

2. Logic Bomb

One of the oldest types of program threat, predating viruses and worms, is the logic bomb. The logic bomb is coded 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 application. Once triggered, a bomb may alter or delete data or entire files, cause a machine halt, or do some damage.

Logic bombs are programmed threats that lie dormant in commonly used software for an extended period of time until they are triggered; at this point, they perform a function that is not the intended function of the program in which they are contained. Logic bombs usually are embedded in programs by software developers who have legitimate access to the system.

Conditions that might trigger a logic bomb include the presence or absence of certain files, a particular day of the week, or a particular user running the application. The logic bomb might check first to see which users are logged in, or which programs are currently in use on the system. Once triggered, a logic bomb can destroy or alter data, cause machine halts, or otherwise damage the system. In one classic example, a logic bomb checked for a certain employee ID number and then was triggered if the ID failed to appear in two consecutive payroll calculations (*i.e.*, the employee had left the company).

Time-outs are a special kind of logic bomb that are occasionally used to enforce payment or other contract provisions. Time-outs make a program stop running after a certain amount of time unless some special action is taken. The SCRIBE text formatting system uses quarterly time-outs to require licensees to pay their quarterly license fees.

Protect against malicious logic bombs in the same way that you protect against back doors: don't install software without thoroughly testing it and reading it. Keep regular backups so that if something happens, you can restore your data.

3. 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 permission so that the files are readable by any user.

Trojan horses are named after the Trojan horse of myth. Analogous to their namesake, modern-day Trojan horses resemble a program that the user wishes to run - a game, a spreadsheet, or an editor. While the program appears to be doing what the user wants, it actually is doing something else unrelated to its advertised purpose, and without the user's knowledge. For example, the user may think that the program is a game. While it is printing messages about initializing databases and asking questions like "What do you want to name your player?" and "What level of difficulty do you want to play?" the program may actually be deleting files, reformatting a disk, or otherwise altering information. All the user sees, until it's too late, is the interface of a program that the user is trying to run. Trojan horses are, unfortunately, as common as jokes within some programming environments. They are often planted as cruel tricks on bulletin boards and circulated among individuals as shared software.

The best way to avoid Trojan horses is to never execute anything, as a program or as input to an interpreter, until you have carefully read through the entire file. When you read the file, use a program or editor that displays control codes in a visible manner. If you do not understand what the file does, do not run it until you do. And never, ever run anything as root unless you absolutely must.

4. Zombie or Bot

A 'bot' is a type of Malware which allows an attacker to gain complete control over the affected computer. Computers that are infected with a 'bot' are generally referred to as 'zombies'. There are literally tens of thousands of computers on the Internet which are infected with some type of 'bot' and don't even realize it. Attackers are able to access lists of 'zombie' PC's and activate them to help execute DoS (denial-of-service) attacks against Web sites, host phishing attack Web sites or send out thousands of spam email messages. Should anyone trace the attack back to its source, they will find an unwitting victim rather than the true attacker.

Identifying A 'Zombie' Computer

'Bots' are good at hiding in the shadows of your computer so that they are not noticed. If you could easily detect that something was running on your computer, you would quickly remove or disable it. They often have file and process names that are similar, or even identical, to normal system file names and processes so that users won't think twice even if they do see them.

5. Viruses

A virus is a program 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. A computer virus carries in its instructional code the recipe for making perfect copies of itself. The infection can be spread from computer to computer by unsuspecting users, who either swap disks or send programs to another over a network. In a network environment, the ability to access applications and systems services on other computers provides a perfect for the spread of a virus.

Phases of Viruses

Virus goes through the following four phases in its lifetime.

- **Dormant phases :** Here, the virus is idle. It gets activated based on certain action or event, such as a date, the presence of another program or file.
- **Propagation phase :** In this phase, a virus copies itself and each copies starts creating more copies of self, thus propagating the virus.
- **Triggering phase :** A dormant virus moves into this phase when the action/event for which it was waiting is initiated.
- **Execution phase :** This is actual work of the virus, which could be harmless, such as display message on screen or destructive such as delete a file from disk.

Types of Viruses

There are many types of viruses; here we describe main of them.

- **Parasitic virus :** This is the most common form of viruses. Such a virus attaches itself to executable files and keeps replicating. Whenever the infected file is executed, the virus looks for other executable files to attach itself and spread.

- **Memory-resident virus :** This type of virus first attached itself to an area of main memory and then infects every executable program that is executed.
- **Boot Sector viruses :** These viruses infect master boot records in hard disks. They replace the boot record program (which is responsible for loading the operating system in memory) copying it elsewhere on the disk or overwriting it. Boot viruses load into memory if the computer tries to read the disk while it is booting.

  Examples : Form, Disk Killer, Michelangelo, and Stone virus
- **Program viruses :** These infect executable program files, such as those with extensions like .BIN, .COM, .EXE, .OVL, .DRV (driver) and .SYS (device driver). These programs are loaded in memory during execution, taking the virus with them. The virus becomes active in memory, making copies of itself and infecting files on disk.

  Examples : Sunday, Cascade
- **Multipartite viruses :** A hybrid of Boot and Program viruses. They infect program files and when the infected program is executed, these viruses infect the boot record. When you boot the computer next time the virus from the boot record loads in memory and then starts infecting other program files on disk.

  Examples : Invader, Flip, and Tequila
- **Stealth viruses:** These viruses use certain techniques to avoid detection. They may either redirect the disk head to read another sector instead of the one in which they reside or they may alter the reading of the infected file's size shown in the directory listing. For instance, the Whale virus adds 9216 bytes to an infected file; then the virus subtracts the same number of bytes (9216) from the size given in the directory.

  Examples : Frodo, Joshi, Whale
- **Polymorphic viruses:** A virus that can encrypt its code in different ways so that it appears differently in each infection. These viruses are more difficult to detect.

  Examples : Involuntary, Stimulate, Cascade, Phoenix, Evil, Proud, Virus 101
- **Metamorphic virus :** In addition to change its identity on every execution like polymorphic virus, this type of virus keeps rewriting itself every time, making its detection even harder.
- **Macro viruses :** A macro virus is a new type of computer virus that infects the macros within a document or template. When you open a word processing or spreadsheet document, the macro virus is activated and it infects the Normal template (Normal.dot)—a general purpose file that stores default document formatting settings. Every document you open refers to the Normal template, and hence gets infected with the macro virus. Since this virus attaches itself to documents, the infection can spread if such documents are opened on other computers.

  Examples : DMV, Nuclear, Word Concept.
- **Active X :** ActiveX and Java controls will soon be the scourge of computing. Most people do not know how to control there web browser to enable or disable the various functions like playing sound or video and so, by default, leave a nice big hole in the security by allowing applets free run into there machine. There has been a lot of commotion behind this and with the amount of power that JAVA imparts, things from the security angle seem a bit gloom.

6. Worms

A worm is actually different in implementation. A virus modifies a program *i.e.*, it attaches itself to the program under attack. A worm however does not modify a program. A worm replicates itself again and again. The replication grows and at last system slow down, finally coming to a halt.

Difference between Worms and Viruses

S.No.	Worm	Viruses
1	It replicate itself	It does not replicate itself.
2	It does not modify program	It modify the program.
3	It consumes resources to bring it down	It destruct the system.

13.6. FIREWALL

A firewall is a device (usually a router or a computer) installed between the internal network of an organization and rest of the Internet. It is designed to forward some packets and filter others packets.

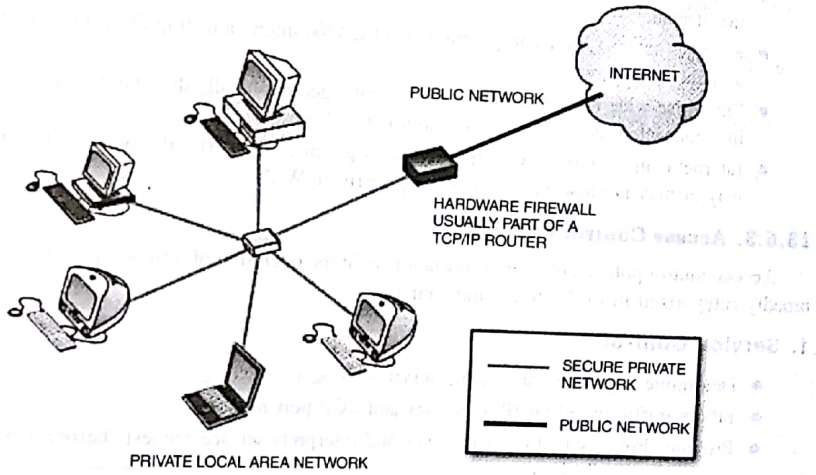


Fig. 13.4. *Firewall*

13.6.1. Why a Firewall is needed?

There is no need for a firewall if each and every host of a private network is properly secured. Unfortunately, in practice the situation is different. A private network may consist of different platforms with diverse OS and applications running on them. Many of the applications were designed and developed for an ideal environment, without considering the possibility of the existence of bad

guys. Moreover, most of the corporate networks are not designed for security. Therefore, it is essential to deploy a firewall to protect the vulnerable infrastructure of an enterprise.

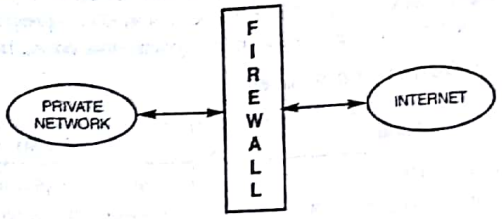


Fig. 13.5. *Firewall*

13.6.2. 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 network (LANs) interconnecting PCs and terminals to each other and the mainframe.
- Premises network, consisting a number of LANs, interconnecting PCs, servers, and perhaps a mainframe or two.
- Enterprise-wide network, consist 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.

13.6.3. Access Control Policies

Access control policies play an important role in the operation of a firewall. The policies can be broadly categorized in to the following four types:

1. Service Control

- Determines the types of internet services to be accessed.
- Filters traffic based on IP addresses and TCP port numbers.
- Provides Proxy servers that receives and interprets service requests before it is passed on.

2. Direction Control

Determines the direction in which a particular service request may be initiated and allowed to flow through the firewall.

3. User Control

- Controls access to a service according to which user is attempting to access it.
- Typically applied to the users inside the firewall perimeter.
- Can be applied to the external users too by using secure authentication technique.

4. Behavioural Control

- Controls how a particular service is used.
- For example, a firewall may filter email to eliminate spam.
- Firewall may allow only a portion of the information on a local web server to an external user.

13.6.4. Firewall Characteristics

The design goals for a firewall are following:

1. All traffic from inside to outside, and vice versa, must pass through the firewall.
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.
3. The firewall itself is immune to penetration. This implies that use of a trusted system with a secure operating system.

13.6.5. Firewall Capabilities

The following capabilities are within the scope of a firewall:

1. A firewall defines a single 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.
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.
4. A firewall can serve as the platform for IPSec.

13.6.6. Types of Firewalls

There are three types of firewalls:

1. Packet filter firewall
2. Application level gateway or Proxy firewall
3. Circuit level gateway

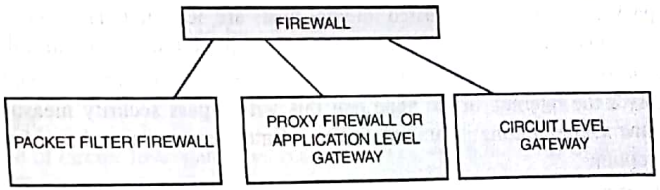


Fig. 13.6. *Types of Firewall*

1. Packet-Filter Firewall

A firewall can be used as a packet filter. It works at network or transport layer. It can forward or block packets based on information in network layer and transport layer headers:

- Source and destination IP addresses

- Source and destination port addresses
- Type of protocol (TCP or UDP)

A packet-filtering is a router that uses a filtering table to decide which packets must be discarded. 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.
- **Destination IP address :** The IP address of the system the IP packet is trying to reach.
- **Source and destination transport-level address :** The transport level (*e.g.*, TCP and UDP) port number, which defines applications such as SNMP and 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 destinated for.

Advantages

One advantage of a packet filtering router is its simplicity. Also packet filters typically are transparent to user's and very fast.

Disadvantages

The weaknesses of Packet filter firewalls are following :

- Because packet filter firewalls do not examine upper layer data, they cannot prevent attacks that employ application-specific vulnerabilities or functions.
- Because of the limited information available to the firewall, the logging functionality present in packet filter firewall is limited.
- Most packet filter firewalls do not support advanced user authentication schemes.
- 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.

Attacks

Some of the attacks that can be made on packet filtering routers and the appropriate counter measures are the following:

- **IP address spoofing :** The intruder transmits packets from outside with a source IP address field containing an address of the 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 internal interface.
- **Source routing attacks :** The source station specifies the route that the packet should take as it crosses the internet, in the hope 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.

2. Application-Level Gateway or Proxy Firewall

It works at application layer. The packet filter firewall is based on information available in network layer and transport layers headers. However sometimes we need to filter a message based on information in message itself.

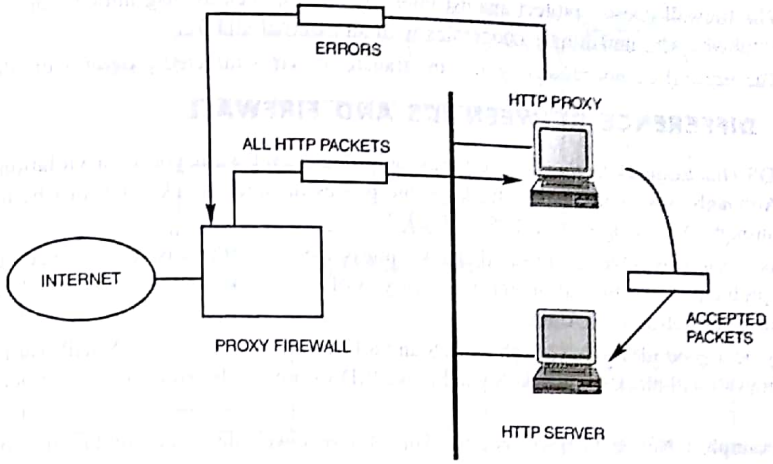


Fig. 13.7. *Proxy Firewall*

For Example : Suppose an organization wants to implement the policies regarding web pages: only those Internet user who have previously relationship with company can have access; access to other users must be blocked. In this case a packet filter is not feasible. Testing must be done at application level (using URLs).

An application-level gateway, also called a proxy server, acts as a relay of application-level traffic. 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 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 end points.

Advantage : Application-level gateways tend to be more secure than packet filters.

Disadvantage : A prime disadvantage of this type of gateway is the additional processing overhead on each connection.

3. Circuit-Level Gateway

A third type of firewall is the circuit-level gateway. 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 b/w itself and a TCP user on an inner host and one b/w itself a TCP user on an outside host.

A typical use of circuit-level gateways is a situation in which the system administrator trusts the internal users. The gateway configured to support application-level or proxy service on inbound connections and circuit-level functions for outbound connections.

13.6.7. Limitation of Firewall

Firewalls have their limitations, include the following:

1. The firewall cannot protect against attacks that bypass the firewall. Internal system may have dial-out capability to connect to an ISP.

2. The firewall doesn't protect against internals threats, such as 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.

13.7. DIFFERENCE BETWEEN IDS AND FIREWALL

An IDS *(Intrusion Detection System)* may only detect and warn you of a violation of your privacy. Although most block major attacks, some probes or other attacks may just be noted and allowed through. An example of an IDS is Black Ice.

A good firewall will block almost all attacks unless specified otherwise or designed otherwise. The only problem is, the firewall might not warn you of the attacks and may just block them. An example of a firewall is ZoneAlarm.

It may be a good idea to have both an IDS and a Firewall, because the IDS will warn you and then the firewall will block the attack. Some Firewall/IDS's are combined into one internet security program.

For example : Norton Internet Security. This is a very well designed combination of both a firewall and IDS.

13.8. TRUSTED SYSTEMS

One way to enhance the ability of a system to defend against intruders and malicious programs is to implement trusted system technology. We begin by looking at some basic concepts of data access control.

Data Access Control : After successful logon, the user has been granted access to one or a set of hosts and applications. Associated with each user, there can be a profile that specifies permissible operations and file accesses.

A general model of access control as exercised by an file or database management system is that of an access matrix.

The basic elements of the model are as follows :

- **Subject :** An entity capable of accessing objects. The concept of subject equates with that of process.
- **Object :** Anything to which access is controlled. For example, files, programs and segments of memory.
- **Access Right :** The way in which an object is accessed by a subject. For example, read, write and execute.

Access Matrix : Access matrix consist of identified subjects and objects that may be accessed. In practice, an access matrix is sparse and is implemented by decomposition in one of two ways :

(i) **Decomposed by columns :** It yields access control lists. Thus for each object, an access control list users and their permitted access rights.

(ii) **Decomposed by rows :** It yields capability tickets. It specifies authorized objects and operations for a user.

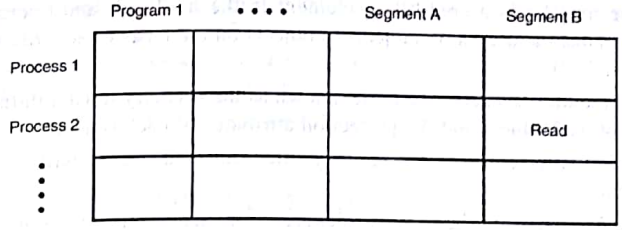


Fig. 13.8. *Access Matrix*

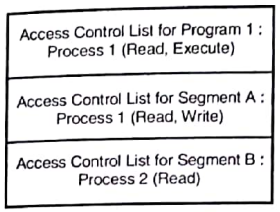


Fig. 13.9. *Access Control List*

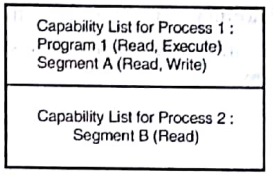


Fig. 13.10. *Capability List*

The Concept of Trusted Systems : Protection of data or resources on the basis of levels of security. This is commonly found in military, where information is divided as

Unclassified (U), Confidential (C), Secret (S), Top Secret (TS)

When multiple categories or levels of data are defined. The requirement is referred to as *multilevel security*. A multilevel secure system must enforce :

- **No read up :** A subject can only read an object of less or equal security level.
- **No write down :** A subject can only write into an object of greater or equal security level.

These two rules, it properly enforced, provide multilevel security.

For a data processing systems, the the approach that has been taken, and has been the object of much research and development, is based on **reference monitor concept.** This is shown in Fig. 13.11.

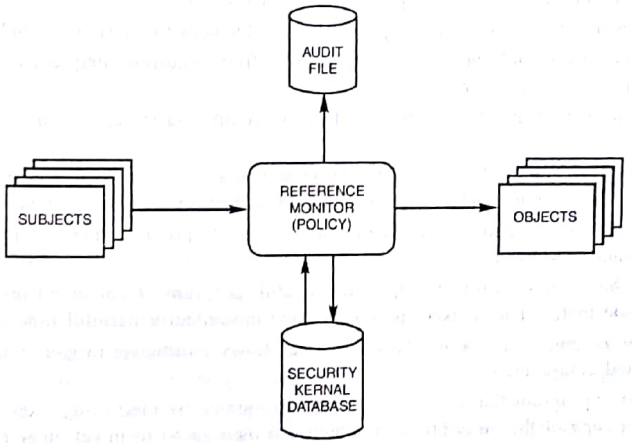


Fig. 13.11. *Reference Monitor Concept*

The reference monitor is a controlling element in the hardware and operating system of a computer that regulates the access of subjects to objects on the basis of security parameters of the subject and object.

The reference monitor has access to a file, known as the **security kernel database,** that list the access privileges of each object and the protection attributes of each object.

The reference monitor enforces the security rules (no read up, no write down) and has the following properties :

- **Complete Mediation :** The security rules are enforced on every access, not just, for example, when a file is opened.
- **Isolation :** The reference monitor and database are protected from unauthorized modification.
- **Verifiability :** The reference monitor's correctness must be provable.

There are stiff requirements. A system that can provide such verification is referred to as a ***trusted system.***

SUMMARY

- Intruder tries to intrude into the privacy of a network.
- Intruder can be of three types Masquerader, Misfeasor and Clandestine User.
- Intrusion prevention is almost impossible to achieve at all times.
- Intrusion prevention technique such as firewall, filtering router policies fails to stop much type of attacks.
- Intrusion means some one attempting to break or misuse the system
- There are many techniques to intrusion detection. Some of main approaches are Anomaly Detection, Misuse Detection or Signature Detection, Combined Anomaly/Misuse detection, Pattern Recognition and Network Monitoring.
- Anomaly Detection has been developed to detect abnormal unusual behaviour (anomalies) on a host or network.
- Misuse Detection or Signature Detection involves the comparison of a user's activities with the known behaviours of attackers attempting to penetrate a system.
- Pattern recognition is a series of penetration scenarios are coded into the system.
- Network monitoring methodologies passively monitor network activity for indications of attacks.
- An IDS does not include preventing the intrusion from occurring, only detecting it and reporting the intrusion to an operator.
- The program which threats the vulnerabilities in computing system is called malicious program or Malware.
- A trap door is a secret entry point into a program that allow someone that is aware of the trapdoor to gain access without going through the usual security access procedures.
- The logic bomb is coded embedded in some legitimate program that is set to "explode" when certain conditions are met.
- 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.
- A 'bot' or Zombie is a type of Malware which allows an attacker to gain complete control over the affected computer.
- A virus is a program that can "infect" other programs by modifying them. The modification includes a copy of the virus program, which can then