KARNATAKA STATE OPEN UNIVERSITY

MUKTHAGANGOTRI, MYSORE- 570 006

DEPARTMENT OF STUDIES IN INFORMATION TECHNOLOGY

M.Sc IN INFORMATION TECHNOLOGY

IV SEMESTER

ELECTIVE 2

CRYPTOGRAPHY AND NETWORK SECURITY

MSIT- 121C

MODULE: 1-4

MSIT-121C (Elective 2): Cryptography and Network Security

Course Design and Editorial Committee

Prof. M.G.Krishnan
Vice Chancellor
Karnataka State Open University
Mukthagangotri, Mysore – 570 006

Prof. Vikram Raj Urs
Dean (Academic) & Convener
Karnataka State Open University
Mukthagangotri, Mysore – 570 006

Head of the Department and Course Co-Ordinator

Rashmi B.S
Assistant Professor & Chairperson
DoS in Information Technology
Karnataka State Open University
Mukthagangotri, Mysore – 570 006

Course Editor

Ms. Nandini H.M
Assistant professor of Information Technology
DoS in Information Technology
Karnataka State Open University
Mukthagangotri, Mysore – 570 006

Course Writers

Dr. Lalitha Rangarajan
Associte Professor,
DoS in Computer Science,
Manasagangothri,
University of Mysore, Mysore

Dr. Sharada
Assistant Professor,
DoS in Computer Science,
Manasagangothri,
University of Mysore, Mysore

Publisher

Registrar
Karnataka State Open University
Mukthagangotri, Mysore – 570 006

Developed by Academic Section, KSOU, Mysore
Karnataka State Open University, 2014

Printed and Published on behalf of Karnataka State Open University, Mysore-6 by the **Registrar (Administration)**

Karnataka State Open University

Muktagangothri, Mysore – 570 006

Master of Science in Information Technology

MSIT – 121C Cryptography and Network Security

Preface

Protection of sensitive information from adversaries has been a practice since centuries. As children, many of us had magic decoder rings for exchanging coded messages with our friends and possibly keeping secrets from parents, siblings or teachers. Sensitive information will be passed on between allies at war times. Cryptographic methods were traditionally used to prevent the enemy from learning sensitive military information. As society has evolved, the need for more sophisticated methods of protecting data has increased.

The study of techniques needed to protect information is the field of Cryptography/Cryptology. Cryptanalysis deals with the methods required for breaking the protection. Of course this is impossible without thorough understanding of the cryptographic method used for protection.

The course material has 4 modules. Module 1 begins with recent trends in security, overview of attacks and threats, standards in security services, mechanisms to provide security services. These topics are discussed in first two units of the module .Later half of module 1 details two important methods of protecting data namely, substitution and transposition.

Units 1 and 2 of module 2 describe an important cipher method called DES (Data Encryption Standard) and attacks specially devised for this method. In unit 3 variants of DES is discussed .The module concludes with stream ciphers in unit 4.

Asymmetric encryption is the topic of discussion in module 3.This covers the necessary background mathematics, public key crypto systems, RSA method, exchange of keys and digital signatures in various units. The last module of the material touches upon, internet security, web security, password management, malicious software and firewalls.

All topics are described in simple terms keeping in mind the target audience who are basically self-learners. Examples are provided, whenever possible to make concepts clear. Readers are encouraged to refer to original texts given in the references at the end of each unit. Answering questions and solving problems given in each unit will make learning thorough and complete.

Authors of the material welcome your comments and suggestions. We hope you will enjoy reading the material and wish you happy learning.

MODULE 1

INTRODUCTION

AND

CLASSICAL ENCRYPTION TECHNIQUES

UNIT -1: SECURITY TRENDS AND ATTACKS

Structure

1.0 OBJECTIVES

After studying this unit, you will be able to discuss

- ✓ Variety of security violations
- ✓ Importance of increased security
- ✓ Challenges in providing security
- ✓ Various models in security
- ✓ Primary goals of security
- ✓ Trends in security
- ✓ Two types of security attacks

1.1 SECURITY OF INFORMATION

We are living in the information age. We need to keep information about every aspect of our lives. In other words, information is an asset that has a value like any other asset. As an asset information needs to be secured from attacks.

To be secured, information needs to be hidden from unauthorized access (confidentiality), protected from unauthorized change (integrity), and available to an authorized entity when it is needed (availability).

Until a few decades ago, the information collected by an organization was stored on physical files. The confidentiality of the file was achieved by restricting the access to a few authorized and trusted people in the organization. In the same way, only a few authorized people were allowed to change the contents of the files. Availability was achieved by designating atleast one person who would have access to the files at all times.

With the advent of computers, information storage became electronic. Instead of being stored on physical media, it was stored in computers. The three security requirements however, did not change. The files stored in computers require confidentiality, integrity and availability. The implementation of these requirements, however, is different and more challenging.

The second major change that affected security is the introduction of distributed systems and the use of networks and communications facilities for carrying data between terminal user and computer and between computer and computer. Network security measures are needed to protect data during their transmission. In fact, the term network security is somewhat misleading, because virtually all business, government, and academic organizations interconnect their data processing equipment with a collection of interconnected networks. Such a collection is often referred to as an internet, and the term **internet security** is used.

There are no clear boundaries between these two forms of security. For example, one of the most publicized types of attack on information systems is the computer virus. A virus may be introduced into a system physically when it arrives on a diskette or optical disk and is subsequently loaded onto a computer. Viruses may also arrive over an internet. In either case,

once the virus is resident on a computer system, internal computer security tools are needed to detect and recover from the virus.

1.2 NEW THREATS

Computing systems are the assets to attackers. Today computers are very powerful, work at unimaginable speed and at very high accuracy. With computers we now have new concerns namely automated attacks, privacy breach, ease of theft etc.

Automating attacks

The speed of computers makes several attacks worthwhile. For example, in the real world, suppose that someone manages to create a machine that can produce counterfeit coins, would that not bother authorities? It certainly would. However, producing so many coins on a mass scale may not be that much economical compared to the return on that investment! How many such coins would the attacker be able to get into the market so rapidly? This is quite different with computers. They are quite efficient and happy in doing routine, mundane and repetitive tasks. For example, they would excel in somehow stealing a very low amount (say half a dollar or Rupees 20) from a million bank accounts in a matter of few minutes. This would give the attacker half a million dollars possibly without any major complaints!

Privacy concerns

Collecting information about people and later misusing it is turning out to be a huge problem, these days. The so called data mining applications gather process and tabulate all sorts of details about individuals. People can then illegally sell this information. For example, companies like Experian (formerly TRW), TransUnion and Equifax maintain credit history of individuals in the USA. Similar trends are seen in the rest of the world. These companies have volumes of information about a majority of citizens of that country. These companies can collect, collate, polish and format all sorts of information to whosoever is ready to pay for that data! Examples of information that can come out of this are: which store the person buys more from, which restaurant she eats in, where she goes for vacations frequently and so on! Every company (Eg. Shop keepers, banks, airlines, insurers) is collecting and processing a mind boggling amount of information about us, without we realizing when and how it is going to be used.

Distance does not matter

Thieves would earlier attack banks, because banks had money. Banks do not have money today! Money is in digital form inside computers and moves around by using computer networks. Therefore, a modern thief would perhaps not like to wear a mask and attempt a robbery! Instead it is far easier and cheaper to attempt an attack on the computer system of the bank, sitting at home! It may be far prudent for the attacker to break into the bank's servers or steal credit card or ATM information from the comforts of her home or place of work.

In 1995, A russian hacker broke into Citibank's computers remotely, stealing $12 million. Although the attacker was traced, it was very difficult to get him extradited for the court case**.**

1.3 EXAMPLES OF SECURITY VIOLATIONS

Here some common examples of violations on security particularly on information transmitted through a network.

1. User A transmits a file to user B. The file contains sensitive information (e.g., payroll records) that is to be protected from disclosure. User C, who is not authorized to read the file, is able to monitor the transmission and capture a copy of the file during its transmission.
2. A network manager, D, transmits a message to a computer, E, under its management. The message instructs computer E to update an authorization file to include the identities of a number of new users who are to be given access to that computer. User F intercepts the message, alters its contents to add or delete entries, and then forwards the message to E, which accepts the message as coming from manager D and updates its authorization file accordingly.
3. Rather than intercept a message, user F constructs its own message with the desired entries and transmits that message to E as if it had come from manager D. Computer E accepts the message as coming from manager D and updates its authorization file accordingly.
4. An employee is fired without warning. The personnel manager sends a message to a server system to invalidate the employee's account. When the invalidation is accomplished, the server is to post a notice to the employee's file as confirmation of the action. The employee is able to intercept the message and delay it long enough to make a final access to the server to

retrieve sensitive information. The message is then forwarded, the action taken, and the confirmation posted. The employee's action may go unnoticed for some considerable time.

5. A message is sent from a customer to a stockbroker with instructions for various transactions. Subsequently, the investments lose value and the customer denies sending the message.

Although this list by no means exhausts the possible types of security violations, it illustrates the range of concerns of network security.

1.4 CHALLENGES IN SECURITY

Internetwork security is complex and at the same time fascinating. Here we highlight some challenges in providing security.

1. Security involving communications and networks is not as simple as it might first appear to the novice. The requirements seem to be straightforward; indeed, most of the major requirements for security services can be given self-explanatory one-word labels: confidentiality, authentication, nonrepudiation, integrity. But the mechanisms used to meet those requirements can be quite complex, and understanding them may involve rather subtle reasoning.
2. In developing a particular security mechanism or algorithm, one must always consider potential attacks on those security features. In many cases, successful attacks are designed by looking at the problem in a completely different way, therefore exploiting an unexpected weakness in the mechanism.
3. Because of point 2, the procedures used to provide particular services are often counterintuitive: It is not obvious from the statement of a particular requirement that such elaborate measures are needed. It is only when the various countermeasures are considered that the measures used make sense.
4. Having designed various security mechanisms, it is necessary to decide where to use them. This is true both in terms of physical placement (e.g., at what points in a network are certain security mechanisms needed) and in a logical sense [e.g., at what layer or layers of an architecture such as TCP/IP (Transmission Control Protocol/Internet Protocol) should mechanisms be placed].

5. Security mechanisms usually involve more than a particular algorithm or protocol. They usually also require that participants be in possession of some secret information (e.g., an encryption key), which raises questions about the creation, distribution, and protection of that secret information. There is also a reliance on communications protocols whose behavior may complicate the task of developing the security mechanism. For example, if the proper functioning of the security mechanism requires setting time limits on the transit time of a message from sender to receiver, then any protocol or network that introduces variable, unpredictable delays may render such time limits meaningless.

Thus, there is much to consider. This chapter provides a general overview of the subject matter that structures the material in the remainder of the book. We begin with a general discussion of network security services and mechanisms and of the types of attacks they are designed for. Then we develop a general overall model within which the security services and mechanisms can be viewed.

1.5 SECURITY MODELS

An organization can take several approaches to implement its security model. Let us summarize these approaches.

No Security: In this simplest case, the approach could be a decision to implement no security at all.

Security through obscurity: In this model, a system is secure simply because nobody knows about its existence and contents. This approach cannot work for too long, as there are many ways an attacker can come to know about it.

Hot Security: In this scheme, the security for each host is enforced individually. This is a very safe approach, but the trouble is that it cannot scale well. The complexity and diversity of modern sites/organizations makes the task even harder.

Network Security: Host security is tough to achieve as organizations grow and become more diverse. In this technique, the focus is to control network access to various hosts and their services, rather than individual host security. This is a very efficient and scalable model.

1.6 SECURITY GOALS

There are three primary goals in any security service. These are confidentially, integrity and availability.

Confidentiality

The principle of confidentiality is that only the sender and the intended recipient should be able to access the contents of a message. Confidentiality gets compromised if an unauthorized person is able to access the message. Example of this could be a confidential email message sent by user A to user B, which is accessed by user C without the permission or knowledge of A and B. This type of attack is called interception.

Integrity

When the contents of a message are changed after the sender sends it, but before it reaches the intended recipient, we say that the integrity of the message is lost. For example, consider that user A sends message to user B. User C tampers with a message originally sent by user A, which is actually destined for user B. User C somehow manages to access it, change its contents and send the changed message to user B. User B has no way of knowing that the contents of the message changed after user A had sent it. User A also does not know about this change. This type of attack is called modification.

Availability

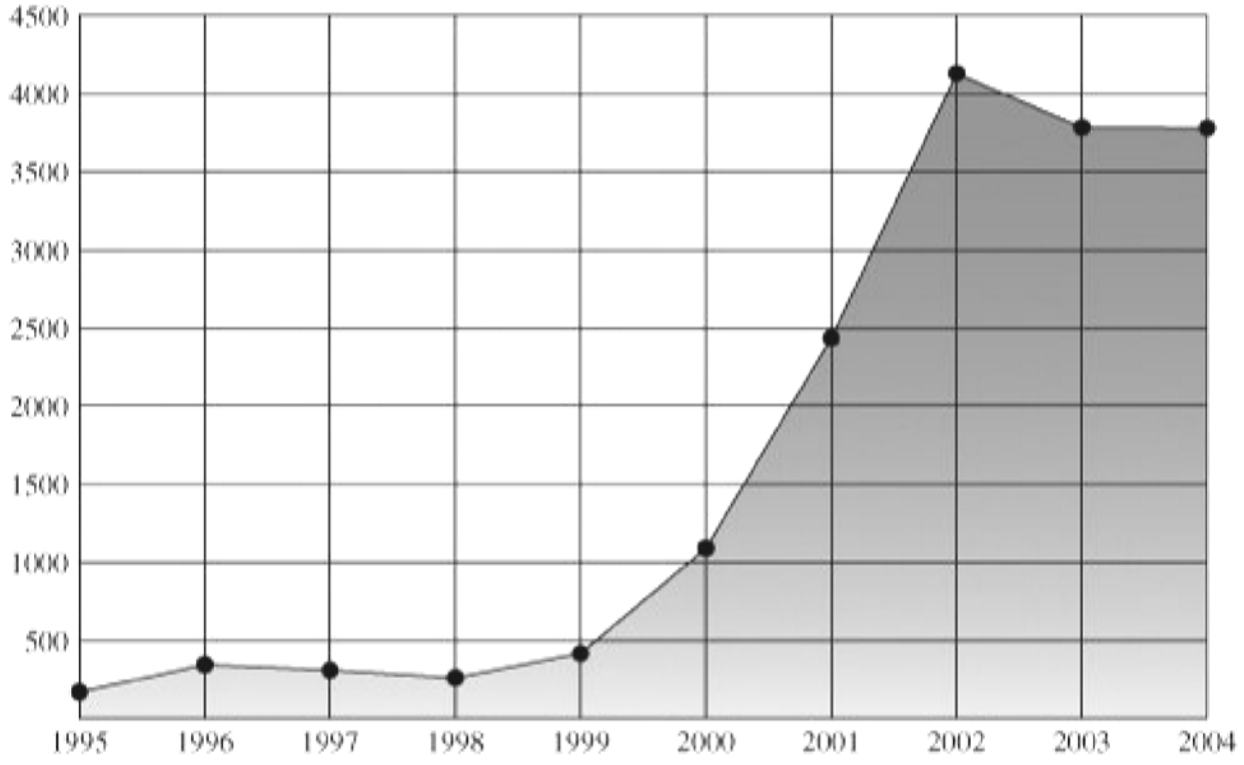
The principle of availability is that resources should be available to authorized parties at all times. For example, due to the intentional actions of an unauthorized user C, an authorized user A may not be able to contact a server B. This would defeat the principle of availability. Such an attack is called interruption.

1.7 SECURITY TRENDS

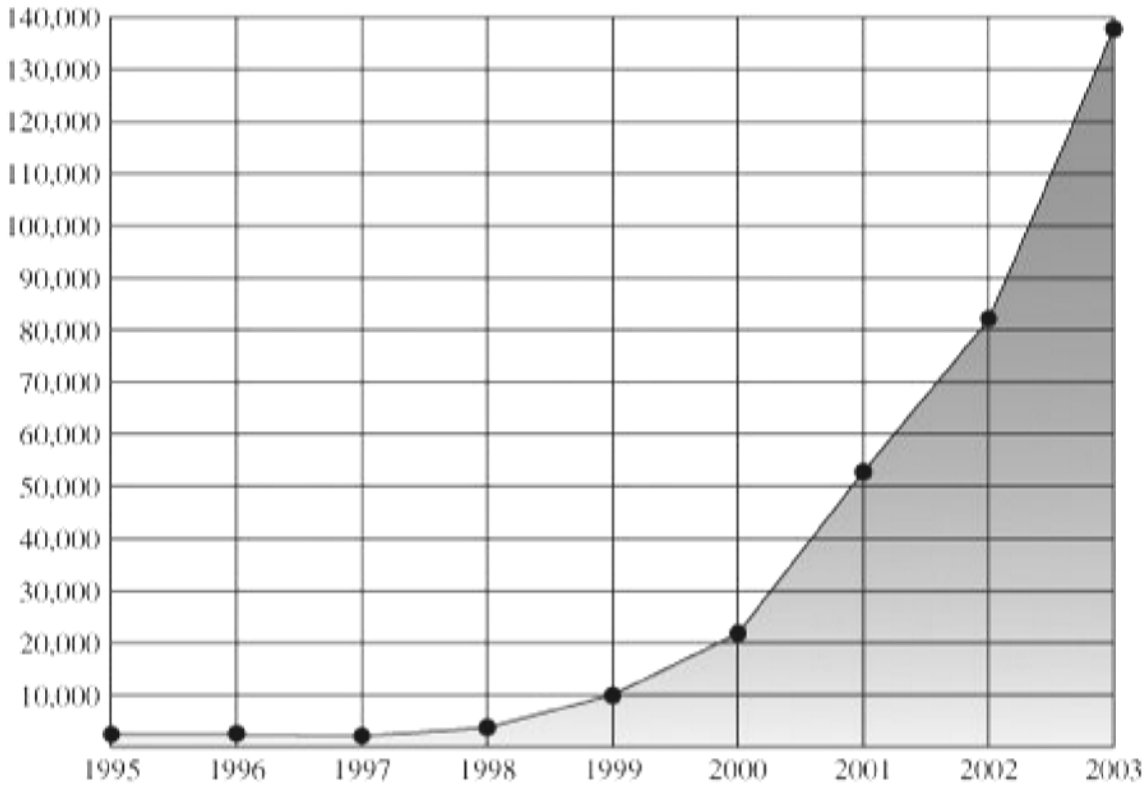
Internet Architecture Board (IAB) has issued report entitled “Security in the Internet Architecture” where they have identified key areas for security mechanisms. Among these were

1. need to secure network and
2. need to secure end to end transmission

These concerns are fully justified. As confirmation, consider the trends reported by the Computer Emergency Response Team (CERT) Coordination Center (CERT/CC). Figure 1.1 a shows the trend in Internet-related vulnerabilities reported to CERT over a 10-year period. These include security weaknesses in the operating systems of attached computers (e.g., Windows, Linux) as well as vulnerabilities in Internet routers and other network devices. Figure 1.1 b shows the number of security-related incidents reported to CERT. These include denial of service attacks; IP spoofing, in which intruders create packets with false IP addresses and exploit applications that use authentication based on IP; and various forms of eavesdropping and packet sniffing, in which attackers read transmitted information, including logon information and database contents.



(a) Vulnerabilities reported



(b) Incidents reported

Figure 1.1: CERT Statistics

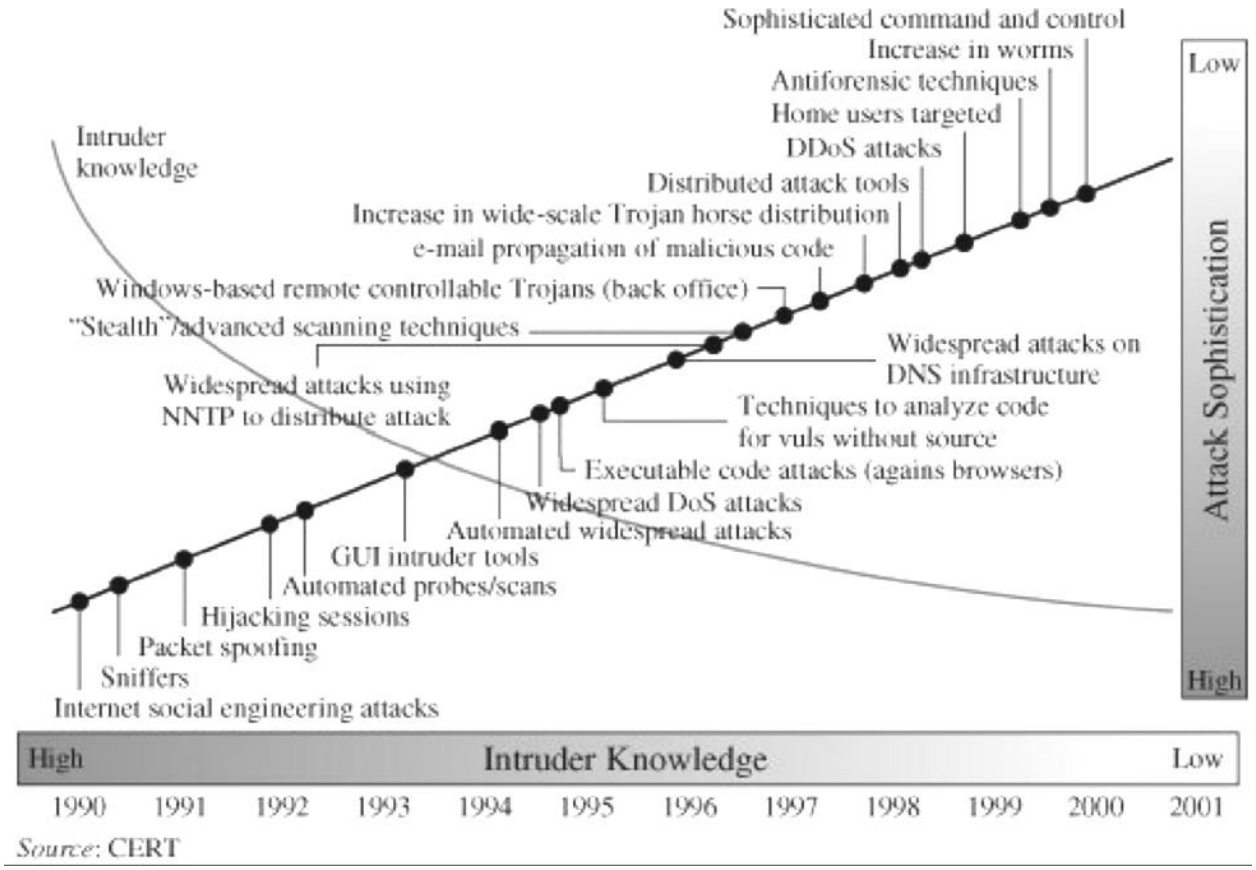


Figure 1.2: Trends in Attack Sophistication and Intruder Knowledge

Over time, the attacks on the Internet and Internet-attached systems have grown more sophisticated while the amount of skill and knowledge required to mount an attack has declined (Figure 1.2). Attacks have become more automated and can cause greater amounts of damage.

This increase in attacks coincides with an increased use of the Internet and with increases in the complexity of protocols, applications, and the Internet itself. Critical infrastructures increasingly rely on the Internet for operations. Individual users rely on the security of the Internet, email, the Web, and Web-based applications to a greater extent than ever. Thus, a wide range of technologies and tools are needed to counter the growing threat. At a basic level, cryptographic algorithms for confidentiality and authentication assume greater importance. As

well, designers need to focus on Internet-based protocols and the vulnerabilities of attached operating systems and applications.

1.8 OSI SECURITY ARCHITECTURE

The OSI security architecture focuses on security attacks, mechanisms, and services. These can be defined briefly as follows:

Security attack

Security attack is any action that compromises the security of information owned by an organization.

Security mechanism

A process (or a device incorporating such a process) that is designed to detect, prevent, or recover from a security attack.

Security service

A processing or communication service that enhances the security of the data processing systems and the information transfers of an organization. The services are intended to counter security attacks, and they make use of one or more security mechanisms to provide the service.

In the literature, the terms *threat* and *attack* are commonly used to mean more or less the same thing. However RFC 2828 (RFC: Request For Comment- is a security standard) differentiates threat and attack

Threat

Threat is a potential for violation of security, which exists when there is a circumstance, capability, action, or event that could breach security and cause harm. That is, a threat is a possible danger that might exploit vulnerability.

Attack

Attack is an assault on system security that derives from an intelligent threat; that is, an intelligent act that is a deliberate attempt (especially in the sense of a method or technique) to evade security services and violate the security policy of a system.

1.9 TYPES OF ATTACKS

Attacks are classified as passive and active. A passive attack is an attempt to learn or make use of information from the system without affecting system resources; whereas an active attack is an attempt to alter system resources or affect their operation.

Passive Attacks

Passive attacks are in the nature of eavesdropping on, or monitoring of, transmissions. The goal of the opponent is to obtain information that is being transmitted. Two types of passive attacks are release of message contents and traffic analysis.

The **release of message contents** is easily understood (Figure 1.3 a). A telephone conversation, an electronic mail message, and a transferred file may contain sensitive or confidential information. We would like to prevent an opponent from learning the contents of these transmissions.

A second type of passive attack, **traffic analysis**, is subtler (Figure 1.3 b). Suppose that we had a way of masking the contents of messages or other information traffic so that opponents, even if they captured the message, could not extract the information from the message. The common technique for masking contents is encryption. If we had encryption protection in place, an opponent might still be able to observe the pattern of these messages. The opponent could determine the location and identity of communicating hosts and could observe the frequency and length of messages being exchanged. This information might be useful in guessing the nature of the communication that was taking place.

Passive attacks are very difficult to detect because they do not involve any alteration of the data. Typically, the messages are sent and received in seemingly normal fashion. Neither the sender nor receiver is aware that a third party has read the messages or observed the traffic pattern. However, it is feasible to prevent the success of these attacks. Message encryption is a simple solution to thwart passive attacks. Thus, the emphasis in dealing with passive attacks is on prevention rather than detection.

Active Attacks

Active attacks involve some modification of the data stream or the creation of a false stream and can be subdivided into four categories: masquerade, replay, modification of messages, and denial of service.

Replay involves the passive capture of a data unit and its subsequent retransmission to produce an unauthorized effect (Figure 1.4 a).

A **masquerade** takes place when one entity pretends to be a different entity (Figure 1.4 b). A masquerade attack usually includes one of the other forms of active attack. For example, authentication sequences can be captured and replayed after a valid authentication sequence has taken place, thus enabling an authorized entity with few privileges to obtain extra privileges by impersonating an entity that has those privileges.

Modification of messages simply means that some portion of a legitimate message is altered, or that messages are delayed or reordered, to produce an unauthorized effect (Figure 1.4 c). For example, a message meaning "Allow John Smith to read confidential file *accounts*" is modified to mean "Allow Fred Brown to read confidential file *accounts*."

The **denial of service** prevents or inhibits the normal use or management of communications facilities (Figure 1.4 d). This attack may have a specific target; for example, an entity may suppress all messages directed to a particular destination (e.g., the security audit service). Another form of service denial is the disruption of an entire network, either by disabling the network or by overloading it with messages so as to degrade performance.

Passive attacks

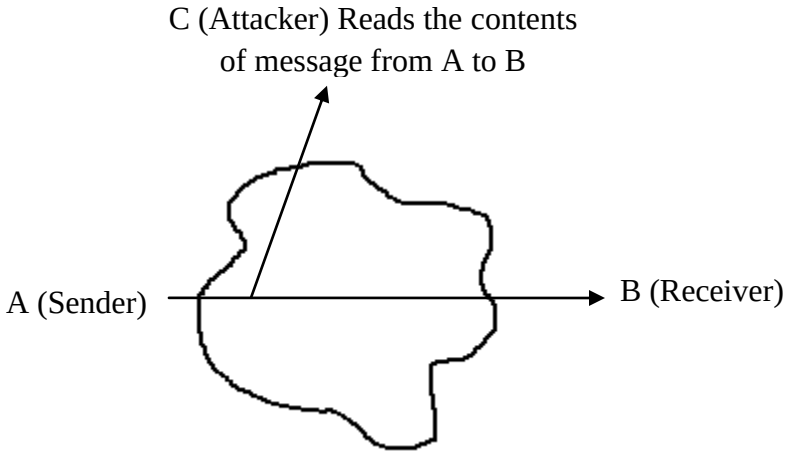


Fig 1.3 a: Release of message

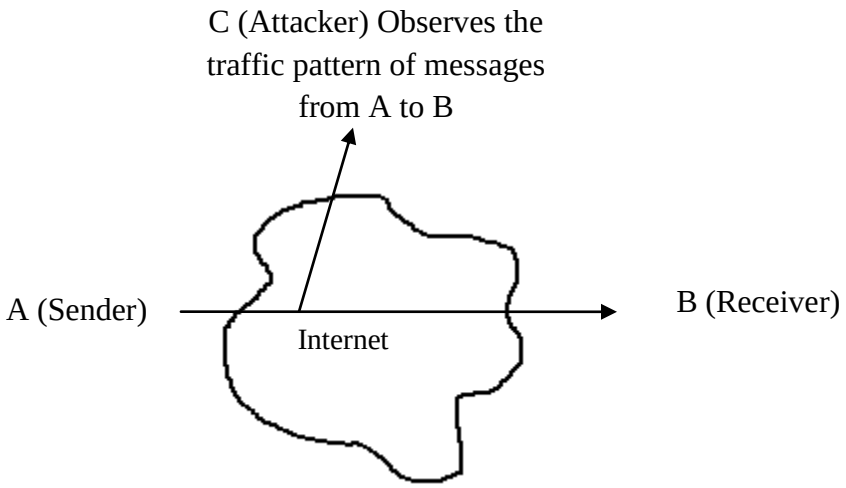


Fig 1.3 b: Traffic analysis

Active attacks

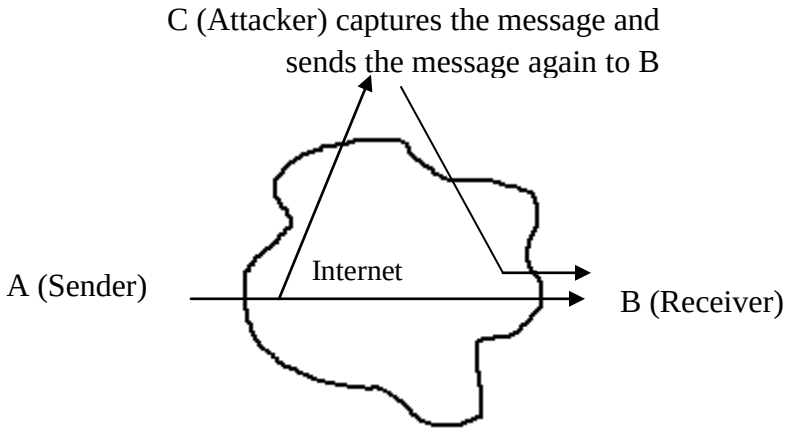


Fig 1.4 a: Replay

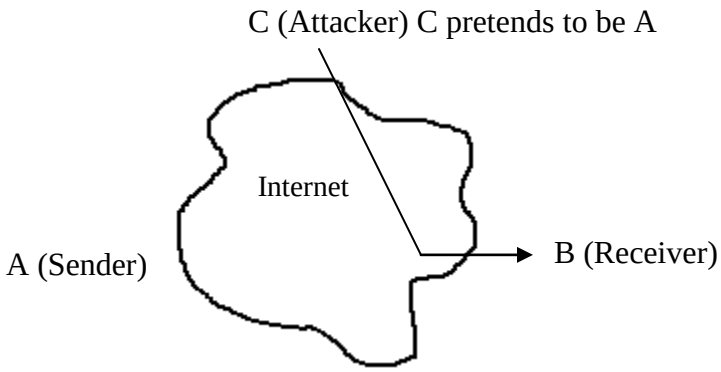


Fig 1.4 b: Masquerade

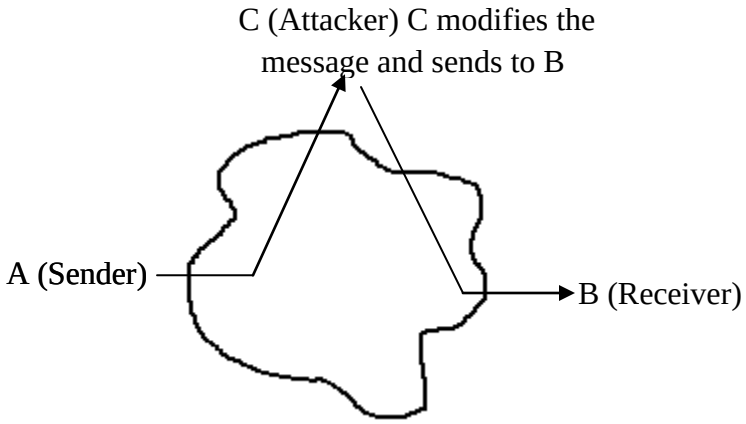
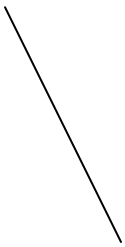


Fig 1.4 c: Modification of message

C (Attacker) disrupts service to A



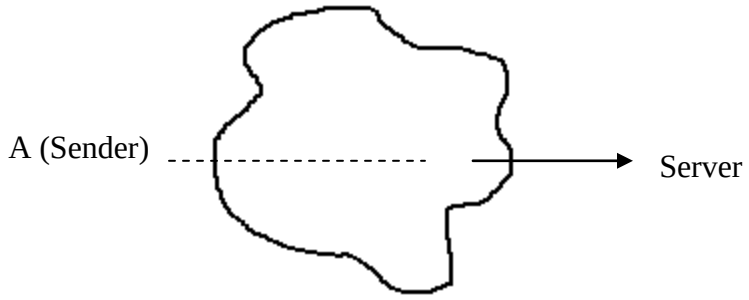


Fig 1.4 d: Denial of service

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

1.10 SUMMARY

Information security increased the ease of threats to information with use of sophisticated computing systems and several examples of security violations are described in detail in first three sections. In sections 1.4 through 1.8 complexities of security service, models, goals and trends in security are explained. Finally various passive and active attacks are discussed in the closing section 1.9.

1.11 KEYWORDS

Security-Threats, Attacks, Security Goals, Security Model, Trends in Security, Security Violations, Active Attacks, Passive Attacks, Virus, Worms.

1.12 QUESTIONS

1. Compare securing information in past and now.
2. How is computer useful in designing attacks?
3. Give the examples of security violations.
4. “Internet security is very challenging”-justify.
5. Describe models of security.
6. Explain primary goals of security.
7. Discuss the past and present trends in attacks.
8. Briefly explain various attacks.

1.13 REFERENCES

1. Atul Kahate, Cryptography and Network Security, Tata MCGrawHill
2. Behrouz A Forouzan, Cryptography and Network security, McGraw Hill
3. Charlie Kauffman, Radia Perlman, Mike Spciner, Network Security, Pearson Education
4. Wade Trappe, Lawrence Washington, Introduction to Cryptography with Coding Theory, Pearson International
5. William Stallings, Cryptography and Network Security, Pearson

UNIT -2: SECURITY SERVICES AND MECHANISMS

Structure

2.0 OBJECTIVES

A thorough study of this unit will make you proficient in

- ✓ Essential Security services to be provided by communication system.
- ✓ Methods/mechanisms that can ensure various services.
- ✓ Techniques to realize security goals.

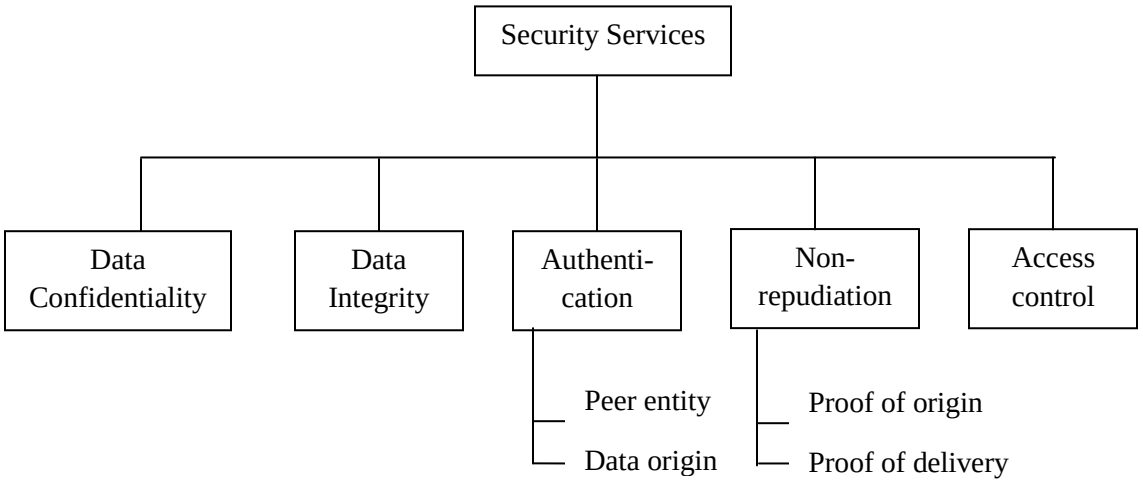
2.1 SECURITY SERVICES.

ITU-T (International telecommunication Union and Standardization Sector) develops standards called relating to Telecommunication and OSI Recommendation. Recommendation X.800 (Security Architecture for OSI) and IETF RFC 2828 (Internet Security Glossary) are used as references to systematically evaluate and define security requirements. Though coming from different standardization bodies, the two standards have many points in common. X.800 is used to define general security-related architectural elements needed when protection of communication between open systems is required. X.800 establishes guidelines and constraints to improve existing recommendations and/or to develop new recommendations in the context of OSI. Similarly, RFC 2828 provides abbreviations, explanations and recommendations for information system security terminology.

Both X.800 and RFC 2828 are designed to assist security managers in defining security requirements and possible approaches to meeting those requirements. They also help hardware and software manufacturers to develop security features for their products and services that follow certain standards. X.800 and RFC 2828 both mention several aspects of security systems, namely security threat and attack, security services and mechanisms and security management. This section gives a brief introduction to these standards. We urge readers to read the original standard documents for more information.

X.800 defines a security service as a service that is provided by a protocol layer of communicating open systems and that ensures adequate security of the systems that are components of data transfers. Perhaps a clearer definition is found in RFC 2828, which is as follows: a processing or communication service that is provided by a system to give a specific kind of protection to system resources; security services implement security policies and are implemented by security mechanisms. X.800 divides these services into five categories and fourteen specific services (Table 2.1). Here we look at each category in turn.

Figure here shows all specific services and the category they belong to.



Service and definition	Specific tasks
Data Confidentiality - Protection of data from unauthorized disclosure	1. Connection confidentiality 2. Connectionless confidentiality 3. Selective field confidentiality 4. Traffic flow confidentiality
Data Integrity - Assurance that data is as sent by authorized entity (contains no modifications, insertion, deletion, or replay)	1. Connection integrity with recovery 2. Connection integrity without recovery 3. Selective field connection integrity 4. Connectionless integrity 5. Selective field connectionless integrity
Authentication - Assurance that communicating entity is the one that it claims to be	1. Peer entity authentication 2. Data origin authentication
Non repudiation - provides protection against one of the entities from denying all or part of the communication	1. Non repudiation of origin 2. non repudiation of destination
Access Control - Prevention of unauthorized use of a resource	

Table 2.1: Category of services and specific tasks

Data Confidentiality

Confidentiality is the protection of transmitted data from passive attacks. With respect to the content of a data transmission, several levels of protection can be identified. The broadest service protects all user data transmitted between two users over a period of time. For example, when a TCP connection is set up between two systems, this broad protection prevents the release of any user data transmitted over the TCP connection. It can detect modifications (insertion, deletion, replay) and attempt recovery (task 1 in the table 2.1). Narrower forms of this service can also be

defined, including the protection of a single message or even specific fields within a message (task 3 in table 2.1). These refinements are less useful than the broad approach and may even be more complex and expensive to implement. The other aspect of confidentiality is the protection of traffic flow from analysis (task 4 in the table). This requires that an attacker not be able to observe the source and destination, frequency, length, or other characteristics of the traffic on a communications facility.

Data Integrity

As with confidentiality, integrity can apply to a stream of messages, a single message, or selected fields within a message. Again, the most useful and straightforward approach is total stream protection.

A connection-oriented integrity service, one that deals with a stream of messages, assures that messages are received as sent with no duplication, insertion, modification, reordering, or replays. The destruction of data is also covered under this service (task 2 in the table 2.1). Thus, the connection-oriented integrity service addresses both message stream modification and denial of service. On the other hand, a connectionless integrity service, one that deals with individual messages without regard to any larger context, generally provides protection against message modification only (task 4 of the table).

We can make a distinction between service with and without recovery. Because the integrity service relates to active attacks, we are concerned with detection rather than prevention. If a violation of integrity is detected, then the service may simply report this violation, and some other portion of software or human intervention is required to recover from the violation. Alternatively, there are mechanisms available to recover from the loss of integrity of data (task 1), as we will review subsequently. The incorporation of automated recovery mechanisms is, in general, the more attractive alternative.

Authentication

The authentication service is concerned with assuring that a communication is authentic. In the case of a single message, such as a warning or alarm signal, the function of the authentication service is to assure the recipient that the message is from the source that it claims to be from. In the case of request for interaction, such as the connection of a terminal to a host, two things are to be taken care of. First, at the time of connection initiation, the service assures that the two

participating entities are authentic, that is, that each is the entity that it claims to be. Second, the service must assure that the connection is not interfered with in such a way that a third party can masquerade as one of the two legitimate parties and perform unauthorized transmission or reception.

Two specific authentication services are defined in X.800:

1. Peer entity authentication: Provides for the corroboration of the identity of two entities participating in communication. Peer entity authentication is provided for use at the establishment of, or at times during the data transfer phase of, a connection. It attempts to provide confidence that an entity is not performing either a masquerade or an unauthorized replay of a previous connection.
2. Data origin authentication: Provides for the corroboration of the source of a message (sender). It does not provide protection against the duplication or modification of data units. This type of service supports applications like electronic mail, where there are no prior interactions between the communicating entities.

Access Control

In the context of network security, access control is the ability to limit and control the access to host systems and applications via communications links. To achieve this, each entity trying to gain access must first be identified, or authenticated, so that access rights can be tailored to the individual.

Non-repudiation

Non-repudiation prevents either sender or receiver from denying message transmission or receipt of message. Thus, 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.

Availability of Service

In addition to services listed in the table above, both X.800 and RFC 2828 define availability to be the property of a system or a system resource being accessible and usable upon demand by an authorized system entity, according to performance specifications for the system (i.e., a system is available if it provides services according to the system design whenever users request them). A

variety of attacks can result in the loss of or reduction in availability. Some of these attacks are amenable to automated countermeasures, such as authentication and encryption, whereas others require some sort of physical action to prevent or recover from loss of availability of elements of a distributed system.

X.800 treats availability as a property to be associated with various security services. However, it makes sense to call out specifically an availability service. An availability service is one that protects a system to ensure its availability to authorized users. This service addresses the security concerns raised by denial-of-service attacks. It depends on proper management and control of system resources and thus depends on access control service and other security services.

2.2 SECURITY MECHANISM

We discuss here the list of the security mechanisms defined in X.800. The mechanisms are divided into those that are implemented in a specific protocol layer, such as TCP or an application-layer protocol, and those that are not specific to any particular protocol layer or security service. These mechanisms are called 'specific security mechanisms' and 'pervasive security mechanism'.

Specific Security Mechanisms

These may be incorporated into the appropriate protocol layer in order to provide some of the OSI security services. Some techniques for realizing security are listed here.

1. **Encipherment**

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

2. **Digital Signature**

Data or cryptographic transformation of a data unit is appended to the data, so that the recipient of the data unit is convinced of the source and integrity of the data unit and this can also serve to protect the data against forgery (e.g., by the recipient).

3. **Access Control**

A variety of mechanisms are available that enforce access rights to resources.

4. **Data Integrity**

A variety of mechanisms may be used to assure the integrity of a data unit or stream of data units.

5. **Authentication Exchange**

This is a mechanism intended to ensure the identity of an entity by means of information exchange.

6. **Traffic Padding**

The insertion of bits into gaps in a data stream is called traffic padding. This helps to thwart traffic analysis attempts.

7. **Routing Control**

Routing control enables selection of particular physically secure routes for certain data transmission and allows routing changes, especially when a breach of security is suspected.

8. **Notarization**

This is the use of a trusted third party to assure certain properties of a data exchange.

Pervasive Security Mechanisms

These are the mechanisms that are not specific to any particular OSI security service or protocol layer.

1. **Trusted Functionality**

The process that which is perceived to be correct with respect to some criteria (e.g., as established by a security policy).

2. **Security Label**

This is the technique of marking of a bound to a resource (which may be a data unit) that names or designates the security attributes of that resource.

3. Event Detection

Detection of security-relevant events such as forgery, denial of sending or receiving of data, alteration of data etc. is another important essential mechanism.

4. Security Audit Trail

Data can be collected and potentially used to facilitate a security audit, which is an independent review and examination of system records and activities.

5. Security Recovery

This deals with requests from mechanisms, such as event handling and management functions, and takes recovery actions.

2.3 SERVICES AND MECHANISMS

Table 2.2, based on one in X.800, indicates the relationship between security services and security mechanisms.

Service	Encipherment	Digital Signature	Access Control	Data Integrity	Authentication Exchange	Traffic Padding	Routing Control	Notarization
Peer Entity Authentication	Y	Y			Y			
Data Origin Authentication	Y	Y						
Access Control			Y					
Confidentiality	Y						Y	
Traffic Flow Confidentiality	Y					Y	Y	
Data Integrity	Y	Y		Y				
Non repudiation		Y		Y				Y
Availability				Y	Y			

2.4 TECHNIQUES

Mechanisms discussed in the previous section are only theoretical recipes to implement security. The actual implementation of security goals needs some techniques. Two techniques are prevalent today: one (cryptography) is very general and the other one (steganography) is specific.

Cryptography

Some security mechanisms listed in the previous section can be implemented using cryptography. Cryptography, a word with Greek origin, means "secret writing". However, we use the term to refer to the science and art of transforming messages to make them secure and immune to attacks. Although in the past cryptography reffered only to the encrytion and decryption of messages using secert keys, today it is defined as involving three distinct mechanisms: symmetric-key encripherment, asymmetric-key encipherment, and hashing. We will briefly discuss these three mechanisms here.

1. Symmetric-key Encipherment

In symmetric encipherment, an entity, say Alice, can send a message to other entity, say Bob, over an insecure channel with the assumption that an adversary, say Eve, cannot understand the contents of the message by simply eavesdropping over the channel. Alice encrypts the message using an encryption algorithm. Bob decrypts the message using a decryption algorithm. Symmetric-key encipherment uses a single secret key for both encryption and decryption. Encryption/decryption can be thought of as electronic locking system. In symmetric-key enciphering, Alice puts the message in a box and locks the box using the shared secret key; Bob unlocks the box with the same key and takes out the messages.

2. Asymmetric Encipherment

In asymmetric encipherment, we have the same situation aas the symmetric-key encipherment, with a few exceptions. First, there are two keys instead of one; one public key and one private key. To send a secure message to Bob, Alice firsts encrypts the message using Bob's public key. To decrypts the message, Bob uses his own private key.

3. **Hashing**

In hashing, a fixed-length message digest is created out of a variable-length message. The digest is normally much smaller than the message. To be useful, both the message and the digest must be sent to Bob. Hashing is used to provide checkvalues, which were discussed earlier in relation to providing data integrity.

Steganography

This is the art of hidiing messages in another form. Message is not alltered as in encryption. A text can hide a message. For exmple "red umbrella needed" may mean the message "run". The first letter of each word in the text becomes the message. An image can also be used for hiding messages. Digital images are after all binary information. Suppose the image is grey image. The least significant bit of consecutive eight pixels may be alterd to be a specific bit pattern of a character. We will discuss this technique of steganograpphy in detail in the unit to come.

2.5 SUMMARY

A thorough description of five major categories of security services may be found in section 2.1.In the section next different mechanisms to provide the security services are elaborately discussed. Mechanisms that ensure services are listed in table 2.2 in section 2.3. In the closing section 2.4 two prevalent techniques cryptography and steganography are explained briefly with interesting illustrations.

2.6 KEYWORDS

Steganography, Symmetric key encipherment, Asymmetric key encipherment, Data integrity, Digital signature, Authentication, Non repudiation, Data confidentiality, Access control, Notarization, Routing control, Digital signature, Hashing

2.7 QUESTIONS

1. What are five categories of security services?
2. Mention and briefly explain the function of specific services.
3. Briefly explain various security mechanisms.
4. Relate security services and mechanisms.

5. Discuss two types of encipherment.
6. Discuss briefly explain covering a message with image and text.

2.8 REFERENCES

1. Atul Kahate, Cryptography and Network Security, Tata MCGrawHill
2. Behrouz A Forouzan, Cryptography and Network security, McGraw Hill
3. Charlie Kauffman, Radia Perlman, Mike Spciner, Network Security, Pearson Education
4. Wade Trappe, Lawrence Washington, Introduction to Cryptography with Coding Theory, Pearson International
5. William Stallings, Cryptography and Network Security, Pearson

UNIT -3: CLASSICAL ENCRYPTION - PART I

Structure

3.0 OBJECTIVES

After going through this unit you will be able to

- ✓ Understand basic principle of symmetric cipher
- ✓ Encrypt and decrypt messages using simple substitution methods
- ✓ Understand the weakness of encryption methods
- ✓ Devise ways to strengthen the methods
- ✓ Devise cryptanalytic attacks on the methods

3.1 SYMMETRIC CIPHER MODEL

A symmetric encryption scheme has five ingredients (Figure 3.1). They are

1. **Plaintext:** This is the original intelligible message or data that is fed into the algorithm as input.
2. **Encryption algorithm:** The encryption algorithm performs various substitutions and transformations on the plaintext.
3. **Secret key:** The secret key is also input to the encryption algorithm. The key is a value independent of the plaintext and of the algorithm. The algorithm will produce a different

output depending on the specific key being used at the time. The exact substitutions and transformations performed by the algorithm depend on the key.

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

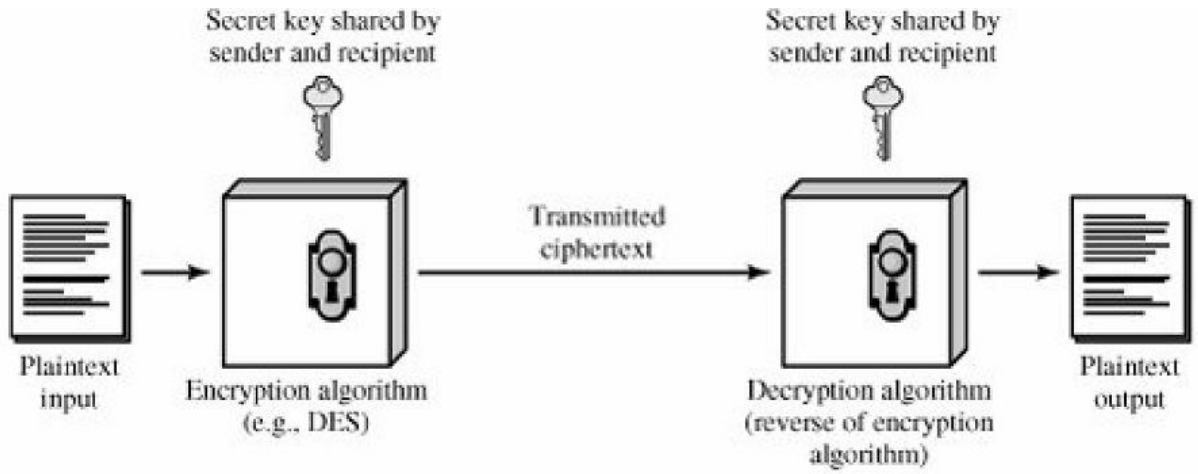


Figure 3.1: Simplified Model of Conventional Encryption

There are two requirements for secure use of conventional encryption:

1. We need a strong encryption algorithm. At a minimum, we would like the algorithm to be such that an opponent who knows the algorithm and has access to one or more cipher texts would be unable to decipher the cipher text or figure out the key. This requirement is usually stated in a stronger form: The opponent should be unable to decrypt cipher text or discover the key even if he or she is in possession of a number of cipher texts together with the plaintext that produced each cipher text.

2. Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure. If someone can discover the key and knows the algorithm, all communication using this key is readable.

We assume that it is impractical to decrypt a message on the basis of the cipher text plus knowledge of the encryption/decryption algorithm. In other words, we do not need to keep the algorithm secret; we need to keep only the key secret. This feature of symmetric encryption is what makes it feasible for widespread use. The fact that the algorithm need not be kept secret means that manufacturers can and have developed low-cost chip implementations of data encryption algorithms. These chips are widely available and incorporated into a number of products. With the use of symmetric encryption, the principal security problem is maintaining the secrecy of the key. For this reason key is sent to the receiver through a separate secure channel. Alternatively, a trusted third party can generate the key and send this to both source and destination.

Let us take a closer look at the essential elements of a symmetric encryption scheme, using Figure 3.2. A source produces a message in plaintext, X = [X1, X2, ... XM]. The M elements of X are letters in some finite alphabet. Traditionally, the alphabet usually consisted of the 26 capital letters. Nowadays, the binary alphabet {0, 1} is typically used. For encryption, a key of the form K = [K1, K2, ... KJ] is generated. If the key is generated at the message source, then it must also be provided to the destination by means of some secure channel. Alternatively, a third party could generate the key and securely deliver it to both source and destination.

With the message X and the encryption key K as input, the encryption algorithm forms the cipher text $Y= [Y_1, Y_2, ..., Y_N]$.

We write this as Y= E(K, X).

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

The intended receiver, in possession of the key, is able to invert the transformation using decryption algorithm and the secret key.

We write this as X= D (K, Y)

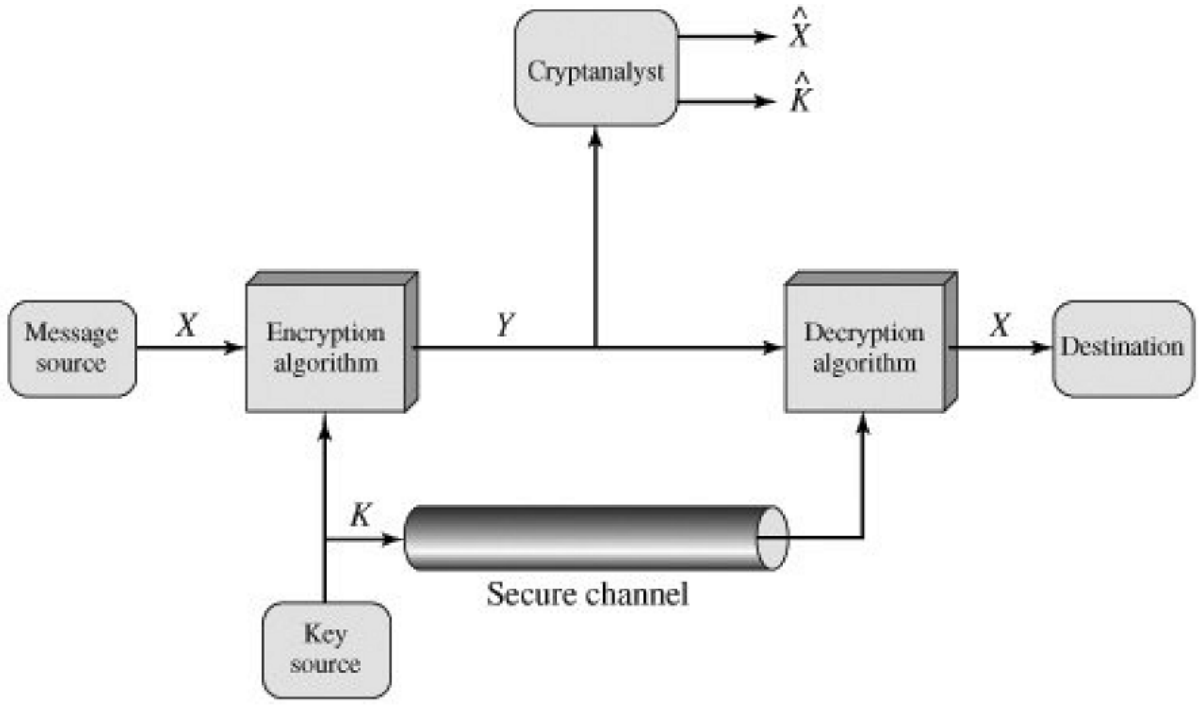


Figure 3.2: Model of Conventional Cryptosystem

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

3.2 CRYPTOSYSTEMS AND CRYPTANALYSIS

Cryptosystems

Cryptographic systems are characterized along three independent dimensions:

1. **The type of operations used for transforming plaintext to cipher text:** All encryption algorithms are based on two general principles: substitution, in which each element in the plaintext (bit, letter, group of bits or letters) is mapped into another element, and transposition, in which elements in the plaintext are rearranged. The fundamental

requirement is that no information be lost (that is, that all operations are reversible). Most systems, referred to as product systems, involve multiple stages of substitutions and transpositions.

2. **The number of keys used:** If both sender and receiver use the same key, the system is referred to as symmetric, single-key, secret-key, or conventional encryption. If the sender and receiver use different keys, the system is referred to as asymmetric, two-key, or public-key encryption.
3. **The way in which the plaintext is processed:** A block cipher processes the input one block of elements at a time, producing an output block for each input block. A stream cipher processes the input elements continuously, producing output one element at a time, as it goes along.

Cryptanalysis

Cryptanalytic attacks rely on the nature of the algorithm plus perhaps some knowledge of the general characteristics of the plaintext or even some sample plain text-cipher text pairs. This type of attack exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or to deduce the key being used.

Table 3.1 summarizes the various types of cryptanalytic attacks, based on the amount of information known to the cryptanalyst. The most difficult problem is presented when all that is available is the cipher text only. In some cases, not even the encryption algorithm is known, but in general we can assume that the opponent does know the algorithm used for encryption. One possible attack under these circumstances is the brute-force approach of trying all possible keys. If the key space is very large, this becomes impractical. Thus, the opponent must rely on an analysis of the cipher text itself, generally applying various statistical tests to it. To use this approach, the opponent must have some general idea of the type of plaintext that is transformed, such as the language of the text namely, English or French text, an EXE file, a Java source listing, an accounting file, and so on.

The cipher text only attack is the easiest to defend against because the opponent has the least amount of information to work with. In many cases, however, the analyst has more information. The analyst may be able to capture one or more plaintext messages as well as their encryptions. Or the analyst may know that certain plaintext patterns will appear in a message.

For example, a file that is encoded in the Postscript format always begins with the same pattern, or there may be a standardized header or banner to an electronic funds transfer message, and so on. All these are examples of known plaintext. With this knowledge, the analyst may be able to deduce the key on the basis of the way in which the known plaintext is transformed.

Closely related to the known-plaintext attack is what might be referred to as a probable-word attack. If the opponent is working with the encryption of some general prose message, he or she may have little knowledge of what is in the message. However, if the opponent is after some very specific information, then parts of the message may be known. For example, if an entire accounting file is being transmitted, the opponent may know the placement of certain key words in the header of the file. As another example, the source code for a program developed by Corporation X might include a copyright statement in some standardized position.

If the analyst is able somehow to get the source system to insert into the system a message chosen by the analyst, then a chosen-plaintext attack is possible. An example of this strategy is differential cryptanalysis. In general, if the analyst is able to choose the messages to encrypt, the analyst may deliberately pick patterns that can be expected to reveal the structure of the key.

Table 3.1 lists two other types of attack: Chosen cipher text and chosen text. These are less commonly employed as cryptanalytic techniques but are nevertheless possible avenues of attack. Only relatively weak algorithms fail to withstand a cipher text-only attack. Generally, an encryption algorithm is designed to withstand a known-plaintext attack.

Two more definitions are worthy of note. An encryption scheme is unconditionally secure if the cipher text generated by the scheme does not contain enough information to determine uniquely the corresponding plaintext, no matter how much cipher text is available. That is, no matter how much time an opponent has, it is impossible for him or her to decrypt the cipher text, simply because the required information is not there. With the exception of a scheme known as the one-time pad (described later in this chapter), there is no encryption algorithm that is unconditionally secure.

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

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

An encryption scheme is said to be computationally secure if either of the foregoing two criteria are met. The rub is that it is very difficult to estimate the amount of effort required to cryptanalyze cipher text successfully. All forms of cryptanalysis for symmetric encryption schemes are designed to exploit the fact that traces of structure or pattern in the plaintext may survive encryption and be discernible in the cipher text. This will become clear as we examine various symmetric encryption schemes. We will see that cryptanalysis for public-key schemes proceeds from a fundamentally different premise, namely, that the mathematical properties of the pair of keys may make it possible for one of the two keys to be deduced from the other.

Brute-force attack

The attacker tries every possible key on a piece of cipher text until an intelligible translation into plaintext is obtained. On average, half of all possible keys must be tried to achieve success. Table 3.2 shows how much time is involved for various key spaces. Results are shown for four binary key sizes. The 56-bit key size is currently in use with the DES (Data Encryption Standard) algorithm, and the 168-bit key size is used for triple DES. The minimum key size specified for AES (Advanced Encryption Standard) is 128 bits. Results are also shown for what are called substitution codes that use a 26-character key (discussed later), in which all possible permutations of the 26 characters serve as keys. For each key size, the results are shown assuming that it takes 1 micro second to perform a single decryption, which is a reasonable order of magnitude for today's machines. With the use of massively parallel organizations of microprocessors, it may be possible to achieve processing rates that are many orders of magnitude greater. The final column of Table 3.2 considers the results for a system that can process 1 million keys per microsecond. As you can see, at this performance level, DES can no longer be considered computationally secure.

Type of Attack	**Known to Cryptanalyst**
Cipher text Only	• Encryption algorithm • Cipher text
Known Plaintext	• Encryption algorithm • Cipher text

	• One or more plaintext–cipher text pairs formed with the secret key
Chosen Plaintext	• Encryption algorithm • Cipher text • Plain text message chosen by cryptanalyst, together with its corresponding cipher text generated with the secret key
Chosen Cipher text	• Encryption algorithm • Cipher text • Cipher text chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key
Chosen Text	• Encryption algorithm • Cipher ext • Plaintext message chosen by cryptanalyst, together with its corresponding Cipher text generated with the secret key • Cipher text chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key

Table 3.1: Types of attacks on encrypted messages

Key Size (bits)	**Number of Alternative Keys**	**Time Required at the rate of 1 Decryption/μs**	**Time Required at 10^6 Decryptions/μs**
32	$2^{32} = 4.3 \times 10^9$	2^{31}ms = 35.8 minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	2^{55}ms = 1142 years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	2^{127}ms = 5.4×10^{24} years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	2^{167}ms = 5.9×10^{36} years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 * 10^{26}$ms = 6.4×10^{12} years	6.4×10^6 years

Table 3.2: Average time for brute force attack

3.3 SUBSTITUTION TECHNIQUES

In this section we examine some classical encryption techniques, based on substitution. A substitution technique is one where letters of plain text are replaced by other letters / numbers / symbols. If plain text is a bit pattern then cipher is another bit pattern of same length. We discuss some substitution techniques that had been used in early times. We follow the convention of using small case letters for plain text and upper case letters for cipher text.

Caesar cipher

This is simple technique, used by Julius Caesar. The Caesar cipher involves replacing each letter of the alphabet with the letter standing *n* places further down the alphabet. For example,

Plain: meet me after the toga party
Cipher: PHHW PH DIWHU WKH WRJD SDUWB

Alphabet set is wrapped around. That is A follows Z. In this example *n*=3.

If the numbers 0 to 25 are assigned to alphabets then c = E(3, p) = (p+3) mod 26.
A shift of *k* characters is the general Caesar algorithm. Encryption and decryption formulas are given as:

c = E (*k*, p) = (p + *k*) mod 26, where *k*= 1 to 25
p = D (*k*, c) = (c-*k*) mod 26.

If it is known that Caesar cipher technique is used, cryptanalytic attack is easy. Attacker can try values 1 to 25 for *k* systematically and whichever *k* gives intelligible text is the key used.

Let us take the cipher text

KEY	PHHW PH DIWHU WKH WRJD SDUWB
1	oggv og chvgt vjg vqic rctva
2	nffu nf bgufs uif uphb qbsuz
3	meet me after the toga party

With three attempts intelligible text is exposed. No other *k* gives intelligible text. Three important characteristics enabled us to use brute force attacks

1. Encryption and decryption are known
2. total key size =25
3. Language of plain text is known and recognizable

In most networking situations, algorithm is known. What makes brute force attack difficult is number of possible keys. If the language of plain text is unknown then plaintext output may not be recognizable. Further input may be compressed or abbreviated. This makes decryption difficult. For example ZIP transformation of plaintext (uses more than 26 characters), a brute force attack will not expose the text.

3.3.2 Mono alphabetic cipher

Caesar cipher has just 25 keys. If we use any permutation of alphabets as a key we have 26! keys = 4×10^{26} keys. In this method, one letter is substituted for another, hence the name mono alphabetic cipher. The key space is greater than that of DES. But this encryption is not stronger than DES.

If the attacker knows the method then the attack (cryptanalytic attack) proceeds as follows: A frequency of characters appearing in the cipher may be obtained. Frequency of letters in a long plaintext may be obtained from a sample plain text. If the message is long, then we can get exact match of frequencies between ciphers and sample plain text. With short messages many cipher characters is likely to have more or less similar frequencies. If there is a single character (cipher) with highest frequency, find the corresponding character in sample plain text. Do similarly for distinct frequency characters. At this stage three to four characters may be revealed. For similar frequencies in cipher text, associate possible group of plaintext characters. The exact group can be discovered with frequency match of two to three characters.

Example

Consider the cipher text given here.

UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZ
VUEPHZHMDZSHZOWSFPAPPDTSVPQUZWYMXUZUHSX

EPYEPOPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ

As a first step the relative frequency of cipher characters in this short text is determined. The high frequency letters are P, Z, S, U, O, M, in order. The frequencies of these letters are 13.33, 11.67, 8.33, 8.33, 7.5, and 6.67. The high frequency letters in plain text (considered from a sample page of an English text) are e, t, a, o, i, n, in order. The frequencies of these letters are 12.702, 9.056, 8.167, 7.507, 6.996, and 6.749. So it is reasonable to assume that cipher characters P, Z correspond to plain text character e, t. Small frequencies cannot be matched, since the cipher text is usually short. The characters S, U, O, M will probably match one or the other in the group {a, o, i, n}. The low frequency characters in cipher are A, B, G, Y, I, J (with frequencies 1.67, 1.67, 1.67, 1.67, 0.83, 0.83). These probably match with one or other low frequency characters in sample text which are in the group {b, j, k, q, v, x, z} having frequencies 1.492, 0.153, 0.772, 0.095, 0.978, 0.15, 0.074. Note that these are the frequencies observed from the sample text. Thus we already have lot of information about cipher plain correspondence. There are number of ways to proceed from here. We can analyze double letter frequency in the sample text and find corresponding double cipher characters. In the cipher the most frequent double letter occurrence is ZW and in the sample plain text it is th. Thus we have many information revealed so far. Given here is the first line of cipher and the corresponding plain text characters. The cipher characters whose equivalent plain characters (given below) are discovered are underlined.

UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZ

t a e e te that e e a a

Only four letters are discovered and already we have quite a bit of information. Continued analysis of frequencies, trial and error matching between groups of cipher and plain text characters plus context of message will expose the plain text easily. Thus we learnt that this cipher can be broken easily with frequency matches. A counter measure is to use multiple substitutions for the same letter after certain number of substitutions, single letter, double letter frequency matches will fail here. Here too a letter can use a substitution letter for a particular homophone. Attacks are still possible since each element in cipher is out of a single element in

plain text. Multiple letter frequencies are more or less same between cipher and sample plain text substitution. Methods that are difficult to decode use two principal methods - one uses multiple (random) substitutions for each plain text character and the other creates a cipher for blocks of plain text characters.

3.3.3 Play fair cipher

Here a 5 ×5 matrix is created with characters of English alphabet. First a keyword is chosen. First few adjacent cells are filled with letters of this keyword (which has no letters repeating). Remaining cells are filled by alphabets (not entered) in order. For example if the keyword is "SAMPLE" the matrix is as follows, which is called digram. As the number of characters is 26 which is one greater than the number of cells, one cell will have two letters. A single cell will have I, J.

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

Write the plain text with no blank spaces, use a filler character such as x if a pair is same character.

Rules for substitution

1. If two adjacent characters are in same row (such as e, c), then the characters that are to their right are substitutions. Here B, D. If characters are q, u the substitution is R, O.
2. If two adjacent characters are in the same column, use substitutions that are one position below.

   Example: c , r : I, X (or J, X)

   q, w :W, A
3. If two adjacent characters are in different rows and columns, they are replaced by characters in the same row and in the column of the other

Example: w, k : Y, H

a, i :M, H

For decryption, the same matrix is used. The receiver knows the keyword. For characters in same row (or column) use ones that are just to the left (or above). Otherwise follow third rule of encryption.

This is much better than mono alphabetic cipher. Because replacement for a character is not constant since it is decided by its neighbor. For example, the cipher word for 'meet' is 'SCDO'. Thus C, D are both cipher characters for 'e'. With 26 characters, there are 676 digram. The identification of individual digrams and frequency match between cipher and plaintexts are more difficult. The relative frequencies of individual letters exhibit a much greater range than that of digrams, making frequency analysis much more difficult. For these reasons Playfair was considered to be safe and was used during world wars I and II by British and Allied forces.

Despite this level of confidence in its security, Playfair is easy to break, because it still leaves much of the structure of plaintext language intact. A few hundred letters of cipher text are generally sufficient.

3.3.4 Hill cipher

This is proposed by Lester Hill in 1929. In this method, m letters together are substituted by m cipher letters. The formula is c = k * p (mod 26), where k is m × m matrix (entries are mod 26). p, c are column vectors of size m. To get plain text back we use the formula $k^{-1}c \pmod{26} = p$.

We now discuss the computation of A^{-1} in mod 26.

Let $A = \begin{pmatrix} 5 & 8 \\ 17 & 3 \end{pmatrix}$, $\quad |A| = 15 - 136 = -121 = 9 \pmod{26}$

$$A^{-1} = \frac{1}{9}\begin{pmatrix} 3 & -8 \\ -17 & 5 \end{pmatrix} \bmod 26$$

1/9=3 (mod 26), since 3 and 9 are multiplicative inverses of mod 26. Thus,

$$A^{-1} = 3\begin{pmatrix} 3 & 18 \\ 9 & 5 \end{pmatrix}$$
$$= \begin{pmatrix} 9 & 54 \\ 27 & 15 \end{pmatrix} \bmod 26$$
$$= \begin{pmatrix} 9 & 2 \\ 1 & 15 \end{pmatrix}$$

Example: Let m=3.

$$k = \begin{pmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{pmatrix}$$

Suppose that plain text is "pay more money". Take first three characters and find its cipher.

$$pay = \begin{pmatrix} 15 \\ 0 \\ 24 \end{pmatrix}$$

$$c = \begin{pmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{pmatrix} \begin{pmatrix} 15 \\ 0 \\ 24 \end{pmatrix} = \begin{pmatrix} 375 \\ 819 \\ 486 \end{pmatrix} \bmod 26 = \begin{pmatrix} 11 \\ 13 \\ 18 \end{pmatrix} = LNS.$$

Decryption requires k^{-1}.

|k| = -939 = 23 (mod 26)

$$k^{-1} = \frac{1}{23}\begin{pmatrix} 300 & -313 & 267 \\ -357 & 313 & -252 \\ 6 & 0 & -51 \end{pmatrix} \bmod 26$$

$$= \frac{1}{23}\begin{pmatrix} 14 & 25 & 7 \\ 7 & 1 & 8 \\ 6 & 0 & 1 \end{pmatrix}$$

$$= \frac{17}{17 \times 23 (\bmod 26)}\begin{pmatrix} 14 & 25 & 7 \\ 7 & 1 & 8 \\ 6 & 0 & 1 \end{pmatrix} \bmod 26$$

$$= \begin{pmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{pmatrix}$$

Note that in the calculations above, 1/23 is multiplied and divided by 17. This is because 17 and 23 are multiplicative inverses of each other and hence the denominator is reduced to 1.

Plain text is given by

$$k^{-1}c = \begin{pmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{pmatrix}\begin{pmatrix} 11 \\ 13 \\ 18 \end{pmatrix} \bmod 26$$

$$= \begin{pmatrix} 15 \\ 0 \\ 24 \end{pmatrix}$$

= “pay “

As with play fair the strength of Hill cipher is that it hides the single letter frequencies. Larger the m value, more the information (frequency information) is hidden. The 3× 3 Hill cipher is stronger against cipher text only attacks. But when m plain cipher pairs are known (may be available from single plain message and corresponding cipher message) key is compromised.

Let $P_j = \begin{pmatrix} p_{ij} \\ . \\ . \\ . \\ p_{mj} \end{pmatrix}$ and corresponding $C_j = \begin{pmatrix} c_{ij} \\ . \\ . \\ . \\ c_{mj} \end{pmatrix}$ be j^{th} plain cipher pair.

That is $C_j = k P_j$, for $1 \leq j \leq m$.

Let $X= (p_{ij})$ be the $m \times m$ matrix of column vectors of plain texts.

Let $Y = (c_{ij})$ be the $m \times m$ matrix of column vectors of m cipher texts.

Now $Y= KX$ and $X^{-1}Y = K$. that is K can be computed as $X^{-1} Y$. If X is not invertible use more plain cipher pairs until X becomes invertible.

Example: Let m=2. Suppose we got plain text "friday" and its cipher "PQCFKU". Now we have three plain cipher pairs. These are " fr" ,"PQ"; "id", "CF" and "ay"," KU".

We know that $K\begin{pmatrix} 5 \\ 17 \end{pmatrix} = \begin{pmatrix} 15 \\ 16 \end{pmatrix}$ *and* $K\begin{pmatrix} 8 \\ 3 \end{pmatrix} = \begin{pmatrix} 2 \\ 5 \end{pmatrix}$

Also we have, $K\begin{pmatrix} 0 \\ 24 \end{pmatrix} = \begin{pmatrix} 10 \\ 20 \end{pmatrix}$.

Using the first two pairs we have

$$\begin{pmatrix} 15 & 2 \\ 16 & 5 \end{pmatrix} = k\begin{pmatrix} 5 & 8 \\ 17 & 3 \end{pmatrix} \bmod 26$$

$$k = \begin{pmatrix} 15 & 2 \\ 16 & 5 \end{pmatrix}\begin{pmatrix} 5 & 8 \\ 17 & 3 \end{pmatrix}^{-1} \bmod 26$$

$$= \begin{pmatrix} 7 & 8 \\ 19 & 3 \end{pmatrix}.$$

Verification of key

$$K\begin{pmatrix} a \\ y \end{pmatrix} = \begin{pmatrix} 7 & 8 \\ 19 & 3 \end{pmatrix}\begin{pmatrix} 0 \\ 24 \end{pmatrix} \bmod 26 = \begin{pmatrix} 10 \\ 20 \end{pmatrix} = KU.$$

3.3.5 Poly alphabetic ciphers:

In this method, we make use of different mono alphabetic substitution as one proceeds through encryption.

Features of the method:

1. A set of related mono alphabetic substitution rules are used.
2. A key determines which rule is selected for a given plain text character position.

Vigenere cipher:

Here 26 Caesar ciphers are used (each is mono alphabetic). These are referred by characters a to z and numbers 0 to 25. Each cipher denoted by a character which is cipher character for letter ‘a’.

Vigenere table:

		Plain text characters								
Key		a	b	c	d	e	….	x	y	z
	a	A	B	C	D	E	….	X	Y	Z
	b	B	C	D	E	….		Y	Z	A
	c	C	D	E	….	X		Z	A	B
	d	D	E	….	X	Y		A	B	C
	e	E	….	X	Y	Z		B	C	D
	: :									
	y	Y	Z	A	B	C		V	W	X
	z	Z	A	B	C	D		W	X	Y

Method

Choose a key word say “deceptive”. To get cipher letter copy keywords as many times and write it on top of the plain text.

```
Keyword   :  deceptivedeceptivedeceptive
plaintext:  wearediscoveredsaveyourself
ciphertext: ZICVTW......VTW..........
```

Decryption: The letter in the key is the row. Find the letter in the column that has the cipher letter.

Strength: Cipher letters of same plain text letter is usually different. Letter frequency information is hidden.

However not all knowledge of plain text is lost. For example the same letters ‘red’ is coded as ‘VTW’ twice. Attack will proceed to find same pattern in cipher text. The probable information is length of keyword. Suppose the attacker knows its Mono alphabetic or Vigenere cipher. If Mono alphabetic cipher is used statistical properties of characters in cipher will break the code. If the opponent discovers it is not a Mono alphabetic, he knows its Vigenere. If identical patterns in cipher text are discovered, then length of the keyword is distance between the patterns or a factor of this. For example distance between VTW’s is 9 and hence the keyword length is 9 or 3 if keyword length is N (guess), then at N+1, 2N+1, … same row is used for cipher characters. All cipher characters at these positions can be picked and frequency match with plaintext characters can be done. Do same for cipher characters at 2, N+2, 2N+2, . . . Continuing this he can get several characters in keyword.

Counter measure: The periodicity of string in cipher can be broken by using different rows for making up cipher. One method is having long key word. Another is use plain text next to keyword and makes it as long as plain text

Example: Key could be

d e c e p t i v e w e a r e d i s c o v e r e d s a

w e a r e d i s c o v e r d s a v e y o u r s e l f - plain text

Even this is not very safe. Key and plaintext share same statistical properties with this method. For example, e enciphered using e with probability $(0.127)^2$ =0.016. t encoded by t with probability $(0.0956)^2$ etc. Thus using frequency match some letters in the keyword can be retrieved. The ultimate defense is to use a keyword that is as long as the text but has no statistical relationship to it.

3.3.5 Vernam Cipher

This cipher method is proposed by Gilbert Vernam (1918) that is to be used for binary string encoding. This is nothing but simple bit wise XOR operation given by,

$$c_i = p_i \oplus k_i$$

Decryption is the XOR operation of *c* and *k*

Example: P = 01101110, K = 11011001, C = 10110111

Decryption: P= C ⊕ K

C = 10110111

K = 11011001

P = 01101110

The strength lies in the length of the key. However long it is, it may have to repeat for long messages. This is also susceptible to access of some plain – cipher pair attacks => bit values of key at these positions are known. However it is difficult to get all the bits of key.

3.4 SUMMARY

In section 3.1 symmetric cipher model and cryptanalytic attacks are explained in detail. Encryption is an age old practice. Traditional Cryptosystems based on substitution are discussed in section 3.3. To make concepts clear, examples, attacks and counter measures are also explained.

3.5 KEYWORDS

Symmetric cipher, Brute force attacks, cryptanalytic attack, classical encryption techniques- Caesar, Mono alphabetic Polyalphabetic, Vigenere, Playfair, Hill, Vernam

3.6 QUESTIONS

1. Differentiate symmetric and asymmetric ciphers.
2. What are two basic functions used in encryption? Give examples. Discuss plus and minus points of these methods.
3. Discuss the general approaches of attacks against a cipher.
4. Distinguish unconditionally secure and computationally secure ciphers.
5. Discuss classical substitution ciphers with examples, possible attacks and counter measures.

3.7 REFERENCES

1. Atul Kahate, Cryptography and Network Security, Tata MCGrawHill
2. Behrouz A Forouzan, Cryptography and Network security, McGraw Hill
3. Charlie Kauffman, Radia Perlman, Mike Spciner, Network Security, Pearson Education
4. Wade Trappe, Lawrence Washington, Introduction to Cryptography with Coding Theory, Pearson International
5. William Stallings, Cryptography and Network Security, Pearson

UNIT -4: CLASSICAL ENCRYPTION - PART II

Structure

4.0 OBJECTIVES

When you go through the material discussed in this unit you will be able to

- ✓ Use transposition for encryption
- ✓ Understand the operation of rotor machines which is multiple encryption
- ✓ Appreciate the strength and simplicity of steganography for hiding messages

4.1 TRANSPOSITION TECHNIQUES

The basic principle of these techniques is to permute letters in the message.

4.1.1 Rail fence

In this method, letters are written on alternate rows and columns. Suppose that message is "meet me at the toga party". Write this as follows

m e m a t e o a a t

e t e t h t g p r y

The letters in the text are written alternatively in two lines.

The cipher is composed as writing first line and then the second line.

M E M A T E O A A T E T E T H T G P R Y

Decryption: Split cipher into two halves, write characters alternately from each half.

M E M A T E O A A T (one half)
E T E T H T G P R Y

Recovered text is “meetmeatthetogaparty”

While decrypting, if there are odd number of characters in the cipher text, write greater half as first line and smaller half as second line.

Attack: If the adversary knows the encryption to be rail fence, attack is a too simple and recovery of plain text is too simple. The attacker has to split the cipher into two halves and write characters one at a time from each half.

4.1.2 Use of permutation

In this technique, the message is written in adjacent rows and columns are permuted. Suppose that message is “meet me at the toga party at nine pm”.

The characters in the message are written in successive rows, each row having fixed number of columns. Then columns are permuted and cipher is generated.

Example: Suppose that given message is written in 6 columns

M E E T M E

A T T H E T

O G A P A R

T Y A T N I

N E P M X Y

Column permutation: 4 1 3 2 5 6

The column permutation and the number of rows is the key for decryption. Here x, y are filler characters to complete the matrix.

Cipher text is ETGYETHPTMETAAPMAOTNMEANXETRIY

Decryption: Key is the number of rows (5) and the column permutation (4 1 3 2 5 6). First 5 characters in cipher is written as column 2 (number 1 is in second position of the permutation of columns), next 5 characters as column 4 (number 2 is in fourth position of the permutation of columns) and so on.

Attack: A pure transposition is easy to break. Knowing the length of the message, various matrix sizes and permutation may be tried.

In the example above, the length of the cipher is 30 characters. So attacker will try various matrix sizes such as 2 x 15, 3 x 10, and 5 x 6. Also various permutations of the columns should be tried.

4.1.3 Making multiple transpositions

Encrypt the cipher text again with the same permutation of columns

4	1	3	2	5	6
e	t	g	y	e	t
h	p	t	m	e	t
a	a	p	m	a	o
t	n	m	e	a	n
x	e	t	r	i	y

Cipher text: TPAWE YMMER GTPMTEHATX EEAAI TTONY

To realize the strength of double permutation we give here the character positions in cipher text.

Given positions of plain text characters

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

With one permutation of columns namely, (4 1 3 2 5 6)

The cipher text characters are from positions (in plaintext)

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

There is some regularity among groups of five elements.

Now for one more transposition of the cipher with the same key, we write the cipher text into matrix and perform the permutation again.

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

Cipher text letters are from positions

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

Observe that there is no structure here and hence the attack is complicated.

4.2 ROTOR MACHINES

We saw that multiple stages of transpositions are difficult to crack. Some is true of substitution. Rotor machines are used for this purpose. This was used during second world war. This is now used in DES, a standard in modern encryption.

The machine has 3 independently rotating cylinders. There could be any number of cylinders. More the number of cylinders greater is the security. The number of substitutions depends on the number of cylinders. Each cylinder has 26 input pins and 26 output pins, with internal wiring, connecting pins. Each cylinder is mono alphabetic substitution. When an input letter is depressed there is electrical current to an output letter. After each input, the cylinder 1 rotates one position and so identical connections are satisfied. This is just poly alphabetic substitution with a period of 26. But with more cylinders say 31 the number of unique substitutions is 26*26*26 = 17576, before it repeats itself. Thus the method is safe against cryptanalytic attacks such as frequency detection.

First cylinder rotates once after each input selection. Cylinder 2 rotates once after 26 rotations of first. Cylinder 3 rotates once after 26 rotations of cylinder 2.

Example: The example given here make concept of rotation of cylinders clear. When ‘a’ is plain text character this key is depressed and the cipher character B is output. With the initial setting shown in figure 4.1, the character ‘a’ points to number 24 in cylinder 1. The internal wiring in this cylinder points to number 24 in the second cylinder and the internal wiring here points to 18 in cylinder 3 and the output cipher character is B (against number 18 in cylinder 3). Similarly it is easy to follow that the cipher characters corresponding to plain text characters ‘b’ and ‘c’ are I and E respectively given the initial setting in the figure. After one output the cylinder 1 rotates one position down. That is ‘a’ points to 23 (also numbers on the right move down one position) and hence the cipher output for the same ‘a’ after one stroke is Y. Thus the word “bull” will be coded as IDSQ. Note the plain text letter ‘l’ is coded as S and Q in the cipher.

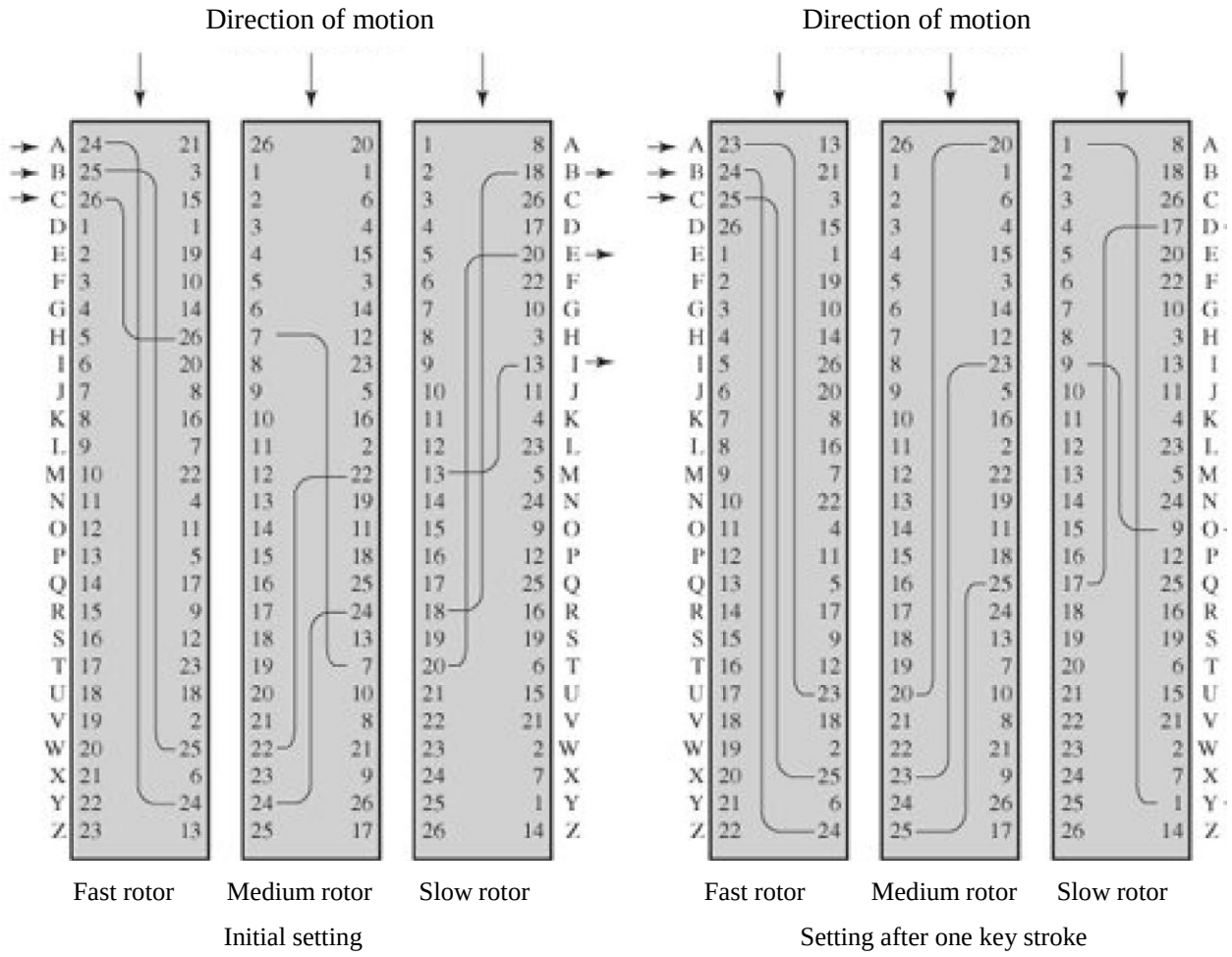


Fig 4.1: Rotor machine with 3 cylinders

4.3 STEGANOGRAPHY

Cryptography is a technique based on substitution and transposition for implementing security mechanisms. Another technique that was used for secret communication in the past is being revived at the present time. This is nothing but Steganography. The word Steganography, with origin in Greek, means "covered writing". In contrast with cryptography, which means "secret writing", is the technique of concealing the contents of a message by enciphering. Steganography means concealing the message itself by covering it with something else.

Historical use

History is full of facts and myths about the use of steganography. In China, war messages, were written on thin pieces of silk and rolled into a small ball and swallowed by the messenger. In Rome and Greece, messages were carved on pieces of wood, that were later dipped into wax to cover the writing. Invisible inks were also used to write a secret message between the lines of the covering message or on the back of the paper and the secret message was exposed when the paper was heated or treated with another substance.

In recent times other methods have been devised. Some letters in an innocuous message might be overwritten in a pencil lead that is visible only when exposed to light at an angle. Null ciphers were used to hide a secret message inside an innocuous simple message. For example, the first or second letter of each word in the covering message might compose a secret message. Microdots were also used for this purpose. Secret messages were photographed and reduced to a size of a dot and inserted into simple cover messages in place of regular periods at the end of sentences.

Modern use

Today, any form of data, such as text, image, audio, or video, can be digitized, anad it is possible to insert secret binary information into the data during digitization process. Such hidden information is not necessarily used for secrecy; it can also be used to protect copyright, prevent tampering, or add extra information.

Text Cover

The cover of secert data can be text. There are several ways to insert binary data into an innocuous text. For example, we can use single space between words to represent the binary data 0 and double space to represent binary digit 1. The following short messages hides the 8-bit binary representation of the letter A in ASCII code (01000001).

```
This book  is mostly about cryptography, not  steganography
   0      1  0        0         0                  0      1
```

In the above message there are two spaces between the “book” and “is” and between the “not” and “steganography”. Of course, sophisticated software can insert spaces that differ only slightly to hide from immediate recognition.

Another, more efficient method, is to use a dictionary of words organised according to their grammatical usages. We can have a dictionary containing 2 articles, 8 verbs, 32 nouns, and 4 prepositions. Suppose that we agree to use cover text that always use sentences with the pattern article-noun-article-noun. The secret binary data can be divided into 16-bit chunks. The first bit of binary data can be represented by an article. The next five bits can be represented by a noun, the next four bits can be represented by a verb, the next bit by the second article, and the last five bits by another noun. For example, the secert data "Hi", which is 01001000 01001001 in ASCII, could be a sentence like the following:

A friend called a doctor
0 10010 0001 0 01001

This is a very trivial example. The actual approach uses more sophisticcated design and a variety of patterns.

Image Cover

Secret data can also be covered under a color image. Digitized images are made of pixels, in which normally each pixels uses 24 bits. Each byte represents one of the primary colors. We can thererfore have 2^8 different shades of each color. In a method called LSB (least significant bit), the least significant bit of each byte is set to zero. This may make the image a little bit lighter in some areas, but this is not normally noticed. Now we can hide a binary data in the image by keeping or changing the least significant bit. If our binary digit is 0, we keep the bit; if it is 1, we change the bit to 1. In this way, we can hide a character in three pixels. For example, the following three pixels can represent the letter H.

0101001**0**	1011110**1**	0101010**0**
0101111**0**	1011110**1**	0110010**0**
0111111**0**	0100101**0**	00010100

Of course, more sophisticated approaches are used these days.

Other Covers

Other covers are also possible. The secert message, for example, can be covereed under audio and video. Both audio and video are compressed today; the secert data can be embedded during or before the compression.

4.4 SUMMARY

This unit introduced to the readers the concept of transposition techniques. These are different from techniques discussed earlier. Two methods of transposition techniques namely Rail fence and using permutation are discussed in section 4.1. Section 4.2 is about rotor machines a device used traditionally for encryption. Steganography is not an encryption technique. This is way of hiding data in some other medium such as texts and images. Examples of Steganography are given in 4.3.

4.5 KEYWORDS

Transposition techniques, Rail fence, Permutation method, Multiple permutations, Rotor machines, Steganography, Text cover for messages, Image cover for messages

4.6 QUESTIONS

1. Differentiate transposition and substitution methods.
2. Discuss rail fence transposition and comment on its safety.
3. Describe through examples permutation technique. Suggest an attack for this cipher.
4. Explain multiple permutation method and justify this better than single permutation.
5. Define Steganography and what the various forms are.
6. Explain text cover.
7. Describe the role of images for covering messages.
8. Explain other form of Steganography.

4.7 REFERENCES

1. Atul Kahate, Cryptography and Network Security, Tata MCGrawHill
2. Behrouz A Forouzan, Cryptography and Network security, McGraw Hill
3. Charlie Kauffman, Radia Perlman, Mike Spciner, Network Security, Pearson Education

4. Wade Trappe, Lawrence Washington, Introduction to Cryptography with Coding Theory, Pearson International
5. William Stallings, Cryptography and Network Security, Pearson

MODULE 2

BLOCK AND STREAM CIPHERS

UNIT -5: BLOCK CIPHER PRINCIPLES

Structure

5.0 OBJECTIVES

After understanding the concepts discussed in this unit you will understand

- ✓ Basic difference in two types of ciphers: stream and block
- ✓ Reversible and irreversible block ciphers
- ✓ Feistel cipher structure
- ✓ Ideal block cipher
- ✓ Manageable block cipher
- ✓ Encryption and decryption with Feistel structure
- ✓ Confusion
- ✓ Diffusion

5.1 MOTIVATION FOR FEISTEL STRUCTURE

The objective of this section is to introduce the fundamental principles of modern symmetric ciphers. For this purpose, we focus on the most widely used symmetric cipher: the Data Encryption Standard (DES). Although, numerous symmetric ciphers have been developed since the introduction of DES, and it is destined to be replaced by the Advanced Encryption Standard (AES), DES remains the most important algorithm. Furthermore, a detailed study of DES provides an understanding of the principles used in other symmetric ciphers.

The section begins with a discussion on general principles of symmetric block ciphers. First the primary differences between stream ciphers and block ciphers are discussed.

Stream versus Block ciphers

A **stream cipher** is one that encrypts a digital data stream one bit or one byte at a time. Examples of classical stream ciphers are the auto keyed Vigenère cipher and the Vernam cipher. In the ideal case, a one-time pad version of the Vernam cipher would be used, in which the key stream is as long as the plaintext bit stream. If the cryptographic key stream is random, then this cipher is unbreakable by any means other than acquiring the key stream. However, the key stream must be provided to both users in advance via some independent and secure channel. This introduces insurmountable logistical problems if the intended data traffic is very large.

Accordingly, for practical reasons, the bit-stream generator must be implemented as an algorithmic procedure, so that the cryptographic bit stream can be produced by both users. In this approach, the bit-stream generator is a key-controlled algorithm and must produce a bit stream that is cryptographically strong. Now, the two users need only share the generating key, and each can produce the key stream.

A **block cipher** is one in which a block of plaintext is treated as a whole and used to produce a cipher text block of equal length. Typically, a block size of 64 or 128 bits is used. As with a stream cipher, the two users share a symmetric encryption key. Using some of the modes of operation explained later in the material, a block cipher can be used to achieve the same effect as a stream cipher.

Far more effort has gone into analyzing block ciphers. In general, they seem applicable to a broader range of applications than stream ciphers. The vast majority of network-based symmetric cryptographic applications make use of block ciphers. Accordingly, the concern in this chapter, and in our discussions throughout the book of symmetric encryption, will primarily focus on block ciphers.

The section next will describe an important block cipher model proposed by Feistel in 1973. We now bring out the motivation for Feistel cipher structure.

A block cipher operates on a plaintext block of *n* bits to produce a cipher text block of *n* bits. There are possible different plaintext blocks and, for the encryption to be reversible (i.e., for decryption to be possible), each must produce a unique cipher text block. Such transformation is called reversible, or nonsingular .The following examples illustrate nonsingular and singular transformation for n=2. Encoding given in table 5.1 is a reversible mapping and that in table 5.2 is an irreversible mapping.

Reversible Mapping

Plaintext	**Cipher text**
00	11
01	10
10	01
11	00

Table 5.1

Irreversible Mapping

Plaintext	**Cipher text**
00	11
01	10
10	01
11	01

Table 5.2

In the latter case, a cipher text of 01 could have been produced by one of two plaintext blocks. So if we limit ourselves to reversible mappings, the number of different transformations is n!

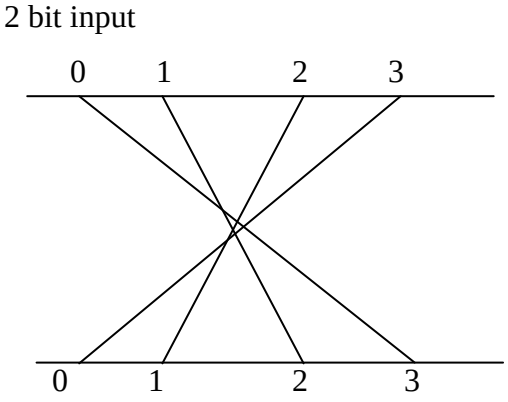


Fig. 5.1:2-bit substitution cipher

Figure 5.1 illustrates the logic of a general substitution cipher for n=2, corresponding to the mapping shown in table 5.1.

A 4-bit input produces one of 16 possible input states, which is mapped by the substitution cipher into a unique bit pattern of 16 possible output states, each of which is represented by 4 cipher text bits. The encryption and decryption mappings can be defined by two tables as shown in Tables 5.3 and 5.4. This is the most general form of block cipher and can be used to define any reversible mapping between plaintext and cipher text. Feistel refers to this as the *ideal block cipher*, because it allows for the maximum number of possible encryption mappings from the plaintext block.

Plain text	Cipher text
0000	0100
0001	1100
0010	0101
0011	1111
0100	0111
0101	0000
0110	1010
0111	0001
1000	1110
1001	1000
1010	0010
1011	1001
1100	0011
1101	1101
1110	1011
1111	0110

Table 5.3: Encryption

Cipher text	Plain text
0000	0101
0001	0111
0010	1010
0011	1100
0100	0000
0101	0010
0110	1111
0111	0100
1000	1001
1001	1011
1010	0110
1011	1110
1100	0001
1101	1101
1110	1000
1111	0011

Table 5.4: Decryption

But there is a practical problem with the ideal block cipher. If a small block size, such as n=4, is used, then the system is equivalent to a classical substitution cipher. Such systems, as we have seen, are vulnerable to a statistical analysis of the plaintext. This weakness is not inherent in the use of a substitution cipher but rather results from the use of a small block size. If sufficiently

large and an arbitrary reversible substitution between plaintext and cipher text is allowed, then the statistical characteristics of the source plaintext are masked to such an extent that this type of cryptanalysis is infeasible.

An arbitrary reversible substitution cipher (the ideal block cipher) for a large block size is not practical, however, from an implementation and performance point of view. For such a transformation, the mapping itself constitutes the key. Consider again Table 5.3, which defines one particular reversible mapping from plaintext to cipher text for n =4. The mapping can be defined by the entries in the second column, which show the value of the cipher text for each plaintext block. This, in essence, is the key that determines the specific mapping from among all possible mappings. In this case, using this straightforward method of defining the key, the required key length is (4 bits) × (16 rows) = 64 bits. In general, for a n-bit ideal block cipher, the length of the key defined in this fashion is $n \times 2^n$ bits. For a 64-bit block, which is a desirable length to thwart statistical attacks, the required key length is $64 \times 2^{64} = 2^{70} = 10^{21}$ bits .

In considering these difficulties, Feistel points out that what is needed is an approximation to the ideal block cipher system for large *n*, built up out of components that are easily realizable. A tractable general block cipher is to go for a manageable subset of all possible 2^n! n-bit block cipher such as the mapping defined by a set of linear equations

$y_1 = (k_{11}x_1 + k_{12}x_2 + k_{13}x_3 + k_{14}x_4) \bmod 2$

$y_2 = (k_{21}x_1 + k_{22}x_2 + k_{23}x_3 + k_{24}x_4) \bmod 2$

$y_3 = (k_{31}x_1 + k_{32}x_2 + k_{33}x_3 + k_{34}x_4) \bmod 2$

$y_4 = (k_{41}x_1 + k_{42}x_2 + k_{43}x_3 + k_{44}x_4) \bmod 2$

Here block size =4, x_i's are binary digits of a block and y_i's are computed output bit. Note that key size is 4^2 (all k_{ij} for i=j= 1 to 4). This is much smaller compared to 2^4! Note that the equations above is essentially Hill cipher discussed in module 1, applied to binary data. Such simple linear equations are vulnerable to attacks. Recall the attack discussed in the section of Hill cipher. Feistel proposed a product cipher which made the cipher scheme strong against attacks.

5.2 FEISTEL CIPHERS

Feistel's method (developed in 1973) is a practical application of Claude Shannon's proposal in 1945 to alternate confusion and diffusion functions in the product cipher. It is worth

commenting that modern symmetric cipher is based on Feistel's structure which in turn is developed on Claude Shannon's suggestions. Thus today's wide used symmetric encryption is dated back to more than half a century.

Feistel proposed that we can approximate the ideal block cipher by utilizing the concept of a product cipher, which is the execution of two or more simple ciphers in sequence in such a way that the final result or product is cryptographically stronger than any of the component ciphers. The essence of the approach is to develop a block cipher with a key length of *k* bits and a block length of *n* bits, allowing a total of possible 2^k transformations, rather than the 2^n! transformations available with the ideal block cipher.

In particular, Feistel proposed the use of a cipher that alternates substitutions and permutations, where these terms are defined as follows:

• **Substitution:** Each plaintext element or group of elements is uniquely replaced by a corresponding cipher text element or group of elements.

• **Permutation:** A sequence of plaintext elements is replaced by a permutation of that sequence. That is, no elements are added or deleted or replaced in the sequence, rather the order in which the elements appear in the sequence is changed.

We now discuss the meaning of the terms confusion, diffusion. The terms *diffusion* and *confusion* were introduced by Claude Shannon to capture the two basic building blocks for any cryptographic system. Shannon's concern was to thwart cryptanalysis based on statistical analysis. The reasoning is as follows. Assume the attacker has some knowledge of the statistical characteristics of the plaintext. For example, in a human-readable message in some language, the frequency distribution of the various letters may be known. Or there may be words or phrases likely to appear in the message (probable words). If these statistics are in any way reflected in the cipher text, the cryptanalyst may be able to deduce the encryption key, part of the key, or at least a set of keys likely to contain the exact key. In what Shannon refers to as a strongly ideal cipher, all statistics of the cipher text are independent of the particular key used. The arbitrary substitution cipher that we discussed previously (Table 5.3) is such a cipher, but as we have seen, it is impractical.

Other than recourse to ideal systems, Shannon suggests two methods for frustrating statistical cryptanalysis: ***diffusion*** and ***confusion***. In **diffusion**, the statistical structure of the plaintext is dissipated into long-range statistics of the cipher text. This is achieved by having each plaintext digit affect the value of many cipher text digits; generally, this is equivalent to having each cipher text digit be affected by many plaintext digits. An example of diffusion is to encrypt a message of characters with an averaging operation: adding successive letters to get a cipher text letter. One can show with this type of encryption, the statistical structure of the plaintext has been dissipated. Thus, the letter frequencies in the cipher text will be more nearly equal than in the plaintext; the digram frequencies will also be more nearly equal, and so on. In a binary block cipher, diffusion can be achieved by repeatedly performing some permutation on the data followed by applying a function to that permutation; the effect is that bits from different positions in the original plaintext contribute to a single bit of cipher text.

Every block cipher involves a transformation of a block of plaintext into a block of cipher text, where the transformation depends on the key. The mechanism of diffusion seeks to make the statistical relationship between the plaintext and cipher text as complex as possible in order to thwart attempts to deduce the key. On the other hand, **confusion** seeks to make the relationship between the statistics of the cipher text and the value of the encryption key as complex as possible, again to thwart attempts to discover the key. Thus, even if the attacker can get some handle on the statistics of the cipher text, the way in which the key was used to produce that cipher text is so complex as to make it difficult to deduce the key. This is achieved by the use of a complex substitution algorithm. In contrast, a simple linear substitution function would add little confusion. Diffusion and confusion in capturing the essence of the desired attributes of a block cipher has become the cornerstone of modern block cipher design.

5.3 FEISTEL ENCRYPTION AND DECRYPTION

We turn our attention to Feistel's structure. Figure 5.2 and 5.3 show the structure proposed by Feistel.

Encryption:

The inputs to the encryption algorithm are a plaintext block of length 2*n* bits and a key K. The plaintext block is divided into two halves, L_0 and R_0. The two halves of the data pass through rounds of processing and then combine to produce the cipher text block. Each round *i* has as

inputs L_{i-1} and R_{i-1} derived from the previous round, as well as a subkey K_i derived from the overall K. In general, the subkeys are different from K and from each other. In Figure 5.2, 16 rounds are used, although any number of rounds could be implemented.

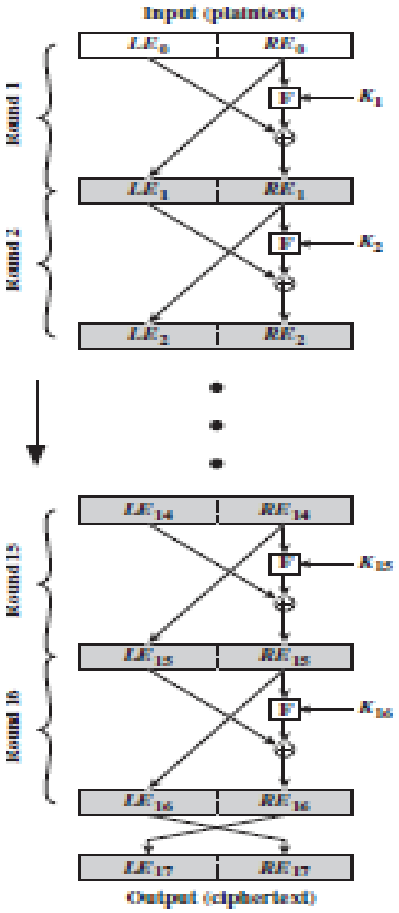


Figure 5.2: Feistel encryption

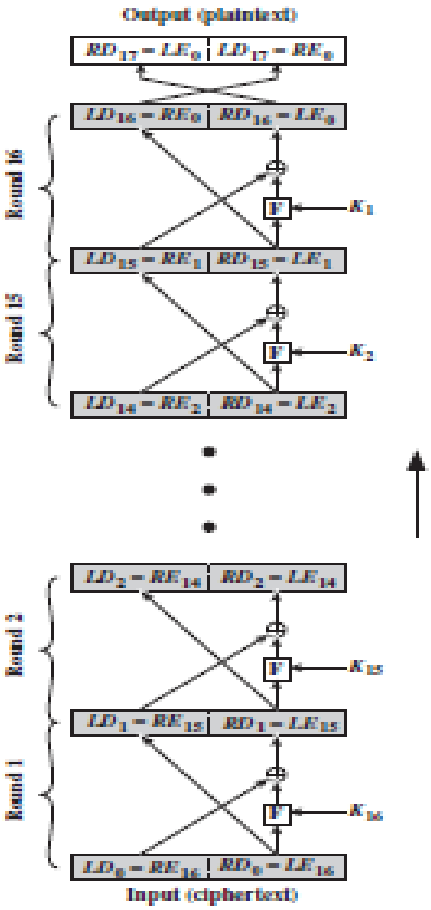


Figure 5.3: Feistel decryption

All rounds have the same structure. A **substitution** is performed on the left half of the data. This is done by applying a *round function* F to the right half of the data and then taking the exclusive-OR of the output of that function and the left half of the data. The round function has the same general structure for each round but is parameterized by the round subkey K_i . Another way to express this is to say that F is a function of right-half block and a subkey, which produces an output value of length *w* bits (F (RE_i, K_{i+1})). Following this substitution, a **permutation** is performed that consists of the interchange of the two halves of the data. This structure is a particular form of the substitution-permutation network proposed by Shannon.

The exact realization of a Feistel network depends on the choice of the following parameters and design features:

• **Block size:** Larger block sizes mean greater security (all other things being equal) but reduced encryption/decryption speed for a given algorithm. The greater security is achieved by greater diffusion. Traditionally, a block size of 64 bits has been considered a reasonable tradeoff and was nearly universal in block cipher design. However, the new AES uses a 128-bit block size.

• **Key size:** Larger key size means greater security but may decrease encryption/ decryption speed. The greater security is achieved by greater resistance to brute-force attacks and greater confusion. Key sizes of 64 bits or less are now widely considered as inadequate, and 128 bits has become a common size.

• **Number of rounds:** The essence of the Feistel cipher is that a single round offers inadequate security but that multiple rounds offer increasing security. A typical size is 16 rounds.

• **Subkey generation algorithm:** Greater complexity in this algorithm should lead to greater difficulty of cryptanalysis.

• **Round function F:** Again, greater complexity generally means greater resistance to cryptanalysis.

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

• **Fast software encryption/decryption:** In many cases, encryption is embedded in applications or utility functions in such a way as to preclude a hardware implementation. Accordingly, the speed of execution of the algorithm becomes a concern.

• **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.

Decryption:

The process of decryption with a Feistel cipher is essentially the same as the encryption process. The rule is as follows: Use the cipher text as input to the algorithm, but use the sub keys K_i in reverse order. That is, use K_n in the first round, K_{n-1} in the second round, and so on, until K_1 is used in the last round. This is a nice feature, because it means we need not implement two different algorithms; one for encryption and one for decryption.

To see that the same algorithm with a reversed key order produces the correct result, Figure 5.2 shows the encryption process going down the left-hand side and the decryption process going up the right-hand side for a 16-round algorithm. For clarity, we use the notation LE_i and RE_i for data traveling through the encryption algorithm (Figure 5.2) and LD_i and RD_i for data traveling through the decryption algorithm (Figure 5.3). The diagram indicates that, at every round, the intermediate value of the decryption process is equal to the corresponding value of the encryption process with the two halves of the value swapped. To put this another way, let the output of the i^{th} encryption round be $LE_i \| RE_i$ Then the corresponding output of the $(16 - i)^{th}$ decryption round is $LD_i \| RD_i$ or, equivalently, $RE_{16-i} \| LE_{16-i}$.

Let us walk through Figure 5.3 to demonstrate the validity of the preceding assertions. After the last iteration of the encryption process, the two halves of the output are swapped, so that the cipher text is $RE_{16}\|LE_{16}$. The output of that round is the cipher text. Now take that cipher text and use it as input to the same algorithm. The input to the first round is $RE_{16}\|LE_{16}$, which is equal to the 32-bit swap of the output of the sixteenth round of the encryption process.

Now we would like to show that the output of the first round of the decryption process is equal to a 32-bit swap of the input to the sixteenth round of the encryption process. First, consider the encryption process. We see that

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

$$RE_{16} = LE_{15} \oplus \mathrm{F}(RE_{15}, K_{16})$$

On the decryption side,

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

$$RD_1 = LD_0 \oplus \mathrm{F}(RD_0, K_{16})$$

$$= RE_{16} \oplus \mathrm{F}(LE_{16}, K_{16}) = RE_{16} \oplus \mathrm{F}(RE_{15}, K_{16})$$

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

Note that the XOR operation has the following properties:

$[A \oplus B] \oplus C = A \oplus [B \oplus C]$

$D \oplus D = 0$

$E \oplus 0 = E$

Therefore, the output of the first round of the decryption process is $RE_{15} \| LE_{15}$, which is the 32-bit swap of the input to the sixteenth round of the encryption. This correspondence holds all the way through the 16 iterations, as is easily shown. We can cast this process in general terms. For the i^{th} iteration of the encryption algorithm,

$$LE_i = RE_{i-1} \text{ and } RE_i = LE_{i-1} \oplus F(RE_{i-1}, K_i)$$

Rearranging terms:

$$RE_{i-1} = LE_i \text{ and } LE_{i-1} = RE_i \oplus F(RE_{i-1}, K_i) = RE_i \oplus F(LE_i, K_i)$$

Thus, we have described the inputs to the i^{th} iteration as a function of the outputs, and these equations confirm the assignments shown in the Figure 5.2.

Finally, we see that the output of the last round of the decryption process is $RE_0 \| LE_0$. A 32-bit swap recovers the original plaintext, demonstrating the validity of the Feistel decryption process.

Note that the derivation does not require that F be a reversible function. To see this, take a limiting case in which F produces a constant output (e.g., all ones) regardless of the values of its two arguments. The equations still hold.

5.4 COMPLETE ENCRYPTION AND DECRYPTION

In the previous section an outline of encryption, decryption process is given. Here we demonstrate the process of encryption, decryption totally. The internal design of F is not necessary for showing encryption or recovery of input text during decryption. To make discussion easy to follow, consider only 4 rounds of encryption.

Encryption:

Input: LE_0, RE_0 (split into two halves)

E1: At the end of round 1, the intermediate cipher text is $LE_1 \| RE_1$ where $LE_1 = RE_0$, $RE_1 = F(RE_0, K_1) \oplus LE_0$

E2: At the end of round 2 the intermediate cipher text is $LE_2 \| RE_2$ where $LE_2 = RE_1$, $RE_2 = F(RE_1, K_2) \oplus LE_1$

E3: After round 3 the intermediate cipher text is $LE_3 \| RE_3$ where $LE_3 = RE_2$, $RE_3 = F(RE_2, K_3) \oplus LE_2$

E4: After round 4 the intermediate cipher text is $LE_4 \| RE_4$ where $LE_4 = RE_3$, $RE_4 = F(RE_3, K_4) \oplus LE_3$

E5: Finally there is left, right swap after completion of 4 rounds giving the output cipher text as $LE_5 \| RE_5$ where $LE_5 = RE_4$, $RE_5 = LE_4$.

Decryption:

Input: $LE_5 \| RE_5 = LD_0 \| RD_0$

Note that $LD_0 = RE_4$, $RD_0 = LE_4$ (Step 5 of encryption - E5)

D1: At the end of round 1, the intermediate text is $LD_1 \| RD_1$, where $LD_1 = RD_0 = LE_4$ $RD_1 = F(RD_0, K_4) \oplus LD_0$.

(i.e. $RD_1 = F(LE_4, K_4) \oplus RE_4$

$= F(RE_3, K_4) \oplus [F(RE_3, K_4) \oplus LE_3]$

$= [F(K_4, RE_3) \oplus F(K_4, RE_3)] \oplus LE_3$ (by E_4)

Thus $RD_1 = LE_3$ and

$LD_1 = LE_4 = RE_3$

D2: After round 2, the intermediate text is $LD_2 \| RD2$ where $LD_2 = RD_1 = LE_3 = RE_2$ (by D_1 and E_3)

$RD_2 = F(RD_1, K3) \oplus LD1$

$= F(LE_3, K_3) \oplus RE_3$ (by D_1)

$= F(LE_3, K_3) \oplus [F(RE_2, K_3) \oplus LE_2]$ (by E_3)

$= [F(LE_3, K_3) \oplus F(LE_3, K_3)] \oplus LE_2$ (by E_3)

$= LE_2$.

Thus $RD_2 = LE_2$ and $LD_2 = RE_2$

D3: After round 3, the intermediate text is

$LD_3 \| RD_3$ where $LD_3 = RD_2 = LE_2 = RE_1$ (by D_2 and E_2)

$RD_3 = F(RD_2, K_2) \oplus LD2$

$= F(LE_2, K_2) \oplus RE_2$ (by D_2)

$= F(RE_1, K_2) \oplus [F(RE_1, K_2) \oplus LE_1]$ (by E_2)

$= [F(RE_1, K_2) \oplus F(RE_1, K_2)] \oplus LE_1$

$= LE_1$

Thus $RD_3 = LE_1$, $LD_3 = RE_1$

D4: At the end of round 4, the intermediate text is

$LD_4 \| RD_4$ where $LD_4 = RD_3 = LE_1 = RE_0$ (by D_3 and E_1)

$RD_4 = F(RD_3, K_1) \oplus LD_3$

$= F(RE_0, K_1) \oplus RE_1$ (by D3 and E1)

$= F(RE_0, K_1) \oplus [F(RE_0, K_1) \oplus LE_0]$ (by E_1)

$= LE_0$

Thus $LD_4 = RE_0$, $RD_4 = LE_0$

D5: Finally there is left, right swap of round 4 output which gives the output $RD_4 \| LD_4 = LE_0 \| RE_0$ = given text.

5.5 SUMMARY

In this unit the basic principle of block cipher is discussed in detail. Feistel block structure being the building block for modern symmetric cipher system is discussed in detail in sections 5.2 and 5.3. The closing section of the unit namely 5.4 shows the working details of encryption, decryption process in Feistel's proposal, by using smaller number of rounds.

5.6 KEYWORDS

Stream cipher, Block cipher, Reversible mapping, Irreversible mapping, Feistel structure, Confusion, Diffusion.

5.7 QUESTIONS

1. Differentiate stream and block ciphers.
2. Give examples for reversible and irreversible mappings.
3. What is the practical problem in using arbitrary reversible mapping? How can this be resolved?
4. Explain the terms confusion and diffusion.
5. Describe the Feistel block structure.
6. Show encryption, decryption process with 4 or more rounds.

REFERENCES

1. Atul Kahate, Cryptography and Network Security, Tata MCGrawHill
2. Behrouz A Forouzan, Cryptography and Network security, McGraw Hill
3. William Stallings, Cryptography and Network Security, Pearson

UNIT -6: DES - DATA ENCRYPTION STANDARD

Structure

6.0 OBJECTIVES

After going through the contents of this unit you will

- ✓ Understand the principle of the most popular symmetric encryption called DES
- ✓ Be able to know the details of each round
- ✓ Be able to perform reduction and expansion of the bits in a block using permutation tables
- ✓ Be able to execute all steps of DES and show encryption and decryption

6.1 DEVELOPMENT OF DES

The most widely used encryption scheme is based on the Data Encryption Standard (DES) adopted in 1977 by the National Bureau of Standards, now called the National Institute of Standards and Technology (NIST), as Federal Information Processing Standard 46 (FIPS PUB 46). The algorithm itself is referred to as the Data Encryption Algorithm (DEA) 7. For DES, data

are encrypted in 64-bit blocks using a 56-bit key. The algorithm transforms 64-bit input in a series of steps into a 64-bit output. The same steps, with the same key, are used to reverse the encryption.

The DES enjoys widespread use. It has also been the subject of much controversy concerning how secure the DES is. To appreciate the nature of the controversy, let us quickly review the history of the DES.

Year	Activity/Project	Development /Outcome	Features
1960	IBM's project in cryptography led by H.Feistel	Algorithm called LUCIFER sold to Bank of London	Block size =64 bits Key size= 128 bits
1970	IBM's effort to market encryption software/hardware. Combined effort of W.Tuchman & C.Meyer and NSA	Refined version of LUCIFER resistant to attacks	Key size =56 bits (can fit in a chip) Block size = 64 bits
1973	NBS invited proposals for encryption standards	IBM submitted refined LUCIFER and was adopted as standard in 1977 & renamed as DES	Changes done to design of s- boxes as suggested by NSA
1994	NIST extended use of DES federal system for 5 years		
1999	NIST recommended Triple DES for federal use		

6.2 OVERVIEW OF FUNCTION OF DES

The overall scheme for DES encryption is illustrated in Figure 6.1. As with any encryption scheme, there are two inputs to the encryption function: the plaintext to be encrypted and the key. In this case, the plaintext must be 64 bits in length and the key is 56 bits in length.

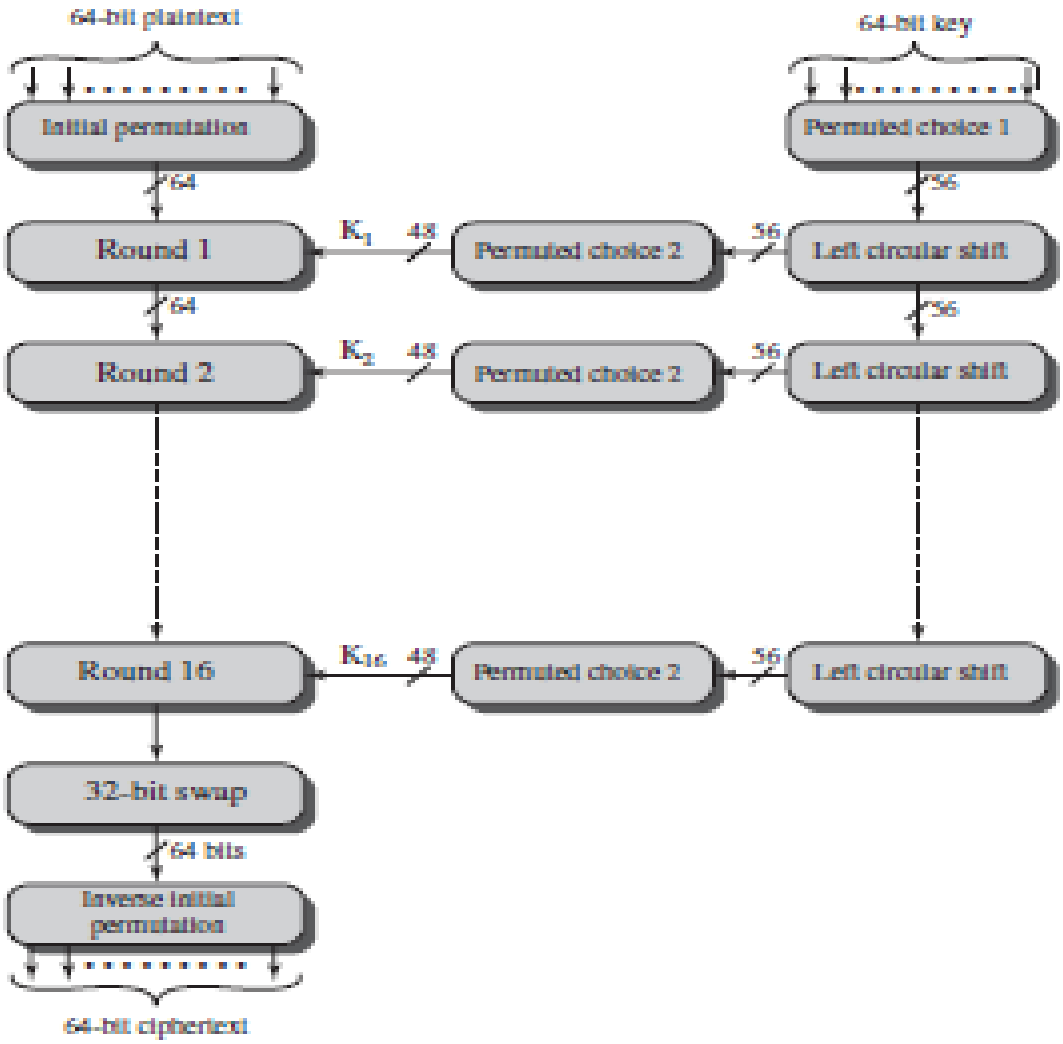


Figure 6.1: General depiction of DES Encryption Algorithm

Looking at the left-hand side of the figure, we can see that the processing of the plaintext proceeds in four phases. (i) First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the *permuted input*. (ii) This is followed by a phase consisting of sixteen rounds of the same function, which involves both permutation and substitution functions. (iii) The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the **pre-output**. (iv) Finally, the pre-output is passed through a permutation that is the inverse of the initial permutation function, to produce the 64-bit cipher text. With the exception of the initial and final permutations, DES has the exact structure of a Feistel cipher.

The right-hand portion of Figure 6.1 shows the way in which the 56-bit key is used. Initially, the key is passed through a permutation function. Then, for each of the sixteen rounds, a *subkey* (*Ki*) is produced by the combination of a left circular shift and a permutation. The permutation function is the same for each round, but a different subkey is produced because of the repeated shifts of the key bits.

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	16	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

Table 6.1: Initial Permutation (IP)

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Table 6.2: Inverse Initial Permutation (IP^{-1})

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25

24	25	26	27	28	29
28	29	30	31	32	1

Table 6.3: Expansion Permutation (E)

17	16	20	21	29	12	28	7
10	1	23	26	5	18	31	15
9	2	24	14	32	27	3	8
25	19	30	6	22	11	4	13

Table 6.4: Permutation Function (P)

6.3 FUNCTION OF DES IN DETAIL

Initial permutation: The initial permutation and its inverse are defined by tables, as shown in Tables 6.1 and 6.2, respectively. The tables are to be interpreted as follows. The input to a table consists of 64 bits numbered from 1 to 64. The 64 entries in the permutation table contain a permutation of the numbers from 1 to 64. Each entry in the permutation table indicates the position of a numbered input bit in the output, which also consists of 64 bits.

To see that the permutation in tables 6.1 and 6.2 are inverses of each other one can find inverse of the inverse permutation and verify that original permutation is recovered.

Details of single round: Figure 6.2 shows the internal structure of a single round. Again, begin by focusing on the left-hand side of the diagram. The left and right halves of each 64-bit intermediate value are treated as separate 32-bit quantities, labeled L (left) and R (right). As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulae:

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

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

The round key is 48 bits. The input is 32 bits. This input is first expanded to 48 bits by using a table that defines a permutation plus an expansion that involves duplication of 16 of the bits (Table 6.3).The resulting 48 bits are XOR-ed with reduced key for the round. This 48-bit result passes through a substitution function that produces a 32-bit output, which is permuted as

defined by Table 6.4. The role of the S-boxes in the function F is illustrated in Figure 6.3.The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output. One of the S boxes (S_1) is defined in Table 6.5.

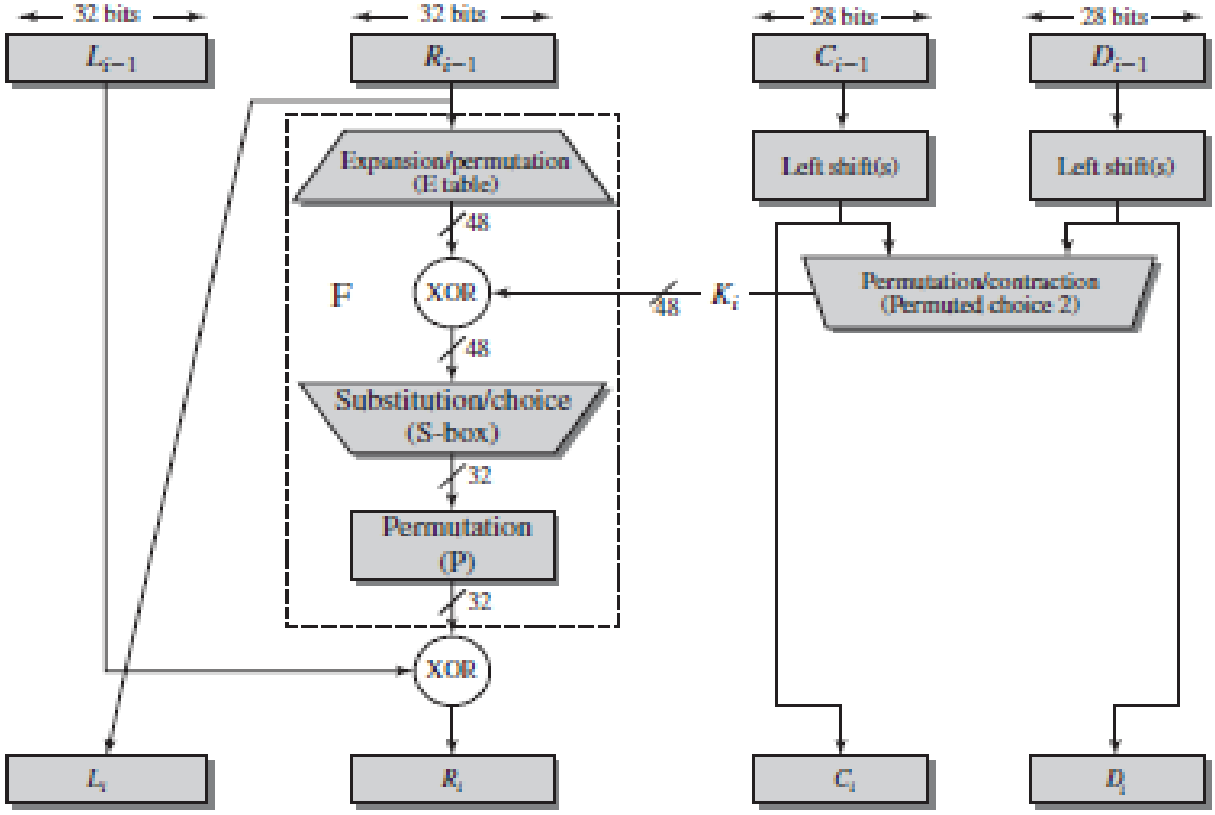


Figure 6.2: Single round of DES algorithm

Function of S boxes can be interpreted as follows: The first and last bits of the input to box form a 2-bit binary number to select one of four substitutions defined by the four rows in the table for S_i. The middle four bits select one of the sixteen columns. The decimal value in the cell selected by the row and column is then converted to its 4-bit representation to produce the output. For example, in S_1, for input 011001, the row is 01 (row 1) and the column is 1100 (column 12) and assuming a value in row 1, column 12 is 9, the output is 1001. The row, column and the output (in decimal) are shown in bold in this Table 6.5.

Each row of an S-box defines a general reversible substitution. Figure 6.3 shows eight S boxes where each S box performs a reduction of 6 bits to 4 bits.

The operation of the S-boxes is worth further comment. Ignore for the moment the contribution of the key (K_i). If you examine the expansion table, you see that the 32 bits of input

are split into groups of 4 bits and then become groups of 6 bits by taking the outer bits from the two adjacent groups. For example, if part of the input string is

...1 0011 1001 1110 0…

This becomes

... **1**0011**1** **1**1001**1** **1**1110**0**...

The outer bit inclusion of previous and subsequent parts is shown in bold font.

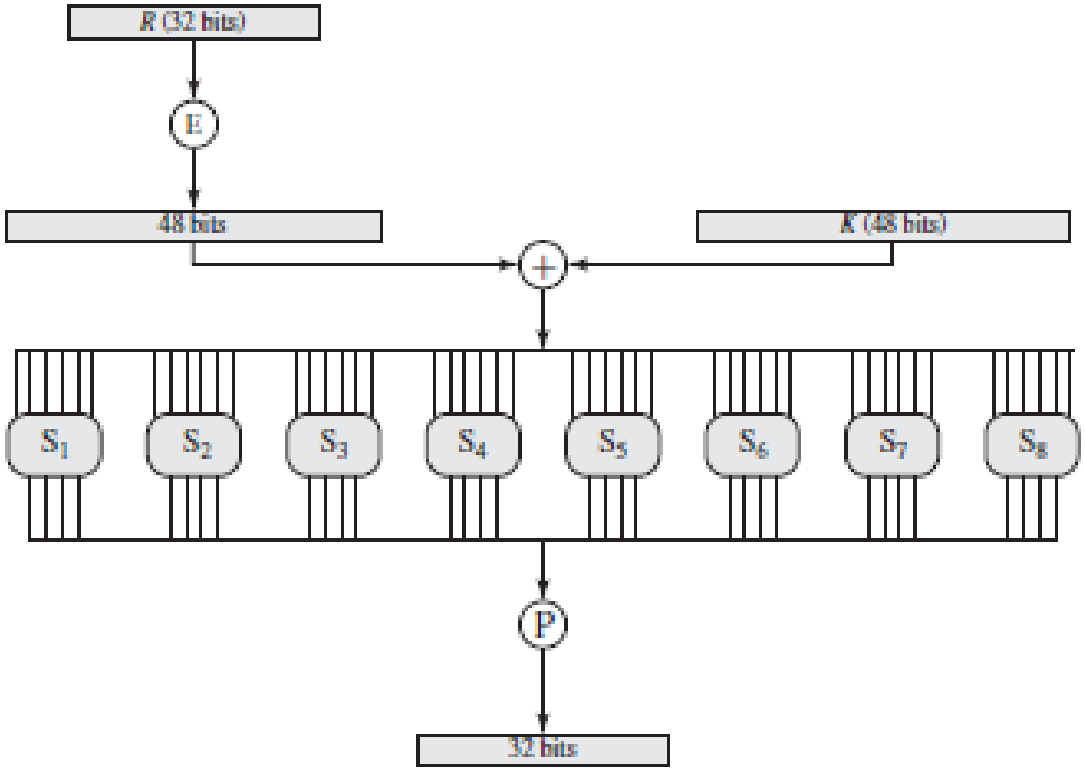


Figure 6.3: Calculation of F(R, K)

The outer two bits of each group select one of four possible substitutions (one row of an S-box). Then a 4-bit output value is substituted for the particular 4-bit input (the middle four input bits). The 32-bit output from the eight S-boxes is then permuted, so that on the next round, the output from each S-box immediately affects as many others as possible.

Row/ Column	0	1	2	3	4	5	6	7	8	9	10	11	**12**	13	14	15
0	14	10	9	13	7	6	11	4	15	2	1	0	3	5	8	12

1	15	5	12	3	11	4	6	7	8	14	0	2	**9**	1	10	13
2	10	9	4	15	14	7	11	6	2	0	1	5	3	12	13	
3	11	7	0	8	4	12	15	13	1	5	3	2	9	10	6	

Table 6.5: S_1 box

Key Generation: Returning to Figures 6.1 and 6.2, we see that a 64-bit key is used as input to the algorithm. The bits of the key are numbered from 1 through 64; every eighth bit is ignored, as indicated in Table 6.6.The key is first subjected to a permutation governed by a table labeled Permuted Choice One (Table 6.7). This is nothing but permutation of numbers (referring to bit positions) in the left of Table 6.6. The resulting 56-bit key is then treated as two 28-bit strings, labeled C_i, D_i. At each round, C_i, D_i are separately subjected to a circular left shift or (rotation) of 1 or 2 bits, as governed by Table 6.8.These shifted values serve as input to the next round. They also serve as input to the part labeled Permuted Choice Two (Table 6.9), which produces a 48-bit output that serves as input to the function F(R_{i-1}, K_i).

Bits included							Bits excluded
1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Table 6.6: Bits included and excluded in reducing the key size

Table 6.7: Permuted Choice One

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

Left shifts	1	1	2	1	2	2	1	1	1	2	2	2	2	1	2	1

Table 6.8: Table of left shift

23	35	18	2	46	59	47	63
38	12	52	22	13	21	45	33
5	47	60	41	11	3	51	1
62	42	10	20	4	9	14	25
58	39	29	53	6	26	31	34
27	44	26	37	41	57	55	7

Table 6.9: Permuted Choice Two

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

DES Decryption:

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

6.4 DES ILLUSTRATION

- Block size be 8 bits
- Key size = 8 bits (all bits of the key are used. No initial reduction)
- Plain text be 0110 1100

* Number rounds be 2
* Other tables for encryption
* T1. Starting permutation $\begin{bmatrix} 1 & 4 & 6 & 8 \\ 3 & 5 & 2 & 7 \end{bmatrix}$
* T2. Inverse of the initial permutation $\begin{bmatrix} 1 & 7 & 5 & 2 \\ 6 & 3 & 8 & 4 \end{bmatrix}$
* Half the block size = 4
* This has to be expanded to 8 bits to do XOR with key
* The expansion table is
* T3. $\begin{bmatrix} 4 & 1 & 2 & 3 \\ 2 & 3 & 4 & 1 \end{bmatrix}$
* Two S boxes

S1. $\begin{bmatrix} 0 & 2 & 1 & 3 \\ 1 & 0 & 3 & 2 \\ 1 & 3 & 2 & 0 \\ 3 & 0 & 2 & 1 \end{bmatrix}$

S2. $\begin{bmatrix} 1 & 3 & 0 & 2 \\ 0 & 2 & 3 & 1 \\ 0 & 3 & 0 & 2 \\ 2 & 1 & 0 & 3 \end{bmatrix}$

* T4. Permutation of 4 bits

  $\begin{bmatrix} 2 & 1 & 4 & 3 \end{bmatrix}$

* T5. Key shift table

Round	1	2
Shift	2	1

DES Encryption steps

Operations done at the beginning and end of all rounds

1. Permute data using T1 and then start round 1 (starting operation)
2. After all rounds swap L and R and use inverse permutation as per T2 (end operation)

Operations in each round

1. Divide input data into two halves L_{i-1} and R_{i-1}
2. Expand R_{i-1} using T3
3. Preparation of key

A) Divide key into two halves C_{i-1} and D_{i-1}

B) Left shift both C_{i-1} and D_{i-1}

4. XOR outputs at step 2 and 3
5. Perform S box operations to get 4 bit output
6. Permute 4 bit output using T4
7. XOR L_{i-1} and output at step 6

$L_i = R_{i-1}$ and R_i = output at step 7

Beginning operation

T1 on plain text gives the string 00101110

Round 1

1. L_0 =0010, R_0=1110
2. Expanded R_0 (using T3) is 01111101
3. Key preparation

A) C_0 = 0110 and D_0 = 1010

B) Left shift using T5 gives 1001 (C_1) and 1010 (D_1)

K_1 = 10011010

4. Expanded $R_0 \oplus K_1$ = 01111101 $\oplus$

10011010 = 11100111

5. S box reduction

1110 returns 00 (entry in the cell 10, 11 is 0)

0111 returns 01 (entry in the cell 01, 11 is 1)

So reduced 4 bit string is 0001

6. Permute this using T4 to get 0010

7. R_1 is 0010 $\oplus$ 0010 (L_0) = 0000

and L_1 is R_0 =1110

Round 2

1. L_1 =1110, R_1=0000

2. Expanded R_1(using T3) is 00000000

3. Key preparation

A) C_1 = 1001 and D_1 = 1010

B) Left shift using T5 gives 0011 (C_2) and 0101 (D_2)

K_2 = 00110101

4. Expanded $R_1 \oplus K_2$ = 00000000 $\oplus$

00110101 = 00110101

5. S box reduction

0011 returns 00 (entry in the cell 01, 01 is 0)

0101 returns 11 (entry in the cell 01, 10 is 3)

So reduced 4 bit string is 0001

6. Permute this using T4 to get 0011

7. R_2 is 0011 $\oplus$ 1110 (L_1) = 1101

and L_2 is R_1 =0000

Ending operations

Swap L_2 and R_2 to get 11010000

Invert starting permutation as per T2 to get the output cipher text as 10010001

DES Decryption on the example

All operations of encryption done including starting and ending operations using the keys in reverse order

Cipher text is 10010001

Starting operation:

Permute as per T1 to get the string 11010000

Round 1

1. L_0 = 1101 R_0 = 0000 (observe these are L_2 and R_2 of encryption)
2. Expand R_0 as per T3 to get 00000000
3. Use K_2 = 00110101
4. XOR of K_2 and as R_0 . The new string is 00110101
5. S box reduction 0011 gives 00 and 0101 gives 11
6. Permute 0011 as per T4 to get 0011
7. XOR of 0011 and 1101 is 1110 = R_1 and L_1 = R_0 = 0000

Round 2

1. L_1= 0000, R_1 = 1110
2. Expand R_1 as per T3 to get 01111101
3. Use K_1 = 10011010
4. XOR of K_1 and as R_1 . The new string is 11100111
5. S box reduction 1110 gives 00 and 0111 gives 01
6. Permute 0001 as per T4 to get 0010
7. XOR of 0010 and 0000 is 0010 = R_2 and L_2 = R_1 = 1110

Ending operations

Swap L_2 and R_2 to get 00101110

Inverse of initial permutation yields 01101100 which is the given plain text

6.5 SUMMARY

This unit has a detailed description of very widely used symmetric encryption called DES. Important milestones in the development of DES algorithm is given in section 2.1.Overview of DES function can be found in 2.2. Description of each step of DES is discussed in the section 2.3. Finally in the closing section of this unit an illustration of DES for a small block size is shown.

6.6 KEYWORDS

Symmetric encryption, Block cipher, DES, Permutation and its inverse, S-box bit reduction, Bit expansion, Shifting bits

6.7 QUESTIONS

1. Explain Feistel block structure.
2. Show that plain text can be retrieved in Feistel block cipher method.
3. Write about the development of DES.
4. Using a flowchart discuss the overall function of DES.
5. Write a table of permutation of 16 bits and its inverse. Show permutation of an example 16 bit block and find its inverse.
6. Using figures explain the steps in a single round.
7. Illustrate DES on an 8 bit block. Assume a 6 bit key.
8. Mention the advantages of using S boxes.

6.8 REFERENCES

1. Atul Kahate, Cryptography and Network Security, Tata MCGrawHill
2. Behrouz A Forouzan, Cryptography and Network security, McGraw Hill
3. William Stallings, Cryptography and Network Security, Pearson

UNIT -7: ATTACKS ON DES AND MULTIPLE ENCRYPTIONS

Structure

7.0 OBJECTIVES

When you have understood the topics discussed in this unit you will be familiar with

- ✓ Strength of DES
- ✓ Special attacks on DES
- ✓ Ways to strengthen DES
- ✓ Issues in design of block cipher
- ✓ Advantages of multiple encryption

7.1 STRENGTH OF DES

Since its adoption as a federal standard, there have been serious concerns about the level of security that DES provides. These concerns, by and large, fall into two areas: key size and the nature of the algorithm.

The Use of 56-Bit Keys

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

half the key space has to be searched, a single machine performing one DES encryption per microsecond would take more than a thousand years to break the cipher. The justification for this statement is given as follows:

Note that the minimum and maximum number of keys to be used in a brute force attack is 0 and 2^{56}. Thus 2^{55} (= $(0+2^{56})/2$) is the average number of keys used to break the cipher. At the rate of 1 encryption in a microsec (=$1/10^6$ sec), the average time of brute force attack = $2^{55} \times 1/10^6$ sec = $2^{55} \times 1/10^6 \times 1/60$ (minutes) × 1/60 (hours) × 1/24 (days) × 1/365 (years) ~ 1142 years.

However, the assumption of one encryption per microsecond is overly conservative. In 1977, Diffie and Hellman postulated that the technology existed to build a parallel machine with 1 million encryption devices, each of which could perform one encryption per microsecond. The authors also quoted an estimate of 420 million for such a sophisticated machine. But this would bring the average search time down to about 10 hours. The calculation of time is given here.

The average number of keys to be tried in a brute force attack is 2^{55}. The rate of encryption/decryption being 1 million per second, the time needed for attack = $2^{55} \times 1/10^6$ (one encryption time) $\times 1/10^6$ (with 1 million parallel encryptions) × 1/60 (minutes) × 1/60 (hours) ~ 10 hours.

DES was shown to be grossly insecure in July 1998, when the Electronic Frontier Foundation (EFF) announced that it had broken a DES encryption using a special-purpose "DES cracker" machine that was built with less than $250,000. EEF reported that the attack took less than three days. The EFF has published a detailed description of the machine, so that others can build their own cracker. With the trend of falling hardware prices and faster machines, DES could virtually be worthless very soon. It is important to note that there is more to a key-search attack than simply running through all possible keys. Unless known plaintext is provided, the analyst has additional problem of recognizing plaintext as plaintext. If the message is just plain text in English, then the result pops out easily, although the task of recognizing English would have to be automated. If the text message has been compressed before encryption, then recognition is more difficult. And if the message is some more general type of data, such as a numerical file, and this has been compressed, the problem becomes even more difficult to automate. Thus, to supplement the brute-force approach, some degree of knowledge about the expected plaintext is needed, and some means of automatically distinguishing plaintext from

garble is also needed. The EFF approach addresses this issue as well and introduces some automated techniques that would be effective in many contexts. Fortunately, there are a number of alternatives to DES, the most important of which are AES and triple DES, and in the end of this unit we will discuss the attacks on multiple encryptions as well.

The Nature of the DES Algorithm

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

Timing Attacks

A timing attack is one in which information about the key or the plaintext is obtained by observing how long it takes for a given implementation to perform decryptions on various cipher texts. A timing attack exploits the fact that an encryption or decryption algorithm often takes slightly different amounts of time on different inputs. There are reports on an approach that yields the Hamming weight (number of bits equal to one) of the secret key. However this is a long way from knowing the actual key, but it is an intriguing first step. It has been concluded by attack proposals that DES is fairly resistant to a successful timing attack but suggest some avenues to explore. Although this is an interesting line of attack, it so far appears unlikely that this technique will ever be successful against DES or more powerful symmetric ciphers such as triple DES and AES.

7.2 ATTACKS ON DES

For most of its life, the prime concern with DES has been its vulnerability to brute-force attack because of its relatively short (56 bits) key length. However, there has also been interest in

finding cryptanalytic attacks on DES. With the increasing popularity of block ciphers with longer key lengths, including triple DES, brute-force attacks have become increasingly impractical. Thus, there has been increased emphasis on cryptanalytic attacks on DES and other symmetric block ciphers. In this section, we provide a brief overview of the two most powerful and promising approaches: differential cryptanalysis and linear cryptanalysis.

Differential Cryptanalysis

One of the most significant advances in cryptanalysis in recent years is differential cryptanalysis. In this section, we discuss the technique and its applicability to DES. Differential cryptanalysis is the first published attack that is capable of breaking DES in less than 2^{55} encryptions. The scheme as reported by Biham and Shamir in 1993, can successfully crypt analyze DES with an effort on the order of 2^{47} encryptions, requiring 2^{47} chosen plaintexts. Although 2^{47} is certainly significantly less than 2^{55}, the need for the adversary to find 2^{47} chosen plaintexts makes this attack of only theoretical interest.

Although differential cryptanalysis is a powerful tool, it does not do very well against DES. The reason is that differential cryptanalysis was known to the team as early as 1974. The need to strengthen DES against attacks using differential cryptanalysis played a large part in the design of the S-boxes and the permutation P. Differential cryptanalysis of an eight-round LUCIFER algorithm requires only 256 chosen plaintexts, whereas an attack on an eight-round version of DES requires 2^{14} chosen plaintexts.

Differential Cryptanalysis Attack:

The differential cryptanalysis attack is complex. Here, we provide a brief overview so that you can get a flavor of the attack. The rationale behind differential cryptanalysis is to observe the behavior of pairs of text blocks evolving along each round of the cipher, instead of observing the evolution of a single text block.

We begin with a change in notation for DES. Consider the original plaintext block m to consist of two halves m0, m1. Each round of DES maps the right-hand input into the left-hand output and sets the right-hand output to be 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. If we label each new block m_i, i=2 to 17, then the intermediate message halves are related as follows:

$$m_{i+1} = m_{i-1} + f(m_i, K_i), \quad i = 1, 2, \ldots, 16$$

In differential cryptanalysis, we start with two messages, m and m', with a known XOR difference dm = m ⊕ m', and consider the difference between the intermediate message halves: $dm_i = m_i \oplus m_i'$. Then we have

$$\begin{aligned} dm_{i+1} &= m_{i+1} \oplus m'_{i+1} \\ &= [m_{i-1} \oplus f(m_i, K_i)] \oplus [m'_{i-1} \oplus f(m_i', K_i)] \\ &= dm_{i-1} \oplus [f(m_i, K_i) \oplus f(m_i', K_i)] \end{aligned}$$

Now, suppose that many pairs of inputs to f with the same difference yield the same output difference if the same sub key is used. To put this more precisely, let us say that X may cause Y with probability p, if for a fraction p of the pairs in which the input XOR is X, the output XOR equals Y. We want to suppose that there are a number of values of that have high probability of causing a particular output difference. Therefore, if we know dm_{i-1} and dm_i with high probability, then we know dm_{i+1} with high probability. Furthermore, if a number of such differences are determined, it is feasible to determine the sub-key used in the function f.

The overall strategy of differential cryptanalysis is based on these considerations for a single round. The procedure is to begin with two plaintext messages *m* and *m'* with a given difference and trace through a probable pattern of differences after each round to yield a probable difference for the cipher text. Actually, there are two probable patterns of differences for the two 32-bit halves. Next, we submit the plain text for encryption to determine the actual difference under the unknown key and compare the result to the probable difference. If there is a match, then we suspect that all the probable patterns at all the intermediate rounds are correct. With that assumption, we can make some deductions about the key bits. This procedure must be repeated many times to determine all the key bits.

$$E(K, m) \oplus E(K, m') = (dm_{17} || dm_{16})$$

Linear cryptanalysis

A more recent development is linear cryptanalysis. This attack is based on finding linear approximations to describe the transformations performed in DES. This method can find a DES key given known plaintexts, as compared to chosen plaintexts for differential cryptanalysis. Although this is a minor improvement, because it may be easier to acquire known plaintext rather than chosen plaintext, it still leaves linear cryptanalysis infeasible as an attack on DES. So far, little work has been done by researchers to validate the linear cryptanalytic approach.

7.3 BLOCK CIPHER DESIGN ISSUES

S box

- No output bit of any S box should be linear function of input bits.
- If two inputs to any S box differ by a bit out should differ by at least 2 bits.

F

- Should have good avalanche effect (SAC, GA and BIC)
- SAC – Strict avalanche effect: Any output of an S box should change with probability ½ when a single bit of input is changed
- Bit independence criteria states that bits at j and k should change independently when input i is inverted
- GA – Guaranteed avalanche effect: GA of order x means that a single bit input change result in change of x bits of output always. GA ensures good diffusion
- Difficult to meet these conditions for large S box
- We then use random number generation for S box entries and do tests on random numbers

Permutation

- Four output bits from each S box at round I are distributed so that two of them affect middle two bits and other two affect end bits

Number of rounds

- More rounds is better. DES and LUCIFER uses 16 rounds, a kind of optimal number

Key schedule

- Guessing any individual sub keys should be very difficult
- One suggestion is to make algorithm of key generation satisfy SAC and GA

7.4 MULTIPLE ENCRYPTIONS

DES is known to be vulnerable to many types of attacks. One alternative is to go for a new algorithm such as AES. Another option is to use the investment in available DES. DES can be best used to do multiple encryptions with multiple keys. First, plaintext is converted to cipher

text using the encryption algorithm and cipher is used as input for a second time encryption. This process may be repeated through any number of stages.

Double DES

DES can be done twice for additional security. The procedure of double encryption/decryption is describes as follows:

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

Note that the key size is now 112 bits.

Encryption:

$$P \xrightarrow{K_1} X \xrightarrow{K_2} C$$

Decryption:

$$C \xrightarrow{K_2} X \xrightarrow{K_1} P$$

Strength of double DES

With double DES the cipher text is obtained as $E(K_2, E(K_1, P)) = C$. one may wonder it is after all encryption done with a new key perhaps K_3. That is $E(K_2, E(K_1, P)) = E(K_3, P)$. If this is so, there is no difference between single or double encryption. It is easy to shoe that equation above is not true.

A single DES has key space of 2^{56} . Whereas with double DES and block size of 64 we have 2^{64} input blocks and the number of mappings possible is 2^{64}! Note that,

$2^{64}! > 10^{10}{}^{20}$ and $2^{56} < 10^{17}$

Thus DES used twice will provide many more mappings that are defined by single DES.

Attack on double DES

A special type of attack called meet in the middle attack is useful for double DES. Details of this attack are given here.

Suppose a P, C pair is known

Try various possible keys for K_1 and do encryption on P at the same time decrypt C with various possible K_2. Stop when $X = E(K_1, P) = D(K_2, C)$.

Probably you have discovered the pair of keys K_1 and K_2. To confirm this is the right the pair of keys, encrypt P with these two keys and see if get C. If not continue with decryption. A systematic way of the attack is described in the following steps:

1. Use all 2^{56} keys on P and encrypt
2. Store the result in a table in the sorted order of ciphers (X) produced
3. Use systematically the values for K_2 and decrypt C
4. Check the result of decryption against table value
5. If match occurs then K_1 and K_2 could be pair of keys used
6. Confirm this by doing double DES on P
7. If the result is C then keys are discovered
8. If not continue with decryption

One of the keys K_2 that gives a value in the table of encryption is the pair that is used in Double DES. Complexity of the attack is $2^{56} + 2^{54}$ (average) which is not very much greater compared to 2^{55} required by brute force attack on single DES.

Triple DES

Obvious counter measure to thwart meet in the middle attack for double DES is go more encryptions. Triple DES with three keys is a solution. The encryption and decryption with three keys is shown here.

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

Encryption:

$$P \xrightarrow{K_1} X \xrightarrow{K_2} Y \xrightarrow{K_3} C$$

Decryption:

$$C \xrightarrow{K_3} Y \xrightarrow{K_2} X \xrightarrow{K_1} P$$

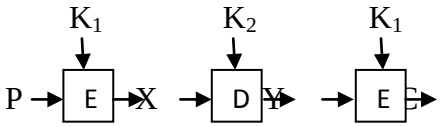
Strength of double DES

Key size will increase to 56 x 3 = 168 bits. Meet in the middle attack would now require 2^{112} trials. This is not practical now and far into the future. The major drawback with triple DES is unwieldy key size (168 bits). Tuchman (1979) proposed an attractive alternative to this namely

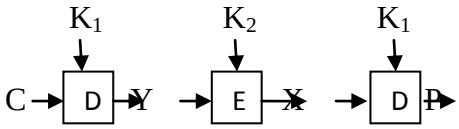
Triple DES with 2 keys. This is currently in use by key management standards ANS X9.17 and ISO 8732. Triple DES with two keys is shown below.

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

Encryption:



Decryption:



By using encryption and decryption alternatively it is possible to reduce this to single DES (usable as single DES also). There is no cryptographic significance of E, D and E. With $K_1=K_2$ it is simply single DES. A single user DES can encode P using key once and triple DES user can decode it with using K_1 three times. A triple DES user can do E, D, E with one key and single DES user can decrypt the file by using key once.

Attack on Triple DES

No practical cryptanalytic attacks have been reported so far. Coppersmith notes that cost of brute force attack is 2^{112} and that of differential cryptanalytic attacks suffers from exponential growth of the order of 10^{52}. Merkle and Hellman (1981) proposed finding plain text which makes intermediate encryption A as 0. But this proposal is not practical.

7.5 SUMMARY

In this unit you will find discussion on strength and weakness of DES, two types of attacks on encryption algorithms and its performance on DES in sections 3.1 and 3.2. The later sections describe ways to strengthen DES further in section 3.3 design issues of various function in a

single round are detailed. In the final section of this unit an important multiple encryption method called triple DES id discussed.

7.6 KEYWORDS

Differential cryptanalysis attack, linear cryptanalsis attack, timing attacks, DES design criteria, strict avalance, guaranteed avalance, bit independence, multiple encryption, double DES, triple DES.

7.7 QUESTIONS

1. Write about the strength of DES.
2. Discuss timing attacks.
3. Describe differential cryptanalysis attack on DES.
4. Briefly discuss the principle of linear cryptanalysis.
5. Writ note on strict avalanche, bit independence, guaranteed avalanche, random number generation.
6. Discuss double and triple DES.
7. Show that triple DES with two keys can be used for three or one encryption.
8. Describe attacks on double and triple DES.

7.8 REFERENCES

1. Atul Kahate, Cryptography and Network Security, Tata MCGrawHill
2. Behrouz A Forouzan, Cryptography and Network security, McGraw Hill
3. William Stallings, Cryptography and Network Security, Pearson

UNIT - 8: STREAM CIPHERS

Structure

8.0 OBJECTIVES

A good understanding of the topics discussed in this unit will make you knowledgeable in

- ✓ Operating principles of a stream cipher
- ✓ An important stream cipher method RC4
- ✓ Analyzing the strength of RC4
- ✓ Ways to generate random numbers

8.1 OPERATION OF STREAM CIPHERS

A typical stream cipher encrypts plaintext one byte at a time; although a stream cipher may be designed to operate on one bit at a time or on units larger than a byte at a time. Figure 8.1 shows the operation of a stream cipher. In stream ciphers, a key is input to a pseudorandom bit generator that produces a stream of 8-bit numbers that are apparently random. The output of the generator, called a **key stream**, is combined one byte at a time with the plaintext stream using

the bitwise exclusive-OR (XOR) operation. For example, if the next byte generated by the generator is 10011100 and the next plaintext byte is 01001110, then the resulting cipher text byte is

```
  01001110    plaintext
⊕ 10011100    key stream
____________
  11010010    ciphertext
```

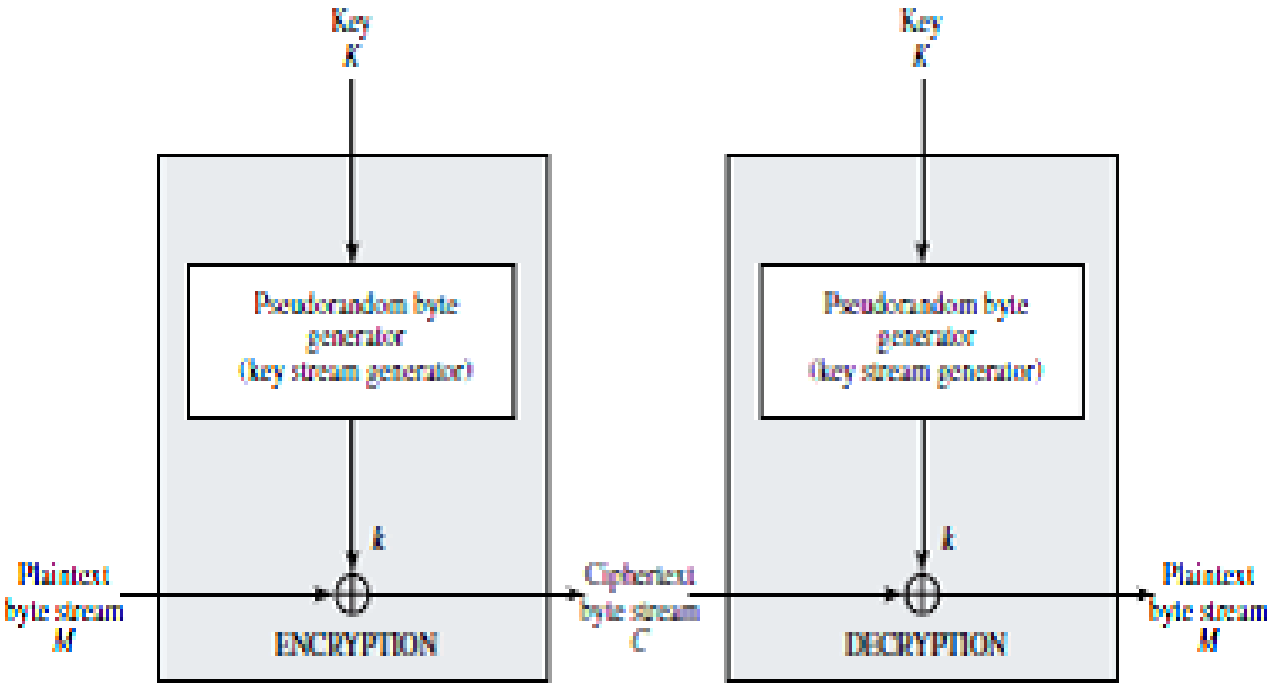


Figure 8.1 Stream cipher diagram

Decryption requires the use of the same pseudorandom sequence:

```
  11010010    ciphertext
⊕ 10011100    key stream
____________
```

01001110 plaintext

The stream cipher is similar to the one-time pad discussed in module 1. The difference is that a one-time pad uses a genuine random number stream, whereas a stream cipher uses a pseudorandom number stream.

The following are some important design considerations for a stream cipher.

1. The encryption sequence should have a large period. Note that the XOR of same plain text block (blocks being small it is possible that blocks repeat very often) and the same key from the generator will result in identical cipher blocks. A pseudorandom number generator uses a function that produces a deterministic stream of bits that eventually repeats. The longer the period of repeat the more difficult it will be to do cryptanalysis. This is essentially the same consideration that was discussed with reference to the Vigenère cipher, namely that the longer the keyword the more difficult the cryptanalysis.
2. It is desirable that the key stream should approximate the properties of a true random number stream as close as possible. For example, there should be an approximately equal number of 1s and 0s. If the key stream is treated as a stream of bytes, then all of the 256 possible byte values should appear approximately equally often. If the key stream is close truly random sequence, the more randomized will the cipher text be, making cryptanalysis more difficult.
3. Note from Figure 8.1 that the output of the pseudorandom number generator is conditioned on the value of the input key. To guard against brute-force attacks, the key needs to be sufficiently long. The same considerations that apply to block ciphers are valid here. Thus, with current technology, a key length of at least 128 bits is desirable.

With an efficiently designed pseudorandom number generator, a stream cipher can be as secure as a block cipher of comparable key length. A potential advantage of a stream cipher is that stream ciphers that do not use block ciphers as a building block are typically faster and use far less code than do block ciphers. The example in this chapter, RC4, can be implemented in just a few lines of code. In table 8.1, a comparison of execution times of RC4 and three popular symmetric block ciphers is given. One advantage of a block cipher is that you can reuse keys. In contrast, if two plaintexts are encrypted with the same key using a stream cipher, then cryptanalysis is often quite simple. If the two cipher text streams are XOR-ed together, the result

is the XOR of the original plaintexts. If the plaintexts are text strings, credit card numbers, or other byte streams with known properties, then cryptanalysis is easy.

For applications that require encryption/decryption of a stream of data, such as over a data communications channel or a browser/Web link, a stream cipher might be the better alternative. For applications that deal with blocks of data, such as file transfer, e-mail, and database, block ciphers may be more appropriate. However, either type of cipher can be used in virtually any application.

A stream cipher can be constructed with any cryptographically strong random numbers such as the ones we will shortly discuss in this unit. In the next section, we look at a stream cipher that uses a function to generate random numbers designed specifically for the stream cipher.

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

Table 8.1: Speed of encryption of various cipher schemes

8.2 RC4

Ron Rivest, one of the authors of asymmetric encoding method RSA, is the designer of RC4. The official name for this algorithm is "Rivest cipher 4". However because of its ease of reference the name "RC4" has stuck. RC4 is widely used in Wired Equivalent Privacy (WEP) protocol and WPA (WiFi Protected Access) protocol. The reason for its wide deployment is its speed and simplicity. Both hardware and software implementations are possible.

RC4 is a stream cipher designed in 1987 by Ron Rivest for RSA Security. It is a variable key size stream cipher with byte-oriented operations. The algorithm is based on the use of a random permutation. Analysis shows that the period of the cipher is overwhelmingly likely to be greater than 10^{100}. Just eight to sixteen machine operations are required per output byte, and

hence the cipher can run very quickly in software also. RC4 is used in the Secure Sockets Layer/Transport Layer Security (SSL/TLS) standards that have been defined for communication between Web browsers and servers.

The RC4 algorithm is remarkably simple and quite easy to explain. A variable-length key of from 1 to 256 bytes (8 to 2048 bits) is used to initialize a 256-byte state vector S, with elements S[0], S[1],..., S[255] . At all times, S contains a permutation of all 8-bit numbers from 0 through 255. For encryption and decryption, a byte *k* is generated from S by selecting one of the 255 entries in a systematic fashion. As each value of *k* is generated, the entries in S are once again permuted.

Initialization of S:

To begin, the entries of are set equal to the values from 0 through 255 in ascending order; that is, S[0] = 0, S[1] = 1, … , S[255]=255 . A temporary vector, T, is also created. The creation of the temporary vector T is as follows: If the length of the key is 256 bytes, then 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. These preliminary operations can be summarized as

/* **Initialization** */

for i = 0 **to** 255 **do**

S[i] = i;

T[i] = K[i **mod** keylen];

Next we use T to produce the initial permutation of S. This involves starting with S[0] with and going through to S[255] , and for each S[i], swapping S[i] with another byte in S according to a scheme dictated by T[i];

/* **Initial Permutation of S** */

j = 0;

for i = 0 **to** 255 **do**

j = (j + S[i] + T[i]) **mod** 256;

Swap (S[i], S[j]);

Because the only operation on S is a swap, the only effect is a permutation.

S still contains all the numbers from 0 through 255.

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], and for each S[i], swapping 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 over again at S[0].

/* **Stream Generation** */

i, j = 0;

while (true)

i = (i + 1) **mod** 256;

j = (j + S[i]) **mod** 256;

Swap (S[i], S[j]);

t = (S[i] + S[j]) **mod** 256;

k = S[t];

To encrypt, XOR the value k with the next byte of plaintext. To decrypt, XOR the value k with the next byte of cipher text. Figure 8.2 illustrates the RC4 logic

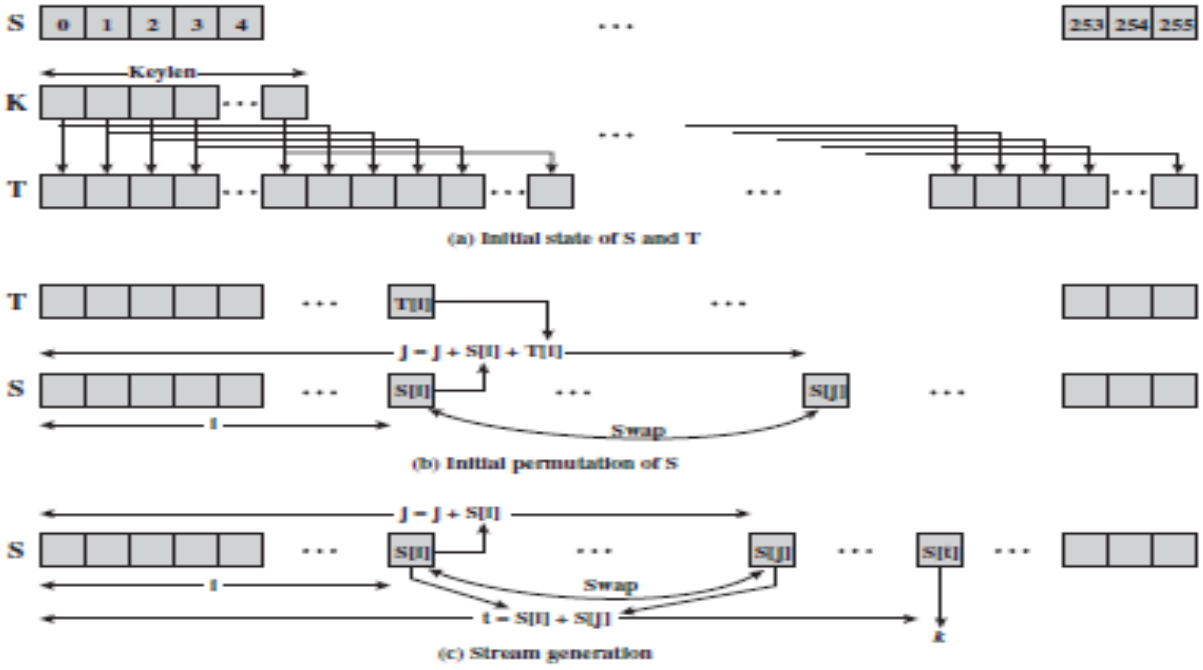


Figure 8.2 RC4

Strength of RC4

A number of attempts have been made attacking RC4. None of these is practical if key size is ≥ 128 bits. However, there is an attack to RC4 on WEP. The problem is not with EC4 but with input random number. But this can be remedied in WEP by changing the way in which keys are generated.

8.3 RC5

RC5 is a symmetric key block encryption algorithm developed by Ron Rivest. The main features of RC5 are that it is quite fast as it uses only the primitive computer operations (such as addition, XOR, shift, etc). It allows for a variables number of rounds and a variable bit-size key to add to the flexibility. Different applications that demand varying security needs can set these values accordingly. Another important aspect is that RC5 requires less memory for execution and is, therefore, suitable not only for desktop computers, but also for smart cards and other devices that have a small memory capacity.

8.4 RANDOM NUMBER GENERATION

Random number is an essential ingredient for many cryptographic systems including RC4.
In this section we discuss some methods for random number generation.

True random number generation

A true random number generator (TRNG) uses a nondeterministic source to produce randomness. Most operate by measuring unpredictable natural processes, such as pulse detectors of ionizing radiation events, gas discharge tubes, and leaky capacitors. Intel has developed a commercially available chip that samples thermal noise by amplifying the voltage measured across undriven resistors.

The following are possible sources of randomness that, with care, easily can be used on a computer to generate true random sequences.

• **Sound/video input:** Many computers are built with inputs that digitize some real-world analog source, such as sound from a microphone or video input from a camera. The "input" from a sound digitizer with no source plugged in or from a camera with the lens cap on is essentially thermal noise. If the system has enough gain to detect anything, such input can provide reasonably high quality random bits.

• **Disk drives:** Disk drives are known to have small random fluctuations in their rotational speed due to chaotic air turbulence. The addition of low-level disk seek-time instrumentation produces a series of measurements that contain this randomness. Such data is usually highly correlated, so significant processing is needed. Nevertheless, experimentation a decade ago showed that, with such processing, even slow disk drives on the slower computers of that day could easily produce 100 bits a minute or more of excellent random data.

A TRNG may produce an output that is biased in some way, such as having more ones than zeros or vice versa. Various methods of modifying a bit stream to reduce or eliminate the bias have been developed. These are referred to as **de-skewing** *algorithms*. One approach to de-skew is to pass the bit stream through a hash function. The hash function produces an n-bit output from an input of arbitrary length. For de-skewing, blocks of m input bits with $m \geq n$, can be passed through the hash function. TRNG is too tedious and complex pseudo random number generators (formula) are available in plenty. In this section we will discuss some important ones very briefly.

Pseudo Random Number Generators (PRNG)

We discuss here two important ways of generating random numbers (hence called pseudo random numbers).

Linear Congruential Generators:

A widely used technique for pseudorandom number generation is an algorithm first proposed by Lehmer, which is known as the linear congruential method. The algorithm is parameterized with four numbers, as follows:

m	the modulus	$m > 0$
a	the multiplier	$0 < a < m$
c	the increment	$0 \leq c < m$

X_0 the starting value, or seed $0 \leq X_0 < m$

The sequence of random numbers is obtained using the following iterative equation:

$$X_{n+1} = (aX_n + c) \bmod m$$

If m, a, c, and X_0 are integers, then this technique will produce a sequence of integers with each integer in the range $0 \leq X_n < m$.

The selection of values for a, c, and m is critical in developing a good random number generator. For example, consider a=c=1. The sequence produced is obviously not satisfactory. This is because if X_0 is smaller than m then all other X_n are same as X_0. The period of this generator is 1. Now consider the values a = 7, c = 0, m = 32, and X_0 = 1. This generates the sequence {7, 17, 23, 1, 7, etc.}, which is also clearly unsatisfactory. Of the 32 possible values, only four are generated; thus, the sequence is said to have a period of 4. If, instead, we change the value to 5, then the sequence is {5, 25, 29, 17, 21, 9, 13, 1, 5, etc.}, which increases the period to 8.

We would like m to be very large, so that there is the potential for producing a long series of distinct random numbers. A common criterion is that m can be nearly equal to the maximum representable nonnegative integer for a given computer. Thus, a value of m near to or equal to2^{31} is typically chosen.

Three important tests to be used in evaluating a random number generator are:

T1: The function should be a full-period generating function. That is, the function should generate all the numbers between 0 and m before repeating.

T2: The generated sequence should appear random.

T3: The function should implement efficiently with 32-bit arithmetic.

With appropriate values of a, c, and m, these three tests can be passed. With respect to T1, it can be shown that if m is prime and c=0, then for certain values of a the period of the generating function is m - 1, with only the value 0 missing. For 32-bit arithmetic, a convenient prime value of m is (2^{31} - 1). Thus, the generating function becomes

$$X_{n+1} = (aX_n) \bmod (2^{31} - 1)$$

The largest the machine can accommodate is $2^{31} – 1$. With this choice of m, there is no necessity to do division operation to find (aX_n) mod (2^{31} - 1). The product of a and X_n is computed bit by

bit from the least significant bit. If the product fits in 32 bits then this the answer of mod (2^{31} - 1). If it does not, the least significant bits that can fit in 32 bits is the result of mod (2^{31} - 1).

Of the more than 2 billion possible choices for a, only a handful of multipliers pass all three tests. One such value is $a = 7^5 = 16807$, which was originally selected for use in the IBM 360 family of computers. This generator is widely used and has been subjected to a more thorough testing than any other PRNG. It is frequently recommended for statistical and simulation work.

The strength of the linear congruential algorithm is that if the multiplier and modulus are properly chosen, the resulting sequence of numbers will be statistically indistinguishable from a sequence drawn at random (but without replacement) from the set 1, 2, …, m - 1. But there is nothing random at all about the algorithm, apart from the choice of the initial value X_0. Once that value is chosen, the remaining numbers in the sequence follow deterministically. This has implications in cryptanalysis.

If an opponent knows that the linear congruential algorithm is being used and if the parameters are known (e.g. $a = 7^5$, $c = 0$, $m = 2^{31}$- 1), then once a single number is discovered, all subsequent numbers are known. Even if the opponent knows only that a linear congruential algorithm is being used, knowledge of a small part of the sequence is sufficient to determine the parameters of the algorithm.

Suppose that the opponent is able to capture or determine values for X_0, X_1, X_2, and X_3.Then

$$X_1 = (aX_0 + c) \bmod m$$
$$X_2 = (aX_1 + c) \bmod m$$
$$X_3 = (aX_2 + c) \bmod m$$

These equations can be solved for a, c, and m. Thus, although it is nice to be able to use a good PRNG, it is desirable to make the actual sequence used non-reproducible, so that knowledge of part of the sequence on the part of an opponent is insufficient to determine future elements of the sequence. This goal can be achieved in a number of ways. For example, using an internal system clock we can modify the random number stream. One way to use the clock would be to restart the sequence after every numbers using the current clock value (mod m) as the new seed. Another way would be simply to add the current clock value to each random number (mod m).

Blum Blum Shub Generator:

A popular approach for generating secure pseudorandom numbers is known as the Blum, Blum, Shub (BBS) generator, named after its developers. It has perhaps the strongest public proof of its cryptographic strength of any purpose-built algorithm. The procedure is as follows. First, choose two large prime numbers, so that both have a remainder of 3 when divided by 4.

That is, $p \equiv q \equiv 3 \pmod 4$.

This notation, simply means that (p mod 4) = (q mod 4) = 3. For example, the prime numbers 7 and 11 satisfy $7 \equiv 11 \equiv 3 \pmod 4$. Let $n = p \times q$. Next, choose a random number s, such that s is relatively prime to n; this is equivalent to saying that neither p nor q is a factor of s.

Then the BBS generator produces a sequence of bits B_i, according to the following algorithm:

$X_0 = s^2 \bmod n$

For $i = 1$ **to** ∞

$X_i = (X_{i-1})^2 \bmod n$

$B_i = X_i \bmod 2$

Thus, the least significant bit is taken at each iteration. Table 8.2, shows an example of BBS operation. Here, $n = 192649 = 383 \times 503$, and the seed $s = 101355$.

i	X_i	B_i
0	20749	
1	143135	1
2	177671	1
3	97048	0
4	89992	0
5	174051	1
6	80649	1
7	45663	1
8	69442	0
9	186894	0
10	177046	0
11	137922	0
12	123175	1
13	8630	0
14	114386	0
15	14863	1
16	133015	1
17	106065	1
18	45870	0
19	137171	1
20	48060	0

Table 8.2: Operation of BBS generator

8.5 SUMMARY

Basic principle of stream cipher and widely used stream ciphers RC4, RC5 are explained in the first three sections. Random numbers are key ingredients for the function of stream cipher. Principles of true random number generators (TRNG) and its inconveniences are discussed in section 8.4. Also an useful alternatives namely pseudo random generator (PRNG) is explained here.

8.6 KEYWORDS

PRNG, TRNG, RC4, Stream cipher, Linear congruential generator, Blum Blum Shub generation of random numbers.

8.7 QUESTIONS

1. Explain stream cipher.
2. RC4 is widely used in many applications – why?
3. Write the algorithm for RC4.
4. Give an overview of RC5.
5. Differentiate TRNG and PRNG.
6. Explain linear congruential generator.
7. Discuss Blum Blum Shub method of random number generation.

8.8 REFERENCES

1. Atul Kahate, Cryptography and Network Security, Tata MCGrawHill
2. Behrouz A Forouzan, Cryptography and Network security, McGraw Hill
3. William Stallings, Cryptography and Network Security, Pearson

MODULE 3

PUBLIC KEY CRYPTOGRAPHY

UNIT-9: ESSENTIAL MATHEMATICS FOR ASYMMETRIC ENCRYPTION

Structure

9.0 OBJECTIVES

When you have understood the concepts covered in this unit you will

- ✓ Know important results in number theory
- ✓ Understand prime factoring of integers
- ✓ Come to know Fermat's theorem
- ✓ Be familiar with Euler's theorem
- ✓ Be able to appreciate the utility of approximate algorithm for primality testing
- ✓ Understand the concept of discrete logarithms

9.1 FERMAT'S THEOREM

We begin this unit with the result on factoring natural numbers into product of prime powers.

Result 1 (Fundamental Theorem of Arithmetic): Any integer a>1 can be factored as product of powers of primes i.e. $a=p_1^{a1} \times p_2^{a2} \times \ldots \times p_k^{ak}$ where p_1 , p_2 , p_k are prime numbers and a_1, a_2, …a_k are integers.

Example 1: $91=7 \times 13$, $3600=2^4 \times 3^2 \times 5^2$, $11011 = 7 \times 11^2 \times 13$

It is useful to write this in another way.

$a= \Pi\, p^{ap}$ where $a_p \geq 0$ and p, a prime number

An integer is then specified by all non zero exponents. Thus,

$91 = 2^0 \times 3^0 \times 5^0 \times 7^1 \times 11^0 \times 13^1 = (a_7=1, a_{13}=1)$

$3600 = 2^4 \times 3^2 \times 5^2 = (a_2=4, a_3=2, a_5=2)$

$11011= 2^0 \times 3^0 \times 5^0 \times 7^1 \times 11^2 \times 13^1 = (a_7=1, a_{11}=2, a_{13}=1)$

$12 = 2^2 \times 3^1$ $(a_2=2, a_3=1)$

$18= 2^1 \times 3^2$ $(a_2=1, a_3=2)$

Multiplication of numbers

As integers are expressed as product of prime powers, the multiplication of integers is thus addition of prime powers.

Example 2: $12 \times 18 = 216 = 2^3 \times 3^3$

Note that $12 = 2^2 \times 3^1$ $(a_2=2, a_3=1)$,

$18= 2^1 \times 3^2$ $(a_2=1, a_3=2)$ and

216 factored as $2^3 \times 3^3$ $(a_2=3, a_3=3)$

Divisibility

If a and b are two integers then a | b if and only if $a_p \leq b_p$ for all p where $a = \Pi\, p^{ap}$ and $b= \Pi p^{bp}$.

Example 3: Let a=12 and b=36. Now a | b

Also $12 = 2^2 \times 3^1$ ($a_2=2$, $a_3=1$) and $36 = 2^2 \times 3^2$ ($b_2=2$, $b_3=2$)

Note that $a_p \leq b_p$ where p=2, 3

That is $a_2=2 \leq b_2=2$ and $a_3=1 \leq b_3=2$

Example 4: Let a=6, b=54. Now a | b

Also, $6 = 2^1 \times 3^1$ ($a_2=1$, $a_3=1$) and $54 = 2^1 \times 3^3$ ($b_2=1$, $b_3=3$)

Note that $a_p \leq b_p$ where p=2, 3

Example 5: a=4, b=20, a= 2^2 ($a_2=2$), b= $2^2 \times 5$ ($b_2=2$, $b_5=1$)

Note that $a_p \leq b_p$ where p=2, 5

GCD of two numbers

When numbers are expressed as product of prime powers it is easy to find GCD of the numbers.

GCD = Πp^{cp} where $c_p = \min(a_p, b_p)$ for all p

Example 6: GCD(300, 18) = 6

$300 = 2^2 \times 3^1 \times 5^2$ and $18 = 2^1 \times 3^2$ and

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

Example 7: GCD(50, 24) = 2

$50 = 2^1 \times 5^2$ and $24 = 2^3 \times 3^1$ and $2^1 \times 3^0 \times 5^0 = 2$

Finding prime factors is not easy and hence this method of finding GCD is hardly useful.

Fermat's theorem: If p is prime and a is positive integer not divisible by p then $a^{p-1} \equiv 1 \bmod p$.

Example 8: a=7, p=19

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

$7^4 \equiv 11^2 \pmod{19} = 121 \pmod{19} = 7$

$7^8 \equiv 7^2 \pmod{19} = 11$

$7^{16} \equiv 11^2 \pmod{19} = 121 \pmod{19} = 7$

$7^{18} \equiv 11 \times 7 \pmod{19} = 77 \pmod{19} = 1$

Example 9: a= 3 and p=5

$3^2 \equiv 9 \pmod 5 = 4$,

$3^4 \equiv 4^2 \pmod 5 = 16 \pmod 5 = 1$

Another form of Fermat's theorem:

If a is any integer and p a prime number then, $a^p \equiv a \pmod p$.

Note that a and p need not be relatively prime.

Example 10: a= 3 and p=5

$3^2 \equiv 9 \bmod 5 = 4$,

$3^4 \equiv 4^2 \bmod 5 = 16 \bmod 5 = 1$ and

$3^5 \equiv 1 \times 3 \bmod 5 = 3 \bmod 5$.

In this example a and p are relatively prime.

Example 11: a= 10 and p=5 (a and p are not relatively prime)

$10^5 = 100000 \bmod 5 = 0$ and a mod p = 10 mod 5 = 0

Example 12: a= 6 and p=3 (a and p are not relatively prime)

$6^3 = 216 \bmod 3 = 0$ and 6 mod 3=0

9.2 EULER'S THEOREM

Another important result of number theory that is useful in understanding public key systems is Euler's theorem. Before going into details of statement and examples, we introduce totient function.

Euler's Totient function

Given an integer n > 1, the number of integers less than n and prime to n is called Euler's totient function denoted by $\phi(n)$. $\phi(1)$ is taken to be 1.

Example 13: n=37 (prime number). As n is prime all numbers less than n (36 in number) are prime to n thus $\phi(37) = 36$. For any prime p, $\phi(p) = p-1$

Example 14: $\phi(35) = 24$. The numbers less than 35 and prime to 35 are 1, 2, 3, 4, 6, 8, 9, 11, 12, 13, 16, 17, 18, 19, 22, 23, 24, 26, 27, 29, 31, 32, 33, 34 and there are 24 numbers in this list.

Result: If n= p × q (p, q are primes) then $\phi(n) = \phi(p) \times \phi(q) = (p-1) \times (q-1)$

Example 15: $\phi(21) = 12$ [numbers less than 21 and prime to 21 are 1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20

Alternatively, $21 = 3 \times 7$ and $\phi(3) = 2$, $\phi(7) = 6$. Thus $\phi(21) = \phi(3) \times \phi(7) = 2 \times 6 = 12$.

Now that you have enough background knowledge, we are in a position to state Euler's theorem.

Euler's theorem: For every a and n that are relatively prime, $a^{\phi(n)} \equiv 1 \bmod n$.

Example 16: Let a=3, n=10; $\phi(10) = 4$, $3^4 = 81 \equiv 1 \bmod 10$

Example 17: Let a=2, n=11; $\phi(11) = 10$, $a^{\phi(n)} = 2^{10} = 1024 \equiv 1 \bmod 11 = 1 \bmod n$

Alternative form of this theorem is as follows: For any two integers a, n we have $a^{\phi(n)+1} \equiv a \bmod n$. (Note that a and n need not be relatively prime)

Example 18: Let a=3, n=10; $\phi(10) = 4$, $a^{\phi(n)+1} = 3^5 = 243 \equiv 3 \bmod 10 = a \bmod n$
In this example a and n are relatively prime.

Example 19: a=3, n=6; $\phi(6)=2$, $a^{\phi(n)+1} = 3^3 = 27 \equiv 3 \bmod 6 = a \bmod n$
In this example a and n are not relatively prime.

9.3 PRIME NUMBERS

For many cryptographic algorithms we need large prime numbers. So, we generate a random number and then test if it is prime. Deterministic algorithm for prime number test is complex (check if any number in the range of 2 to $\sqrt{n}$ divides n; if so it is not prime). Miller (1975) and Rabin (1980) developed efficient algorithm that almost certainly determines if n is prime. More results on number theory are needed to understand Miller Rabin method.

Before discussing the algorithm that tests a given number to be prime, we state some results on prime numbers supported by examples.

Results on prime numbers

1. Any odd positive number n >2 can be expressed as $n-1=2^k q$ where k>0 and q is odd

Example 20: Let n=23 (prime), n-1 = 22 = 2 x 11 (k=1 and q=11)

Example 21: Let n=35 9non prime), n-1 = 34 = 2 x 17 (k=1 and q=17)

2. Property 1 of prime number: If p is prime and a is any positive integer then a^2 mod p =1 iff either a mod p = 1 or a mod p = -1(p-1)

Example 22: Let a=4, p=3; a^2 mod p =16 mod 3 =1;

Also a mod p =1

Example 23: Let a=4, p=2; a^2 mod p =16 mod 2 = 0;

Neither 4 mod 2 = 1 nor 4 mod 2 = -1

Example 24: Let a=6, p=7; a^2 mod p = 36 mod 7 = 1

Also a mod p = -1

3. Property 2 of prime number: Let p be prime > 2. Recall p-1 = 2^k q where k >0, q odd. Let a be any integer such that 1 < a < p-1 then one of the following conditions is true.
    i. a^q mod p =1
    ii. One of a^q, a^{2q}, … $a^{(2\wedge(k-1)*q)}$ is congruent to -1 mod p. That is there is some number j ɜ 1 < j < k so that a^j = -1 mod p.

Example 25: Suppose p=5, a=3; p-1= 4 and k = 2, q=1. a^q mod p = 3 mod 5 is not 1

But a^{2q} mod p = 9 mod 5 = 4 = -1 mod 5

Example 26: Suppose p= 7, a=4; p-1=6 and k=1, q=3. a^q mod p = 64 mod 7 =1

Miller Robin Algorithm

We can use the preceding property to devise a test for primality. The property 2 implies that if n is prime then first number in list of residues (modulo) (a^q, a^{2q}, … $a^{(2\wedge(k-1))*q}$) mod n is 1 or some element in the list is -1. On the other hand if the condition is met it does not imply that the number n is prime.

Example 27: Let n=2047. n is not prime since n = 23 × 89.

Now n-1 = 2046 = 2 × 1023; that is k=1 and q=1023. For a=2 we see that $a^q = 2^{1023}$ = 1 mod 2047 = 1 mod n

The algorithm for testing primality of n is given here.

TEST (n)

1. Find integers k, q with k>0, q odd so that n-1=2^k × q
2. Select a random integer a ɜ 1 < a < n-1.
3. If a^q mod n =1 then return “inconclusive”.
4. For j= 0 to k-1 do

5. If $a^{(2\wedge j)*q}$ is -1 mod n then return "inconclusive"
   End for
6. Return "composite"

The algorithm above returns "composite" implies that n is definitely not a prime and if it returns "inconclusive" then it implies that n may or may not be prime. The algorithm is executed with several random a's. If the result is "inconclusive" for all a's then n is prime with high probability.

Example 28: n=29 (prime). The algorithm returns "inconclusive" for all a's from 1 to 28. This is consistent with n being prime.

n=221 = 13 x 17, a composite number

n-1 = 220 = 2^2 x 55 ; thus q=2, k=55

For a= 21 you get 'inconclusive" since $21^{55\wedge 2}$ is -1 mod 22.

This means that n may be prime (according to the algorithm) but we know that n is definitely composite. You can verify that only for few a's (21, 47, 174, 200) the algorithm returns "inconclusive". For a = 5 the algorithm returns "composite". Thus running the algorithm for several a's will correctly determine composite number. If it returns "inconclusive" for several a's it is highly likely that it is prime and we assume n is prime.

Deterministic algorithm for testing primality

Prior to 2002, there was efficient method known to check primality of very large numbers. All of the algorithms in use, including the most popular (Miller-Rabin), produced a probabilistic result. In 2002, Agrawal, Kayal, and Saxena developed a relatively simple deterministic algorithm that efficiently determines whether a given large number is a prime number. The algorithm, known as the AKS algorithm, does not appear to be as efficient as the Miller-Rabin algorithm. Thus, it is not considered superior to this older, probabilistic technique.

Distribution of primes

It is worth noting how many numbers are likely to be rejected before a prime number is found using the Miller-Rabin test, or any other test for primality. A result from number theory, known as the prime number theorem, states that the primes near n are spaced on the average one every (ln n) integers. Thus, on average, one would have to test on the order of ln(n) integers before a

prime is found. Because all even integers can be immediately rejected, the correct figure is 0.5* ln(n). For example, if a prime on the order of magnitude of 2^{200} were sought, then about 0.5 $\ln(2^{200})$ = 69 trials would be needed to find a prime. However, this figure is just an average. In some places along the number line, primes are closely packed, and in other places there are large gaps.

9.4 DISCRETE LOGARITHMS

The Discrete logarithm is fundamental to many public key encoding systems including Diffel Hellman key exchange algorithms and digital signatures.

Recall Euler totient function $\phi(n)$ and the Euler's theorem, which states that $a^{\phi(n)} \equiv 1 \bmod n$ if a, n are relatively prime. Now consider more general expression $a^m = 1 \bmod n$. If a and n are relatively prime there is at least one integer ($\phi(n)$) that satisfies the general expression.

The least positive integer m which satisfies the equation $a^m = 1 \bmod n$ is called (i) order of a mod n (ii) exponent to which *a* belongs to mod n (iii) length of the period generated by *a*

Example 29:

Consider a=7, n= 19

$7^1 = 7 \bmod 19$

$7^2 = 11 \bmod 19$

$7^3 = 1 \bmod 19$

$7^4 = 7 \bmod 19$ …

The sequence repeats. The period is 3. This is nothing but the smallest integer m for which $a^m = 1 \bmod n$ (in this example $7^3 = 1 \bmod 19$).

For a = 2, you can check that ($a, a^2, a^3, \ldots a^{\phi(19)}$) mod 19 are 2, 4, 8, 16, 13, 7, 14, 9, 18, 17, 15, 11, 3, 6, 12, 5, 10, 1

All numbers 1 to 18 (the value of $\phi(19)$) have appeared. The period is 18, full period. Similarly you can verify that 3, 10, 13, 14, 15 all have full period.

Primitive root

In general the highest possible exponent to whom a number can belong (mod n) is $\phi(n)$. If a number is of this order it is referred to as a primitive root of n. Alternatively if a is a primitive root of n then a, a^2, a^3, … $a^{\phi(n)}$ are all distinct (mod n) and are all prime to n. In particular for a prime number *p*, if *a* is a primitive root of *p* then a, a^2, a^3, … a^{p-1} are distinct (mod p) and prime to p. from the preceding example we see that, primitive roots of 19 are 2, 3, 10, 13, 14 and 15.

Table 9.1 shows all the powers of a, modulo 19 for all positive a < 19. The length of the sequence for each base value is indicated by shading. Note the following:

1. All sequences end in 1. This is consistent with the reasoning of the preceding few paragraphs.
2. The length of a sequence divides $\phi(19)$ =18. That is, an integral number of sequences occur in each row of the table. The length of the sequence for a = 1 is 1. For a = 2 and 3 it is 18. For a = 4 it is 9 and so on. Note that all these lengths divide $\phi(n)$ (=18).

a	**a^2**	**a^3**	**a^4**	**a^5**	**a^6**	**a^7**	**a^8**	**a^9**	**a^{10}**	**a^{11}**	**a^{12}**	**a^{13}**	**a^{14}**	**a^{15}**	**a^{16}**	**a^{17}**	**a^{18}**
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	**4**	**8**	**16**	**13**	**17**	**14**	**9**	**18**	**17**	**15**	**11**	**3**	**6**	**12**	**5**	**10**	**1**
3	**9**	**8**	**5**	**15**	**7**	**2**	**6**	**18**	**16**	**10**	**11**	**14**	**4**	**12**	**17**	**13**	**1**
4	**16**	**7**	**9**	**7**	**11**	**6**	**5**	**1**	4	16	7	9	17	11	6	5	1
5	**6**	**11**	**17**	**9**	**7**	**6**	**4**	**1**	5	6	11	17	9	7	11	4	1
6	**17**	**7**	**4**	**5**	**11**	**9**	**16**	**1**	6	17	7	4	5	11	9	16	1
7	**11**	**1**	7	11	1	7	11	1	7	11	1	7	11	1	7	11	1
8	**7**	**18**	**11**	**12**	**1**	8	7	18	11	12	1	8	7	18	11	12	1
9	**5**	**7**	**6**	**16**	**11**	**4**	**17**	**1**	9	5	7	6	16	11	4	17	1
10	**5**	**12**	**6**	**3**	**11**	**15**	**17**	**18**	**19**	**14**	**7**	**13**	**16**	**8**	**4**	**2**	**1**
11	**7**	**1**	11	7	1	11	7	1	11	7	1	11	7	1	11	7	1
12	**11**	**18**	**7**	**8**	**1**	**12**	**11**	**18**	**7**	**8**	**1**	**12**	**11**	**18**	**7**	**8**	**1**
13	**17**	**12**	**4**	**14**	**11**	**10**	**16**	**18**	**6**	**2**	**7**	**15**	**5**	**8**	**9**	**3**	**1**
14	**6**	**8**	**17**	**10**	**7**	**3**	**4**	**18**	**5**	**13**	**11**	**2**	**9**	**12**	**16**	**15**	**1**
15	**16**	**12**	**9**	**2**	**11**	**13**	**5**	**18**	**4**	**3**	**7**	**10**	**17**	**8**	**6**	**14**	**1**
16	**9**	**11**	**5**	**4**	**7**	**17**	**6**	**1**	16	9	11	5	4	7	17	6	1
17	**4**	**11**	**16**	**6**	**7**	**5**	**9**	**1**	17	4	11	16	6	7	5	9	1

18	**1**	18	1	18	15	18	1	18	1	18	1	18	1	18	1	18	1

3. Some of the sequences are of length 18. In this case, it is said that the base integer a generates (via powers) the set of nonzero integers modulo 19. Each such integer is called a primitive root of the modulus 19. Some primitive roots of 19 are 2, 3, 10.

Table 9.1: Powers of integers modulo 19

More generally, we can say that the highest possible exponent to whom a number can belong (mod n) is $\phi(n)$. If a number is of this order, it is referred to as a primitive root of n. The importance of this notion is that if a is a primitive root of n, then its powers a, a^2, a^3, … $a^{\phi(n)}$ are distinct (mod n) and are all relatively prime to n. In particular, for a prime number p, if a is a primitive root of p, then a, a^2, a^3, … a^{p-1} are distinct (mod p). For the prime number 19, its primitive roots are 2, 3, 10, 13, 14, and 15.

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

Logarithms for modular arithmetic

We review some properties of ordinary logarithms here.

$\log_x(1) = 0$, $\log_x(x) = 1$

$\log_x(yz) = \log_x(y) * \log_x(z)$

$\log_x(y/z) = \log_x(y) - \log_x(z)$

$\log_x(r^a) = a * \log_x(r)$

Consider the primitive root a for some prime number p. We know that powers of a from 1 to *p*-1 are distinct and are integers 1 to p-1 in some order. We also know that any integer b satisfies $b \equiv r \bmod p$ for some r, where $0 \leq r \leq p-1$ by definition of modular arithmetic.

Let *a* be a primitive root of *p*. For the integer b we can find a unique exponent i such that $b \equiv a^i \bmod p$, where $0 \leq i \leq p-1$. The same equation is written in terms of logarithm as $\text{dlog}_{a,p}(b) = i$. The exponent i is called the discrete logarithm of the number b for the base a mod p.

All the properties of ordinary logarithms are satisfied by discrete logarithms.

Example 30: Let p=19. We know that 2 is a primitive root of 19.

Consider b = 52. 52 mod 19 = 14 = 2^7 mod 19

Thus $\text{dlog}_{2,19}(52) = 7$

For 19, 3 is another primitive root

Consider b=52 and 52 mod 19 = 14 = 3^{13} mod 19

Thus $\text{dlog}_{3,19}(52)=13$

Discrete logarithms can also be defined for non prime bases. All we need is a primitive root.

Consider n=9. $\phi(9) = 6$ and a =2 is a primitive root since (2^1, 2^2, … 2^6)mod 9 are distinct and prime to 9. These are nothing but (2, 4, 8, 7, 5, 1)

The logarithm table is given here

No.	2	4	8	7	5	1
$\text{Dlog}_{2,9}$	1	2	3	4	5	6/0

Unlike prime number we don't have logarithm for all numbers. For example, discrete logarithm for number 3, 6 are not defined.

Note that the properties of ordinary logarithms are true in case of discrete logarithms too. We now show some of the properties of discrete logarithms.

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

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

Now consider,

$$x = a^{d\log_{a,p}(x)} \bmod p \qquad y = a^{d\log_{a,p}(y)} \bmod p$$

$$xy = a^{d\log_{a,p}(xy)} \bmod p$$

Using the rules of modular multiplication,

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

$$a^{d\log_{a,p}(xy)} \bmod p = [(a^{d\log_{a,p}(x)} \bmod p)(a^{d\log_{a,p}(y)} \bmod p)] \bmod p$$

$$= a^{d\log_{a,p}(x)+d\log_{a,p}(y)} \bmod p$$

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

$$a^{\phi(n)} \equiv 1 (\text{mod } n)$$

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

$$a^z \equiv a^q (\text{mod } n), \text{ if } z \equiv q \bmod \phi(n)$$

Applying this to the foregoing equality, we have

$$d\log_{a,p}(xy) \equiv [d\log_{a,p}(x) + d\log_{a,p}(y)] (\text{mod } \phi(p))$$

And generalizing,

$$d\log_{a,p}(y^r) \equiv [r * d\log_{a,p}(y)] \bmod \phi(p)$$

This demonstrates the analogy between true logarithms and discrete logarithms.

Calculation of discrete logarithms

Consider the equation $y=g^x$ mod p. Given g, x and p it is straight forward to calculate y. We can multiply *g* with itself *x* times or find modulo at in between steps and find *y*. However given *y, g, p* it is difficult to find *x* (dlog). This is as difficult as factoring a large number into product of primes.

9.5 SUMMARY

In this unit we discussed important results on number theory, such as prime factoring, Fermat's theorem, a useful form of Fermat's theorem, Euler totient function, Euler's theorem, primitive root and discrete logarithms. Sample illustrations are provided to make concepts clear. A good understanding of these topics helps in following public key cryptography discussions.

9.6 KEYWORDS

Prime factoring, Fermat's theorem, Euler totient function, Euler's theorem, primitive root, discrete logarithm.

9.7 QUESTIONS

1. State and illustrate prime factoring of a natural numbers.

2. State Fermat's theorem and provide examples.
3. What is alternative form of Fermat's theorem? Support your statement with examples.
4. Define Euler totient function and illustrate.
5. Write the two forms of Euler's theorem and illustrate both.
6. Define primitive root. Give examples.
7. Define logarithm on modular arithmetic. Provide examples for at least two prime numbers and primitive roots for each.
8. Show that discrete logarithms satisfy many relations as ordinary logarithms.

9.8 REFERENCES

1. Atul Kahate, Cryptography and Network Security, Tata MCGrawHill
2. Behrouz A Forouzan, Cryptography and Network security, McGraw Hill
3. Charlie Kauffman, Radia Perlman, Mike Spciner, Network Security, Pearson Education
4. William Stallings, Cryptography and Network Security, Pearson

UNIT -10: PRINCIPLES OF PUBLIC KEY SYSTEMS

Structure

10.0 OBJECTIVES

A thorough study of the topics in this unit will make clear to the reader the following:

- ✓ Misconceptions of public key system
- ✓ Terminologies used in asymmetric encryption
- ✓ Differences between symmetric and asymmetric encryption
- ✓ Function of public key system
- ✓ Application of public key systems
- ✓ Requirements of public key systems
- ✓ Cryptanalysis of public key encoding

10.1 BRIEF HISTORY

The development of public-key cryptography is regarded as the best and the only true revolution in the entire history of cryptography. Virtually all cryptographic systems are based on the elementary tools of substitution and permutation since the infant stage of cryptography. With the development of the rotor encryption/decryption machine a major advance in symmetric cryptography occurred.

The electromechanical rotor enabled the development of complex cipher systems. More complex systems were devised with the availability of computers. The most prominent of which was the Lucifer effort at IBM that culminated in the Data Encryption Standard (DES). But both rotor machines and DES, although representing significant advances, still relied on the basic tools of substitution and permutation.

Public-key algorithms are based on mathematical functions rather than on substitution and permutation. In contrast to symmetric encryption, which uses only one key, public-key cryptography is asymmetric, which involves the use of two separate keys, The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication.

In the mid-1970s, Whitefield Diffie and his professor Martin Hellman at the Stanford University started thinking about the problem of key exchange. After some research they came up with the idea of asymmetric key cryptography. Diffie and Hellman can be regarded as the fathers of the asymmetric key cryptography.

It is believed that, in 1960s, James Ellis of the British Communications Electronic Security Group (CSEG) proposed the idea of asymmetric key cryptography. However, he could not device a practical algorithm based on his ideas. He then met with Clifford Cocks who joined the CSEG in 1973. After a short discussion, Cocks could come up with a practical working algorithm. However, since CSEG was a secret agency, these findings were never published, therefore these people never got the credit that they deserved.

Based on the theoretical framework of Diffie and Hellman, in 1977, Ron Rivest, Adi Shamir and Len Adleman at MIT developed the first major asymmetric key cryptography system and published their results in 1978. This method is called RSA algorithm. The name RSA comes from the first letters of the surnames of the three researchers.

Misconceptions

Before proceeding, we should mention several common misconceptions concerning public-key encryption. One such misconception is that public-key encryption is more secure from cryptanalysis than the symmetric encryption. In fact, the security of any encryption scheme depends on the length of the key and the computational work involved in breaking a cipher.

There is nothing in principle about either symmetric or public-key encryption that makes one superior to another from the point of view of resisting cryptanalysis.

A second misconception is that public-key encryption is a general-purpose technique that has made symmetric encryption obsolete. On the contrary, because of the computational overhead of current public-key encryption schemes, there seems no foreseeable likelihood that symmetric encryption will be abandoned. As one of the inventors of public-key encryption has put it, "the restriction of public-key cryptography to key management and signature applications is almost universally accepted."

Finally, there is a feeling that key distribution is trivial when using public-key encryption, compared to the rather cumbersome handshaking involved with key distribution centers for symmetric encryption. In fact, some form of protocol is needed, generally involving a central agent, and the procedures involved are neither simpler nor any more efficient than those required for symmetric encryption.

10.2 OVERVIEW OF PUBLIC KEY SYSTEM

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

1. It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key. In addition, some algorithms, such as RSA, also exhibit the following characteristic.
2. Either of the two related keys can be used for encryption, with the other used for decryption.

A **public-key encryption** scheme has six ingredients as shown in figure 10.1.

1. **Plaintext:** This is the readable message or data that is fed into the algorithm as input.
2. **Encryption algorithm:** The encryption algorithm performs various transformations on the plaintext. This encrypts plain text using public key of receiver (as in figure 10.1) or using private key of the sender (as in figure 10.2)
3. **Public and private keys:** This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations

performed by the algorithm depend on the public or private key that is provided as input. In figure 10.1, encryption is done using public key and decryption using private key. Whereas in figure 10.2, encryption is done using private key and decryption using public key.

4. **Cipher text:** This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different cipher texts.
5. **Decryption algorithm:** This algorithm accepts the cipher text and the matching key and produces the original plaintext. In figure 10.1, decryption algorithm uses private key, whereas in figure 10.2, decryption is done using public key.

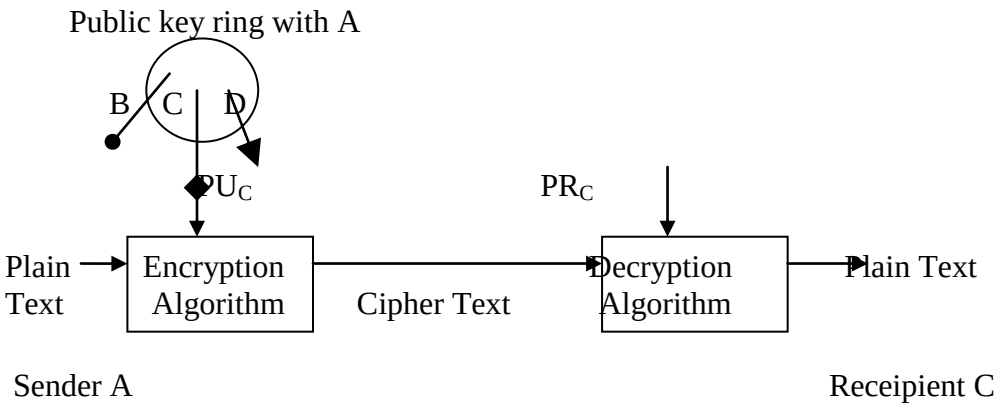


Figure 10.1: Encryption with public key

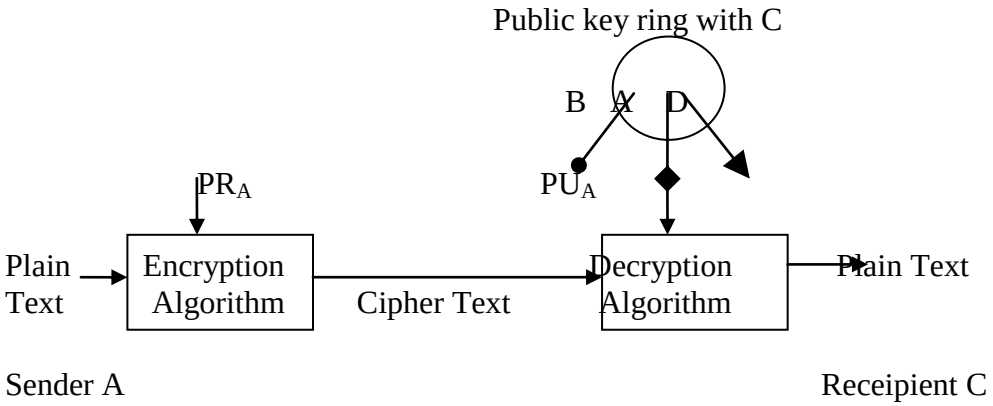


Figure 10.2: Encryption with private key

The essential steps are the following:

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.
2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private. As figures 10.1 and 10.2 suggest, each user maintains a collection of public keys obtained from others.
3. If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key (figure 10.1). Or Bob uses his private key to encrypt the plain text (figure 10.2).
4. When Alice receives the message, she decrypts it using her private key (figure 10.1). No other recipient can decrypt the message because only Alice knows Alice's private key. Or Alice uses public key of Bob (figure 10.2).

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

10.3 CONVENTIONAL VERSUS PUBLIC KEY SYSTEMS

Table 10.1 summarizes the important characteristics of symmetric and asymmetric cryptosystems. To discriminate between the two, we refer to the key used in symmetric encryption as a secret key. The two keys used for asymmetric encryption are referred to as the public key and the private key. Invariably, the private key is kept secret, but it is referred to as a private key rather than a secret key to avoid confusion with symmetric encryption.

Public key encryption can offer confidentiality or sender authentication or both, depending on keys used for encryption/decryption. Figures 10.3, 10.4 and 10.5 show all the three forms of encryption.

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

Table 10.1: Conventional and public key encryption

With the message *X* and the encryption key *PUb* as input, A forms the cipher text *Y* = E(*PUb*, *X*). The intended receiver, in possession of the matching private key, is able to invert the transformation as *X* = D(*PRb*, *Y*). An adversary, observing *Y* and having access to *PUb*, but not having access to *PRb* or *X*, must attempt to recover *X* and/or *PRb*. It is assumed that the adversary does have knowledge of the encryption (E) and decryption (D) algorithms. If the adversary is interested only in this particular message, then the focus of effort is to recover *X* by generating a plaintext estimate *X* ˆ. Often, however, the adversary is interested in being able to read future messages as well, in which case an attempt is made to recover *PRb* by generating an estimate *PRb* ˆ.

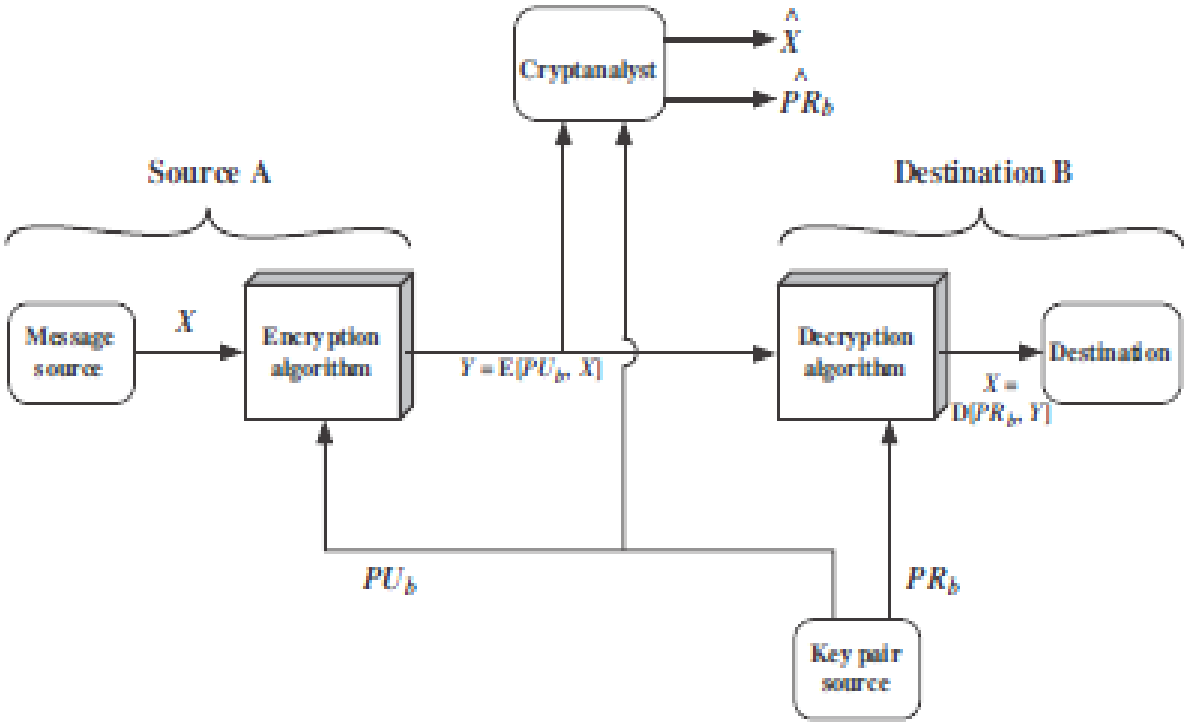


Figure 10.3: Public key encryption for confidentiality

The scheme illustrated in Figure 10.3 provides confidentiality, Figures 10.2 and 10.4 show the use of public-key encryption to provide authentication. Here cipher text is generated using $Y = E(PRa, X)$ and plain is recovered using $X = D(PUa, Y)$

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

In the preceding scheme, the entire message is encrypted, which, although validating both author and contents, requires a great deal of storage. Each document must be kept in plaintext to be used for practical purposes. A copy also must be stored in cipher text so that the origin and contents can be verified in case of a dispute. A more efficient way of achieving the same results is to encrypt a small block of bits that is a function of the document. Such a block, called an authenticator, must have the property that it is infeasible to change the document without

changing the authenticator. If the authenticator is encrypted with the sender's private key, it serves as a signature that verifies origin, content, and sequencing.

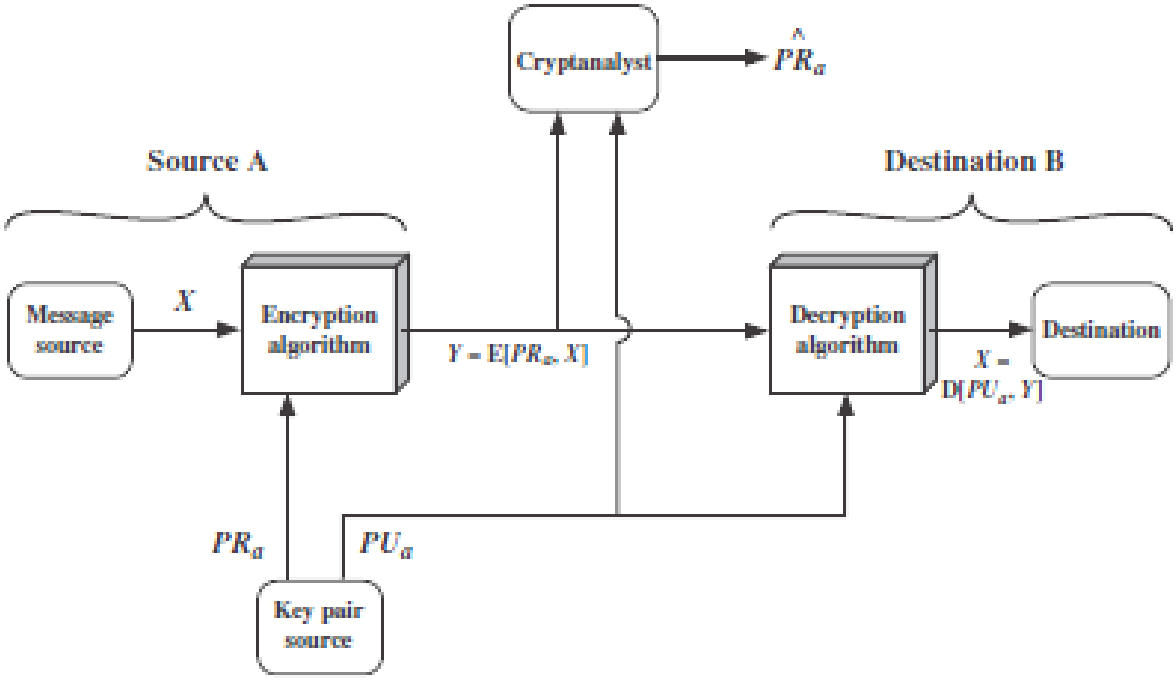


Figure 10.4: Public key encryption for authentication.

It is, however, possible to provide both the authentication function and confidentiality by a double use of the public-key scheme as in figure 10.5. The cipher text is produced by double encryption as Z = E(*PUb*, E(*PRa*, *X*)) and plain text is recovered using double decryption in the reverse order of encryption as X = D(*PUa*, D(*PRb*, *Z*)).

In this case, we begin as before by encrypting a message, using the sender's private key. This provides the digital signature. Next, we encrypt again, using the receiver's public key. The final cipher text can be decrypted only by the intended receiver, who alone has the matching private key. Thus, confidentiality is provided. The disadvantage of this approach is that the public-key algorithm, which is complex, must be exercised four times rather than two in each communication.

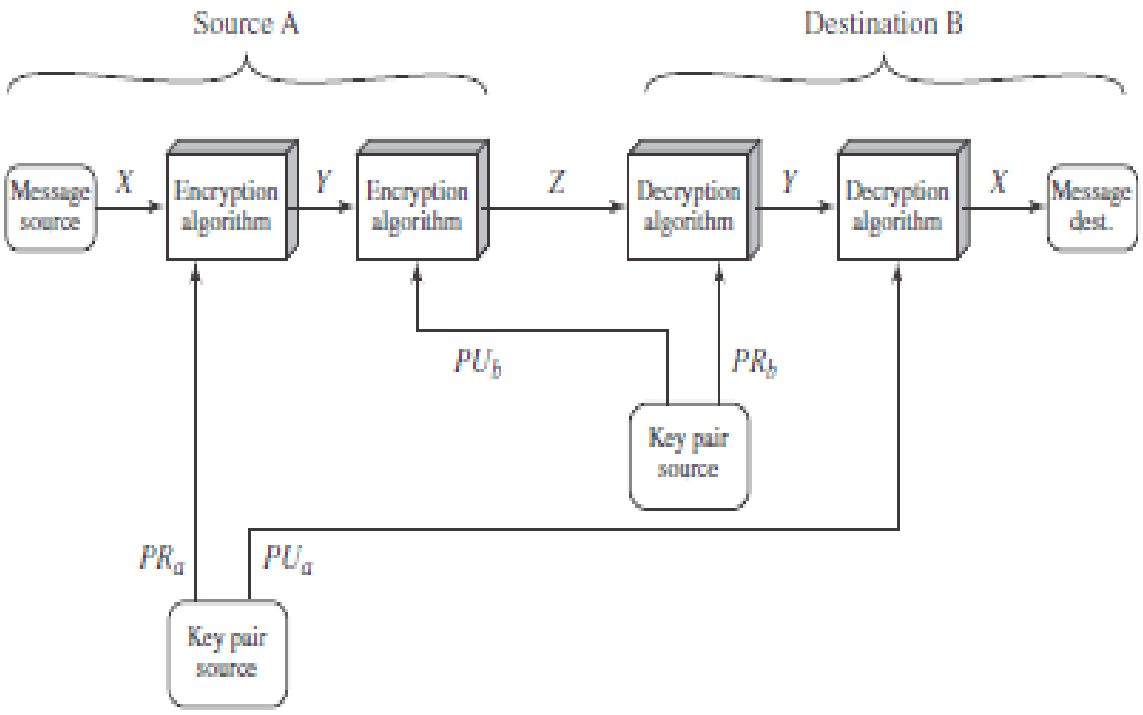


Figure 10.5: Public key encryption for confidentiality and authentication.

Public key systems are useful for purposes other than encryption/decryption. We list here some important applications:

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

Some algorithms are suitable for all three applications, whereas others can be used only for one or two of these applications. Table 10.2 indicates the applications supported by the algorithms discussed.

Algorithm	Encryption/decryption	Digital signature	Key exchange
RSA	Yes	Yes	Yes
Elliptic curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

Table 10.2: Applications for public-key cryptosystems

10.4 REQUIREMENTS OF PUBLIC KEY CRYPTOGRAPHY

The cryptosystem illustrated in Figures 10.3 through 10.5 depends on a cryptographic algorithm based on two related keys. Diffie and Hellman postulated this system without demonstrating that such algorithms exist. However, they did lay out the conditions that such algorithms must fulfill.

1. It is computationally easy for a party B to generate a pair (public key *PUb*, private key *PRb*).
2. It is computationally easy for a sender A, knowing the public key and the message to be encrypted, *M*, to generate the corresponding cipher text: $C = E(PUb, M)$
3. It is computationally easy for the receiver B to decrypt the resulting cipher text using the private key to recover the original message: $M = D(PRb, C) = D[PRb, E(PUb, M)]$
4. It is computationally infeasible for an adversary, knowing the public key, *PUb*, to determine the private key, *PRb*.
5. It is computationally infeasible for an adversary, knowing the public key, *PUb*, and a cipher text, *C*, to recover the original message, *M*.

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

6. The two keys can be applied in either order: $M = D[PUb, E(PRb,M)] = D[PRb, E(PUb, M)]$

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

10.5 CRYPTANALYSIS OF PUBLIC KEY SYSTEMS

As with symmetric encryption, a public-key encryption scheme is vulnerable to a brute-force attack. The countermeasure is the same: Use large keys. However, there is a tradeoff to be considered. Public-key systems depend on the use of some sort of invertible mathematical function. The complexity of calculating these functions may not scale linearly with the number of bits in the key but grow more rapidly than that. Thus, the key size must be large enough to make brute-force attack impractical but small enough for practical encryption and decryption. In practice, the key sizes that have been proposed do make brute-force attack impractical but result in encryption/decryption speeds that are too slow for general-purpose use. Instead, as was mentioned earlier, public-key encryption is currently confined to key management and signature applications.

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

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

10.6 SUMMARY

This unit is a gentle introduction to public key cryptography. Beginning from a brief history of the development in section 10.1, overview of the system, characteristics of symmetric and asymmetric encryption, requirements of public key cryptography and possible special attacks are discussed in sections 10.2 to 10.5.

10.7 KEYWORDS

Public key cryptosystems, confidentiality, authentication, public key, private key, digital signature, key exchange.

10.8 QUESTIONS

1. What are the principal elements of public key cryptosystem?
2. Discuss the roles of public and private keys.
3. Mention the applications of asymmetric encryption.
4. Using figures show and explain how systems are useful to protect data, authenticate, sender and data and do both.
5. List the requirements of public key systems.
6. Write about attacks on public key systems.

10.9 REFERENCES

1. Atul Kahate, Cryptography and Network Security, Tata MCGrawHill
2. Behrouz A Forouzan, Cryptography and Network security, McGraw Hill
3. Wade Trappe, Lawrence Washington, Introduction to Cryptography with Coding Theory, Pearson International
4. William Stallings, Cryptography and Network Security, Pearson

UNIT -11: ASSYMETRIC ENCRYPTION-RSA

Structure

11.0 OBJECTIVES

A study of this unit will enable you to

- ✓ Understand the RSA algorithm, a popular public key cryptography
- ✓ Come to know of increasing efficiency of RSA
- ✓ Appreciate the strength of RSA
- ✓ Become familiar with some attacks on RSA

11.1 RSA ALGORITHM

Diffie and Hellamn [1976] introduced an approach for public key systems and challenged researchers to devise algorithms that met the requirements. The challenge was met by algorithm proposed by Ron Rivest, Adi Shamir, Len Adleman (RSA) in 1978 at MIT. RSA is block cipher in which plain text and cipher text are integers between 0 and n-1 for some n. Typical n has 1024 bits ($n < 2^{1024}$) or 309 decimal digits.

Details of the algorithm

RSA makes use of an expression with exponentials. Plaintext is encrypted in blocks, with each block having a binary value less than some number *n*. That is, the block size must be less than or

equal to $\log_2 (n + 1)$. In practice, the block size is i bits, where $2^i < n < 2^{i+1}$. Encryption and decryption are of the following form, for some plaintext block M and cipher text block C.

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

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

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

For now, we focus on the first requirement and consider the other questions later. We need to find a relationship of the form $M^{ed} \bmod n = M$.

The preceding relationship holds if e and d are multiplicative inverses modulo $\varphi(n)$, where $\varphi(n)$ is the Euler totient function. It is shown that for p, q prime, $\varphi(pq) = (p - 1)(q - 1)$. The relationship between e and d can be expressed as $ed \bmod \varphi(n) = 1$ ----------(11.1)
This is equivalent to saying

$$ed \equiv 1 \bmod \phi(n)$$
$$d \equiv e^{-1} \bmod \phi(n)$$

That is, e and d are multiplicative inverses mod $\phi(n)$. Note that, according to the rules of modular arithmetic, this is true only if d (and therefore e) is relatively prime to $\phi(n)$. Equivalently, $\gcd(\phi(n), d) = 1$. The equation (11.1) satisfies the requirement for RSA.

We are now ready to state the RSA scheme. The ingredients are the following:

1. p, q be two prime numbers (private, chosen)
2. n=p*q (public, calculated)
3. Select e, with $\gcd(\phi(n),e)=1$; $1 < e < \phi(n)$ (public, chosen)
4. Calculate $d=e^{-1} \bmod \phi(n)$ (private, calculated)

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

Suppose A wants to send an encrypted message to B, then the sequence of exchanges between the two are given as follows:

Method 1

1. Key generation by B

a. Select p and q that are prime numbers and $p \neq q$.

b. Calculate $n = p * q$.

c. Calculate $\phi(n) = (p-1)(q-1)$.

d. Select integer e that is prime to $\phi(n)$ and less than $\phi(n)$.

e. Calculate $d = e^{-1} \pmod{\phi(n)}$.

f. $PU_B = \{e, n\}$.

g. $PR_B = \{d, n\}$.

2. Encryption by A with B's public key

a. Plain text be $M < n$.

b. Cipher text is $C = M^e \bmod n$

3. Decryption by B with B's private key

a. Cipher text is C.

b. Retrieved plain text is $C^d \bmod n$.

Method 2

1. Key generation by A

a. Select p, q that are prime numbers and $p \neq q$.

b. Calculate $n = p * q$.

c. Calculate $\phi(n) = (p-1)(q-1)$.

d. Select integer e that is prime to $\phi(n)$ and less than $\phi(n)$.

e. Calculate $d = e^{-1} \pmod{\phi(n)}$.

f. $PU_A = \{e, n\}$.

g. PR_A= {d, n}.

2. Encryption by A with A's private key

a. Plaintext be M < n.

b. Cipher text is $C = M^d \bmod n$

3. Decryption by B with A's public key

a. Cipher text is C

b. Retrieved plaintext is $C^e \bmod n$.

11.2 EXAMPLES OF ENCRYPTION/ DECRYPTION

I. Working example

1. Select p=17, q=11
2. Calculate n=p*q=17 x 11= 187
3. Calculate $\phi(n)$ =(p-1)*(q-1)=16 x 10 = 160
4. Select e such that e is relatively prime to 160 and < 160 say e=7
5. Determine d such that $d*e \equiv 1 \bmod 160$ and d<160 (value of d =23 (since 23 x7= 161)

The keys are PU= [7, 187] and PR= [23, 187]. Let plain text be M=88. For encryption we need to calculate $88^7 \bmod 187 = 11$ (cipher) and for decryption we need to calculate $11^{23} \bmod 187 = 88$ (plain text). Thus plain text is successfully recovered during decryption.

II. Example where plain text not recovered

1. Suppose n=187 (7 bit number), e=7 and d(calculated inverse) =23
2. Let M=189 (7 bits)
3. We get $C=189^7 \bmod 187 = 128$ (encryption)
4. $M=128^{23} \bmod 187 = 2$ (decryption)

Note that retrieved M (2) is different from actual M (189). The problem is because M is not < n. In this example, n is a 7 bit number. Any 7 bit number cannot be M. In other words, block size cannot be taken as the number of bits in n.

11.3 COMPUTATIONAL ASPECTS

This section scrutinizes the computational complexity of encryption and decryption using RSA and the key generation.

Encryption/Decryption computation

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

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

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

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

$$x^{16} = x * x * x * x * x * x * x * x * x * x * x * x * x * x * x * x$$

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

More generally, suppose we wish to find the value a^b with a and m positive integers. If we express b as a binary number $b_k\, b_{k-1} \ldots b_0$, then we have

$$b=b(k)*2^k+b(k-1)*2^{(k-1)}+\ldots\ b(1)*2+b(0)*1$$

$$a^b = a^{b(k)*2\wedge k+b(k-1)*2\wedge(k-1)+\ldots\ b(1)*2+b(0)*1} = \prod_{b(i)\neq 0} a^{2\wedge i}$$

$$a^b = a^{\sum 2\wedge i} = \pi a^{(2\wedge i)}$$

$$a^b \bmod n = [\pi a^{(2\wedge i)}] \bmod n = (\pi[a^{(2\wedge i)} \bmod n]) \bmod n$$

Thus the algorithm for finding a^b mod n is as follows:

Algorithm a-power-b-mod n

c =0; f =1

For i= k down to 0 // k is the number of bits-1

c =2*c

f=(f*f) mod n

If b_i = 1

then c =c+ 1

f =(f * a) mod n

End for

Return f

Example:

Consider computation of 3^5 Here a=3, 5=101 and k=2

The first time loop is executed, i=2: c=0, f=1, b_2=1, c=1, f=a mod n

When the loop is executed again, i=1: c=2,f=a*a mod n,b_1=0,c=2, f=a^2 mod n

In the final execution of the loop, i=0: c=4, f=a^4 mod n, b_0=1,c=5, f=a^5 mod n

Execution of fast modular exponentiation in finding a^{560} mod 561 is shown in the table here. The values of i and the variables b_i, c, f in each iteration may be found in the table. Note that the values of f are reduced to mod 560.

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

Table 11.1: computation of a^{560}

Key Generation

Each participant generates a pair of keys in public key cryptosystem. The steps in key generation are:

1. Find two prime numbers p and q.
2. Select e or d and calculate the other.

Note that n (= p*q) is known to any adversary. In order to make discovery of p, q difficult, these must be large. But detection of large prime numbers is difficult. Probabilistic method such as Miller Rabin's method can be used for testing primality. Principle of such methods is to select a large n and test if it is prime. The steps of the probabilistic method are given here.

1. Pick an odd integer *n* at random (e.g., using a pseudorandom number generator).
2. Pick an integer $a < n$ at random.
3. Perform the probabilistic primality test, such as Miller-Rabin, with *a* as a parameter. If *n* fails the test, reject the value *n* and go to step 1.
4. If *n* has passed a sufficient number of tests, accept *n*; otherwise, go to step 2.

This is a somewhat tedious procedure. However, remember that this process is performed relatively infrequently: only when a new pair (*PU*, *PR*) is needed.

Having determined prime numbers *p* and *q*, the process of key generation is completed by selecting a value of *e* and calculating *d* or, alternatively, selecting a value of *d* and calculating *e*. Assuming the former, then we need to select an *e* such that $\gcd(\phi(n), e) = 1$ and then calculate $d \equiv e^{-1} \pmod{\phi(n)}$. Fortunately, there is a single algorithm that will, at the same time, calculate the greatest common divisor of two integers and, if the gcd is 1, determine the inverse of one of the integers modulo the other. The algorithm is called extended Euclid's algorithm.

11.4 SECURITY OF RSA

Four possible approaches to attacking the RSA algorithm are

1. Brute force attack: This involves trying all possible private keys.
2. Mathematical attack: There are several approaches, all equivalent in effort to factoring the product of two primes.
3. Timing attack: These depend on the running time of the decryption algorithm.
4. Chosen cipher text attack: This type of attack exploits properties of the RSA algorithm.

Brute force attack

The defense against the brute-force approach is the same for RSA as for other cryptosystems, namely, to use a large key space. Thus, the larger the number of bits in *d*, the better is the

security. However, because the calculations involved, both in key generation and in encryption/decryption, are complex, the larger the size of the key, the slower the system will run.

Mathematical attack

We can identify three approaches to attacking RSA mathematically.

1. Factor n into its two prime factors. This enables calculation of $\phi(n) = (p - 1) \times (q - 1)$, which in turn enables determination of $d \equiv e^{-1} \pmod{\phi(n)}$.
2. Determine $\phi(n)$ directly, without first determining p and q. Again, this enables determination of $d \equiv e^{-1} \pmod{\phi(n)}$.
3. Determine d directly, without first determining $\phi(n)$.

Most discussions of the cryptanalysis of RSA have focused on the task of factoring n into its two prime factors. Determining $\phi(n)$ given n is equivalent to factoring n. With presently known algorithms, determining d given e and n appears to be at least as time-consuming as the factoring problem . Hence, we can use factoring performance as a benchmark against which to evaluate the security of RSA. For a large n with large prime factors, factoring is a hard problem, but it is not as hard as it used to be.

The threat to larger key sizes is twofold:

1. Continuing increase in computing power and the continuing refinement of factoring algorithms keep reducing the time needed for factoring.
2. Totally new and different algorithms show tremendous speed up.

Researchers suggest some constraints on n which will make factoring a difficult task.

1. p and q should differ in length by only a few digits. Thus, for a 1024-bit key (309 decimal digits), both p and q should be on the order of magnitude of 10^{75} to 10^{100}.
2. Both $(p - 1)$ and $(q - 1)$ should contain a large prime factor.
3. $\gcd(p - 1, q - 1)$ should be small.

In addition, it has been demonstrated that if $e < n$ and $d < n^{1/4}$, then d can be easily determined.

Timing attacks

If one needed yet another lesson about how difficult it is to assess the security of a cryptographic algorithm, the appearance of timing attacks provides a stunning one. Paul Kocher, a cryptographic consultant, demonstrated that a snooper can determine a private key by keeping

track of how long a computer takes to decipher messages. Timing attacks are applicable not just to RSA, but to other public-key cryptography systems as well. This attack is alarming for two reasons: It comes from a completely unexpected direction, and it is a cipher text-only attack.

A timing attack is somewhat analogous to a burglar guessing the combination of a safe by observing how long it takes for someone to turn the dial from number to number. We can explain the attack using the modular exponentiation algorithm we discussed earlier, but the attack can be adapted to work with any implementation that does not run in fixed time. In this algorithm, modular exponentiation is accomplished bit by bit, with one modular multiplication performed at each iteration and an additional modular multiplication performed for each 1 bit.

As Kocher points, the attack is simplest to understand in an extreme case. Suppose the target system uses a modular multiplication function that is very fast in almost all cases but in a few cases takes much more time than an entire average modular exponentiation. The attack proceeds bit-by-bit starting with the leftmost bit, b_k. Suppose that the first j bits are known (to obtain the entire exponent, start with $j = 0$ and repeat the attack until the entire exponent is known). For a given ciphertext, the attacker can complete the first j iterations of the **for** loop. The operation of the subsequent step depends on the unknown exponent bit. If the bit is 1, $d \leftarrow (d \times a) \bmod n$ will be executed. If the observed time to execute the decryption algorithm is slow in a particular iteration then the corresponding bit is assumed to be 1. On the other hand if the execution of the algorithm is fast then the bit is assumed to be 0.

In practice, modular exponentiation implementations do not have such extreme timing variations, in which the execution time of a single iteration can exceed the mean execution time of the entire algorithm. Nevertheless, there is enough variation to make this attack practical.

Although the timing attack is a serious threat, there are simple countermeasures that can be used, including the following.

Constant exponentiation time: Ensure that all exponentiations take the same amount of time before returning a result. This is a simple fix but does degrade performance.

Random delay: Better performance could be achieved by adding a random delay to the exponentiation algorithm to confuse the timing attack. Kocher points out that if defenders don't

add enough noise, attackers could still succeed by collecting additional measurements to compensate for the random delays.

Blinding: Multiply the ciphertext by a random number before performing exponentiation. This process prevents the attacker from knowing what ciphertext bits are being processed inside the computer and therefore prevents the bit-by-bit analysis essential to the timing attack.

RSA Data Security incorporates a blinding feature into some of its products. The private-key operation $M = C^d \bmod n$ is implemented as follows.

1. Generate a secret random number r between 0 and n - 1.
2. Compute $C' = C(r^e) \bmod n$, where e is the public exponent.
3. Compute $M' = (C')^d \bmod n$ with the ordinary RSA implementation.
4. Compute $M=M'r^{-1} \bmod n$. In this equation, r^{-1} is the multiplicative inverse of $r \bmod n$. It can be demonstrated that this is the correct result by observing that $r^{ed} \bmod n = r \bmod n$.

RSA Data Security reports a 2 to 10% performance penalty for blinding.

Chosen cipher text attack

The basic RSA algorithm is vulnerable to a Chosen Ciphertext Attack (CCA). CCA is defined as an attack in which the adversary chooses a number of ciphertexts and is then given the corresponding plaintexts, decrypted with the target's private key. Thus, the adversary could select a plaintext, encrypt it with the target's public key, and then be able to get the plaintext back by having it decrypted with the private key. Clearly, this provides the adversary with no new information. Instead, the adversary exploits properties of RSA and selects blocks of data that, when processed using the target's private key, yield information needed for cryptanalysis.

11.5 SUMMARY

This unit is a description of an important and widely used public key crypto system called RSA. The computational aspects of encryption, decryption and key generation are explained in section 11.3 of this unit. A simple and elegant method to find modulo of powers of a number is detailed in this section. In section 11.4 some special attacks devised against RSA and counter measures against these are outlined.

11.6 KEYWORDS

RSA, Brute force attack, Mathematical attack, Timing attack, Chosen Cipher Attack, Computation of integral powers in modular arithmetic, Random delay.

11.7 QUESTIONS

1. Explain the steps of encryption and decryption in RSA.
2. Give two examples (good) of RSA.
3. Give two bad examples of RSA.
4. Discuss and illustrate computation of a^b mod n.
5. Write about key generation in RSA.
6. Explain the 4 attacks on RSA in detail.

11.8 REFERENCES

1. Behrouz A Forouzan, Cryptography and Network security, McGraw Hill
2. Charlie Kauffman, Radia Perlman, Mike Spciner, Network Security, Pearson Education
3. Wade Trappe, Lawrence Washington, Introduction to Cryptography with Coding Theory, Pearson International
4. William Stallings, Cryptography and Network Security, Pearson

UNIT -12: KEY EXCHANGE AND AUTHENTICATION

Structure

12.0 OBJECTIVES

When you have completed reading the contents of this unit you will

- ✓ Understand the Diffie Hellman Key exchange protocol
- ✓ Appreciate the simplicity and power of Hash functions in cryptography
- ✓ Understand the ways messages can be authenticated
- ✓ Know digital signature methods

12.1 DIFFIE HELLMAN KEY EXCHANGE

This is the first published public key algorithm [1976]. A lot of commercial products use this algorithm for key exchange.

Algorithm:

Global public: q, a prime number

a, a primitive root of q

User A key generation: Select private X_A with $X_A < q$

Calculate public Y_A as $Y_A = a^{XA} \bmod q$

User B key generation: Select private X_B with $X_B < q$

Calculate public Y_B as $Y_B = a^{XB} \bmod q$

Calculation of secret key by user A: $K = Y_B^{XA} \bmod q$

Calculation of secret key by user B: $K = Y_A^{XB} \bmod q$

Secret Key Calculation:

Both A and B calculate secret key. The calculations of both give the same value.

$K = (Y_B)^{XA} \bmod q$ (key of A)

$= (a^{XB} \bmod q)^{XA} \bmod q$

$= (a^{XB})^{XA} \bmod q$

$= a^{XB * XA} \bmod q$

$= (a^{XA})^{XB} \bmod q$

$= (a^{XA} \bmod q)^{XB} \bmod q$

$= (Y_A)^{XB} \bmod q$ (key of B)

Security of Diffie Hellman key exchange

It is easy to calculate exponentials modulo a prime number and difficult to calculate discrete logarithms. For large prime numbers it is close to infeasible.

Example 1: Let q=353, a=3 (primitive root of 353)

Suppose that A and B select secret keys X_A =97 and X_B =233.

Both compute public keys $Y_A = 3^{97} \bmod 353 = 40$ and $Y_B = 3^{233} \bmod 353 = 248$

They exchange public keys

Common secret key computed by A and B as $Y_B^{XA} \bmod q = 248^{97} \bmod 353 = 160$ and $(Y_A)^{XB} \bmod q = 40^{233} \bmod 353 = 160$

Attacker has access to q= 353, Y_A =40 and Y_B = 248

In this simple example, the attacker can find secret key 160 by brute force. Attacker can find secret key by solving $3^a \bmod 353 = 40$ or $3^b \bmod 353 = 248$. a=97 is found (by systematic testing of $3^a \bmod 353 = 40$) which is the private key of A.

Now secret key is computed using $(Y_B)^{XA} \bmod q = 248^{97} \bmod 353 = 160$

Note that prime number is small in this example

In reality for large numbers finding a ($d_{3,353} \log (40)$) is difficult

Key exchange protocols

Suppose that A wishes to set up a connection with B and use a secret key to encrypt messages on that connection. The steps followed are

1. A and B know q, a
2. User A:
   2.1. Generates random $X_A < q$. Calculates $Y_A = a^{XA} \bmod q$
   2.2. Sends Y_A to B
3. User B
   1.1 Generates random $X_B < q$. Calculates $Y_B = a^{XB} \bmod q$ and the secret key $K= (Y_A)^{XB} \bmod q$
   1.2 Sends Y_B to A
2. A calculates $K= (Y_B)^{XA} \bmod q$

As an example of another use of Diffie Hellman algorithm, consider a group of users (of LAN). Each generate a long lasting private key X_i (for user i) and calculate a public key Y_i. These public keys Y_i together with q, a are stored in some central directory. At any time user j can access i's public key, calculate a secret key and use that to send an encrypted message to i. If central directory can be trusted, this form of communication offers confidentiality and authentication. Confidentiality is ensured. Because only i and j can generate the key no other user of the group can read the message. Recipient i knows that only j could have sent this message since i has used j's public key in computing secret key.

Attacks

A specific attack called Man in the middle attack is possible. Suppose A and B wish to exchange keys, and D is the adversary. The attack proceeds as follows.

1. D prepares for the attack by generating two random numbers for private keys say X_{D1}, X_{D2} and then computes public keys Y_{D1}, Y_{D2}
2. A transmits Y_A to B
3. D intercepts Y_A and sends Y_{D1} to B and calculates $K2 = (Y_A)^{X_{D2}} \bmod q$
4. B receives Y_{D1} and calculates $K1 = (Y_{D1})^{X_B} \bmod q$
5. B transmits Y_B to A
6. D intercepts Y_B and transmits Y_{D2} to A. D also calculates $K1 = (Y_B)^{X_{D1}} \bmod q$ and $K2 = (Y_A)^{X_{D2}} \bmod q$
7. A receives Y_{D2} and calculates $K2 = (Y_{D2})^{X_A} \bmod q$

At this point A and B think that they share a secret key, but B and D share the key K1 and A and D share the key K2. All future communications between A and B are compromised in the following way

1. A sends encrypted message M using K2 to B as E(K2, M).
2. D intercepts the encrypted message and recovers M
3. D sends E(K1, M) or E(K1, M') to B, where M' is any message.

Such attacks are possible because it does not authenticate the participants. Such attacks can be overcome with digital signatures.

12.2 HASH FUNCTION

A **hash function** H accepts a variable-length block of data *M* as input and produces a fixed-size hash value *h* = H(*M*) .A "good" hash function has the property that the results of applying the function to a large set of inputs will produce outputs that are evenly distributed and apparently random. In general terms, the principal object of a hash function is data integrity. A change to any bit or bits in *M* results, with high probability, in a change to the hash code. The

kind of hash function needed for security applications is referred to as a cryptographic hash function. A cryptographic hash function is an algorithm for which it is computationally infeasible (because no attack is significantly more efficient than brute force) to find either (a) a data object that maps to a pre-specified hash result (The one-way property) or (b) two data objects that map to the same hash result (the collision-free property). Because of these characteristics, hash functions are often used to determine whether or not data has changed.

Figure 12.1 depicts the general operation of a cryptographic hash function.

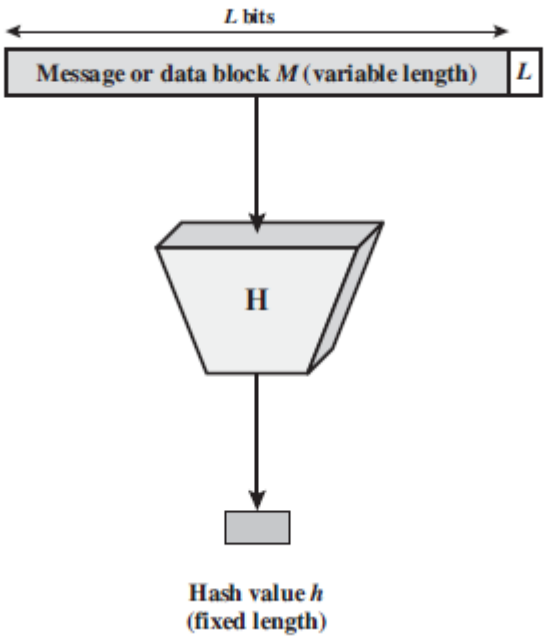


Figure 12.1: Block Diagram of Cryptographic Hash Function; h = H(M)

Typically, the input is padded out to an integer multiple of some fixed length (e.g., 1024 bits), and the padding includes the value of the length of the original message in bits. The length field is a security measure to increase the difficulty for an attacker to produce an alternative message with the same hash value.

Before we go into details of Hash function usage, we list some primary applications of hash functions

i. Message authentication – Mechanism to verify correctness of a message received.
ii. Digital Signature – Mechanism to authenticate sender.
iii. One way password file – Ways of protecting a file of passwords.

iv. Intrusion and Virus detection –Means of detecting if intruder has tampered the file or if virus attack has happened.

v. Pseudo random number generator – Generate random numbers that are useful for many cryptographic algorithms.

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.

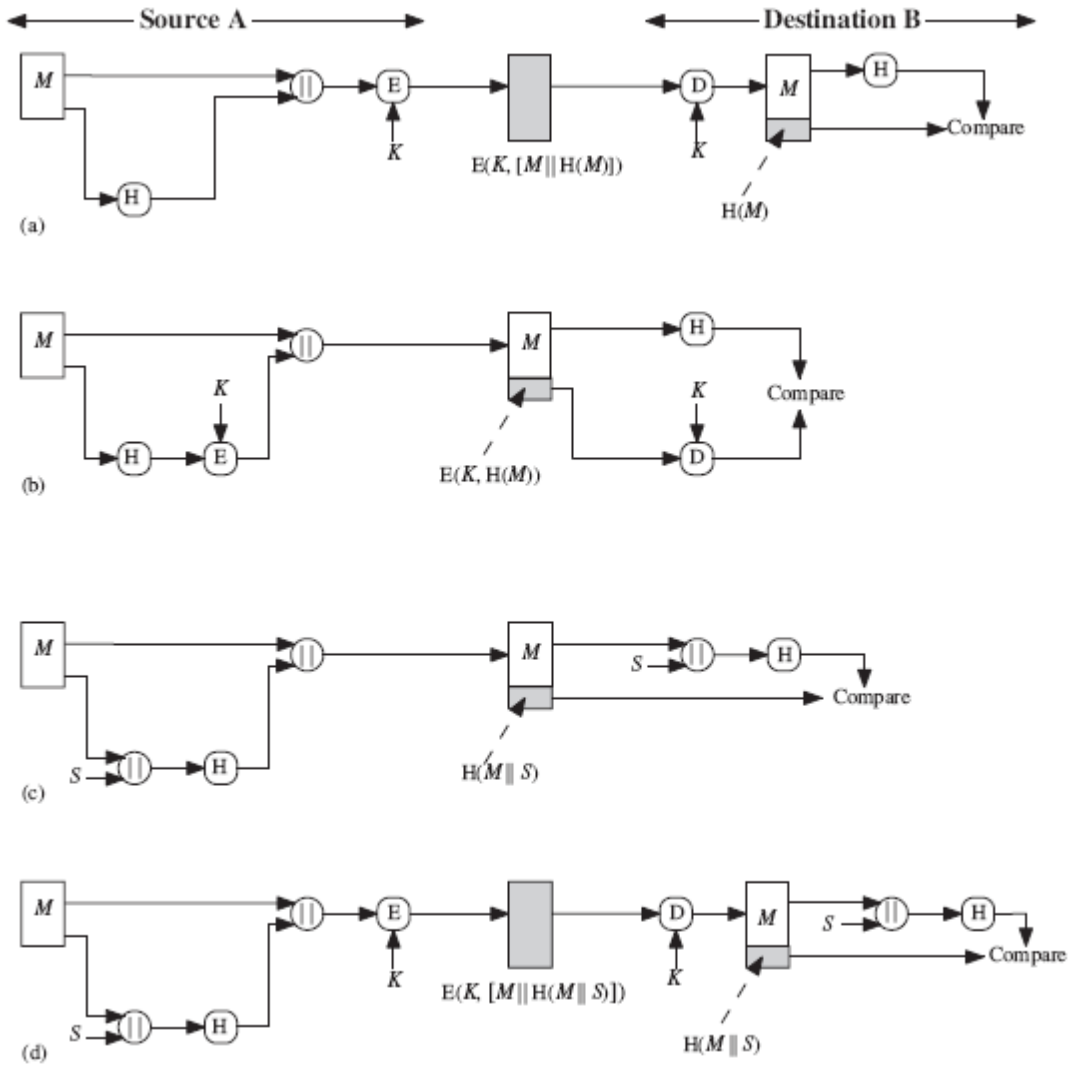


Figure 12.2: Simplified Examples of the use of a Hash Function for Message Authentication

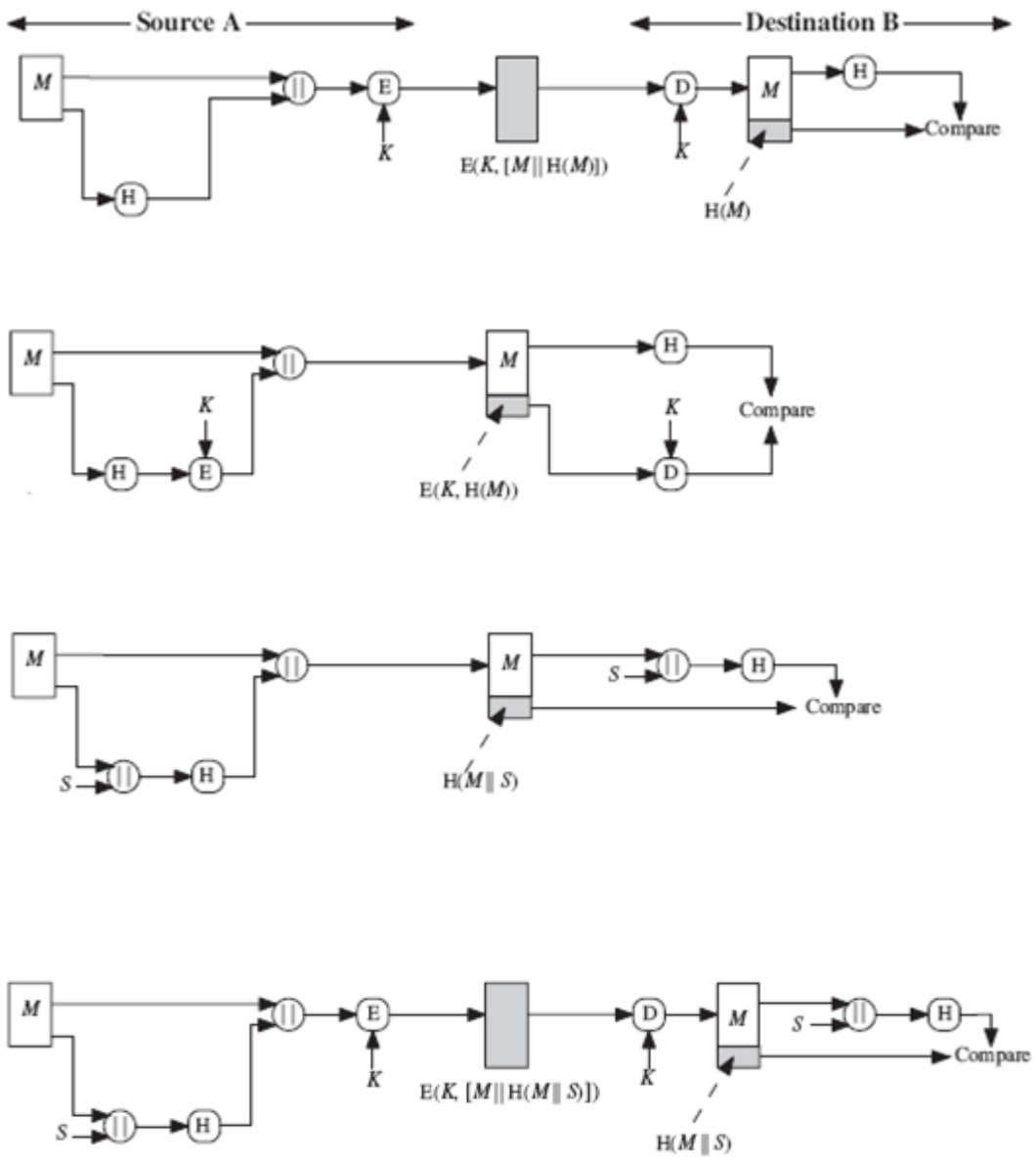


Figure 12.2: Simplified Examples of the use of a Hash Function for Message Authentication

Figures here illustrate a variety of ways in which a hash code can be used to provide message authentication.

12.1 a: The message plus concatenated hash code is encrypted using symmetric encryption. Because only A and B share the secret key, the message must have come from A and has not been altered. The hash code provides the structure or redundancy required to achieve

authentication. Because encryption is applied to the entire message plus hash code, confidentiality is also provided.

12.1 b: Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality.

12.1 c: It is possible to use a hash function but no encryption for message authentication. The technique assumes that the two communicating parties share a common secret value *S*. A computes the hash value over the concatenation of *M* and *S* and appends the resulting hash value to verify. As B has in possession S, hash value can be computed and message can be authenticated. Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.

12.1 d: Confidentiality can be added to the approach of method (c) by encrypting the entire message plus the hash code.

When confidentiality is not required, method (b) has an advantage over methods (a) and (d), which encrypts the entire message, in that less computation is required. Nevertheless, there has been growing interest in techniques that avoid encryption.

Some reasons for avoiding encryption are:

1. Encryption software is relatively slow. Even though the amount of data to be encrypted per message is small, there may be a steady stream of messages into and out of a system.
2. Encryption hardware costs are not negligible. Low-cost chip implementations of DES are available, but the cost adds up if all nodes in a network must have this capability.
3. Encryption hardware is optimized toward large data sizes. For small blocks of data, a high proportion of the time is spent in initialization/invocation overhead.
4. Encryption algorithms may be covered by patents, and there is a cost associated with licensing their use.

Digital signatures

Another important application, which is similar to the message authentication application, is the digital signature. The operation of the digital signature is similar to that of the MAC. In the case of the digital signature, the hash value of a message is encrypted with a user's private key. Anyone who knows the user's public key can verify the integrity of the message that is associated with the digital signature. In this case, an attacker who wishes to alter the message

would need to know the user's private key. As we shall see later, the implications of digital signatures go beyond just message authentication. Figure 12.3 illustrates, in a simplified fashion, how a hash code is used to provide a digital signature.

1. The hash code is encrypted, using public-key encryption with the sender's private key. This provides authentication. It also provides a digital signature, because only the sender could have produced the encrypted hash code. In fact, this is the essence of the digital signature technique.
2. If confidentiality as well as a digital signature is desired, then the message plus the private-key-encrypted hash code can be encrypted using a symmetric secret key. This is a common technique.

Other Applications

Hash functions are commonly used by operating systems to create a one-way password file. A simple scheme to secure user passwords is to find hash values of passwords and store these rather than the password itself. Thus, the actual password is not retrievable by a hacker who gains access to the password file. In simple terms, when a user enters a password, the hash of that password is compared to the stored hash value for verification. This approach to password protection is used by most operating systems.

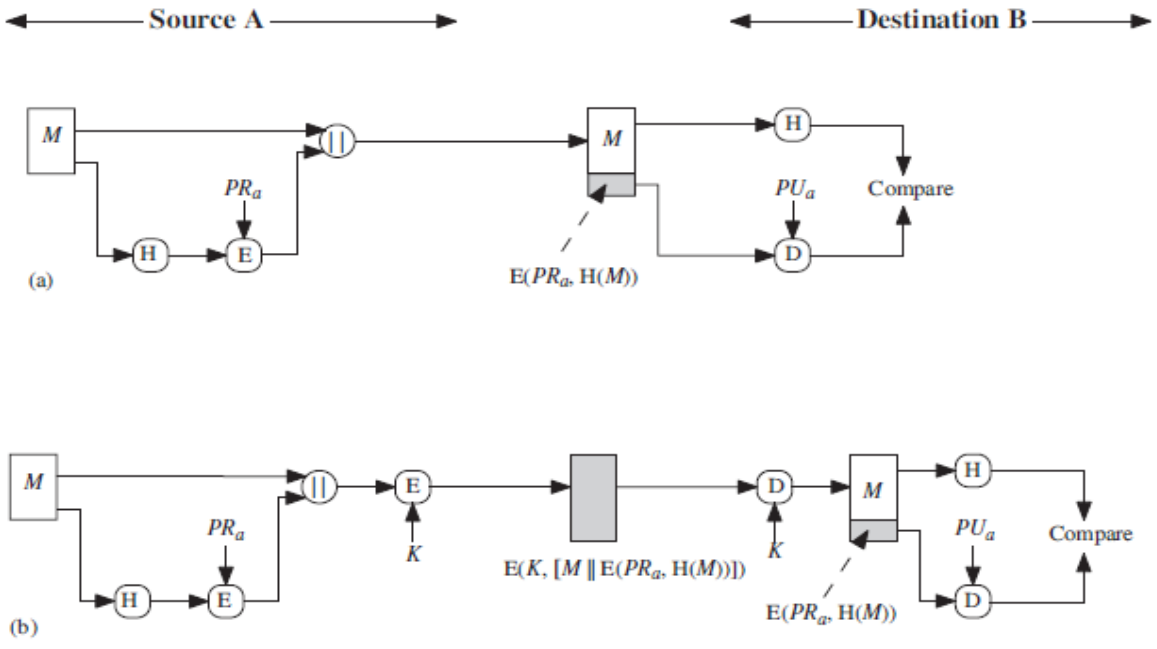


Figure 12.3: Simplified Examples of Digital Signatures

Hash functions can be used for intrusion detection and virus detection. Store H (F) for each file on a system and secure the hash values (e.g., on a CD-R that is kept secure). One can later determine if a file has been modified by re-computing H (F). An intruder would need to change F without changing H (F). A cryptographic hash function can be used to construct a pseudorandom function (PRF) or a pseudorandom number generator (PRNG). A common application for a hash-based PRF is for the generation of symmetric keys.

Two simple hash functions

To get some feel for the security considerations involved in cryptographic hash functions, we present two simple, insecure hash functions in this section. All hash functions operate using the following general principles. The input (message, file, etc.) is viewed as a sequence of -bit blocks. The input is processed one block at a time in an iterative fashion to produce an -bit hash function. One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of every block. This can be expressed as

$$C_i = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{in}$$

Where

C_i = i^{th} bit of the hash code, $1 \leq i \leq n$

n = number of blocks in the input

b_{ij} = i^{th} bit in j^{th} block

$\oplus$ = XOR operator

This operation produces a simple parity for each bit position and is known as a longitudinal redundancy check. It is reasonably effective for random data as a data integrity check. Each n-bit hash value is equally likely. Thus, the probability that a data error will result in an unchanged hash value is 2^{-n}. With more predictably formatted data, the function is less effective. For example, in most normal text files, the high-order bit of each octet is always zero. So if a 128-bit hash value is used, instead of an effectiveness of 2^{-128}, the hash function on this type of data has an effectiveness of 2^{-112}.

A simple way to improve matters is to perform a one-bit circular shift, or rotation, on the hash value after each block is processed. The procedure can be summarized as follows.

1. Initially set the n-bit hash value to zero

2. Process each successive n-bit block of data as follows:

a. Pick the ith bit of the first block

Increase i by 1 and pick the respective bits for hashing

That is, $C_i = b_{i1} \oplus b_{(i+1)2} \oplus \ldots\ldots\ldots\ldots\ldots \oplus b_{(i+n-1)\,n}$

This has the effect of "randomizing" the input more completely and overcoming any regularity that appear in the input.

12.3 MESSAGE AUTHENTICATION CODES

One of the most fascinating and complex areas of cryptography are that of message authentication and the related area of digital signatures. The purpose of this section and the next is to provide a broad overview of the subject.

Message authentication requirements

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

1. **Disclosure:** Release of message contents to any person or process not possessing the appropriate cryptographic key.
2. **Traffic analysis:** Discovery of the pattern of traffic between parties. In either a connection-oriented or connectionless environment, the number and length of messages between parties could be determined.
3. **Masquerade:** Insertion of messages into the network from a fraudulent source. This includes the creation of messages by an opponent that are purported to come from an authorized entity. Also included are fraudulent acknowledgments of message receipt or no receipt by someone other than the message recipient.
4. **Content modification:** Changes to the contents of a message, including insertion, deletion, transposition, and modification.
5. **Sequence modification:** Any modification to a sequence of messages between parties, including insertion, deletion, and reordering.

6. **Timing modification:** Delay or replay of messages. In a connection-oriented application, an entire session or sequence of messages could be a replay of some previous valid session, or individual messages in the sequence could be delayed or replayed. In a connectionless application, an individual message (e.g., datagram) could be delayed or replayed.
7. **Source repudiation:** Denial of transmission of message by source.
8. **Destination repudiation:** Denial of receipt of message by destination.

Measures to deal with the first two attacks are in the realm of message confidentiality. Measures to deal with items (3) through (6) in the foregoing list are generally regarded as message authentication. Mechanisms for dealing specifically with item (7) come under the heading of digital signatures. Generally, a digital signature technique will also counter some or all of the attacks listed under items (3) through (6). Dealing with item (8) may require a combination of the use of digital signatures and a protocol designed to counter this attack.

In summary, message authentication is a procedure to verify that received messages come from the alleged source and have not been altered. Message authentication may also verify sequencing and timeliness. A digital signature is an authentication technique that also includes measures to counter repudiation by the source.

Message authentication functions

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

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

- **Hash function:** A function that maps a message of any length into a fixed length hash value, which serves as the authenticator.
- **Message encryption:** The cipher text of the entire message serves as its authenticator.
- **Message authentication code (MAC):** A function of the message and a secret key that produces a fixed-length value that serves as the authenticator.

Hash functions and how they serve as message authenticator has been discussed in previous section. We examine here briefly the other two options for authenticating messages.

Message encryption:

Figures 12.4 to 12.7 show authenticating messages with symmetric and asymmetric encryptions.

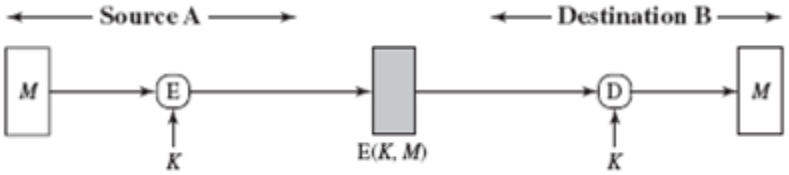


Figure 12.4: Symmetric encryption: confidentiality and authentication

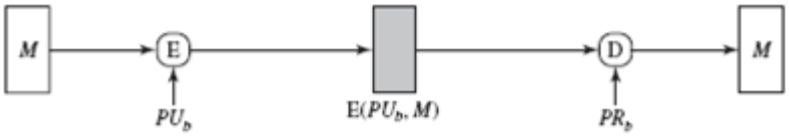


Figure 12.5: Public-key encryption: confidentiality

Figure 12.6: Public-key encryption: authentication and signature

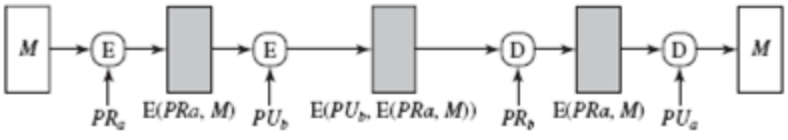


Figure 12.7: Public-key encryption: confidentiality, authentication and signature

In figure 12.4, B is confirmed of the sender A since only A and B has unique key K. This is authentication in symmetric encryption. Also B knows that the message received is not altered.

This is because the opponent, not having knowledge of K would not know how to alter the bit patterns in cipher text to produce the desired changes in the plain text. Any cipher text X will produce a text with decryption algorithm. An opponent can deliberately change bit patterns and the decryption algorithm will produce a text. There should be a way of verifying the source A. This can be done using some checksum for each block (frame) called FCS (frame checksum) and append this to message and then encrypt. Attacker will find it extremely difficult to change cipher text and at same time have same FCS for all frames.

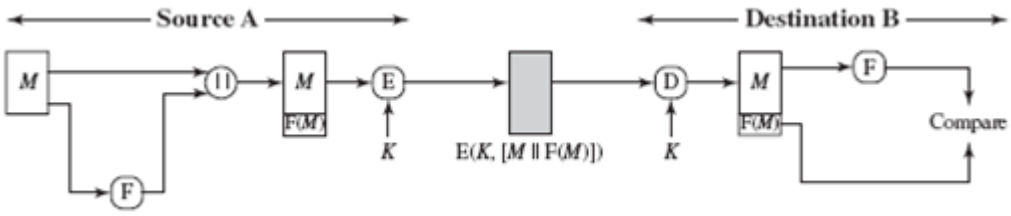


Figure 12.8: Internal error control

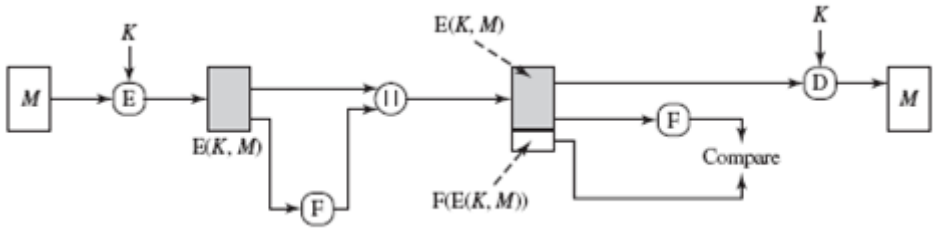


Figure 12.9: External error control

In figures 12.8 and 12.9 two such methods using FCS is shown. With internal error control codes, detection of tampering is easy. With external error control codes, opponent can change cipher bits here and there and still get matching FCS. But the decrypted plain text may be meaningless. However such a successful confusion and disruption to operation secret sharing of sensitive information is possible by the opponent.

Figure 12.5 shows confidentiality of message. B is confirmed that message is not tampered. However anybody else other than A could have sent this message, i.e. there is no sender authentication. If A uses his private key and encrypts the message and sends it to B, then sender

is authenticated. This is shown in figure 12.6, with this scheme there is no confidentiality. The receiver should be informed (or know previously) about some internal structure of the message so that he is able to confirm that message is not altered by the adversary.

The scheme in figure 12.7 provides confidentiality, sender authentication and signature of the sender.

Message authentication code (MAC):

An alternative authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as a cryptographic checksum or MAC that is appended to the message. This technique assumes that two communicating parties, say A and B, share a common secret key .When A has a message to send to B, it calculates the MAC as a function of the message and the key: MAC = MAC (K, M), where M is the input message, C is the MAC function, K is the shared secret key, MAC is the message authentication code.

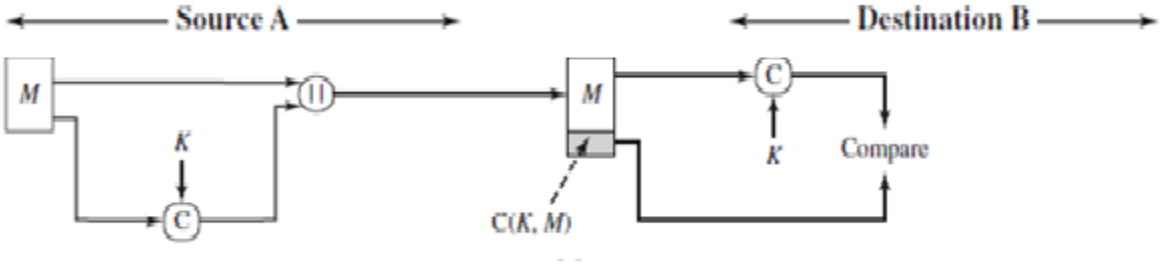


Figure 12.10: Message authentication

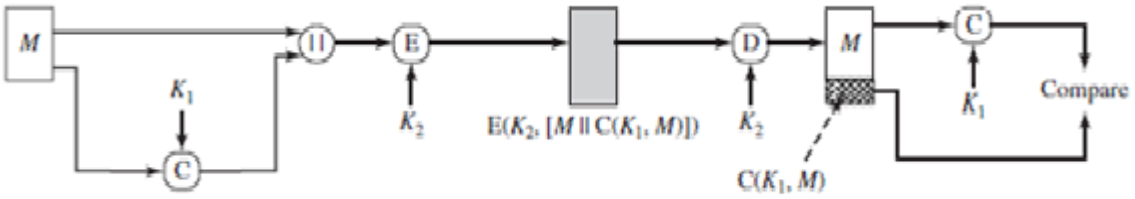


Figure 12.11: Message authentication and confidentiality; authentication tied to plain text

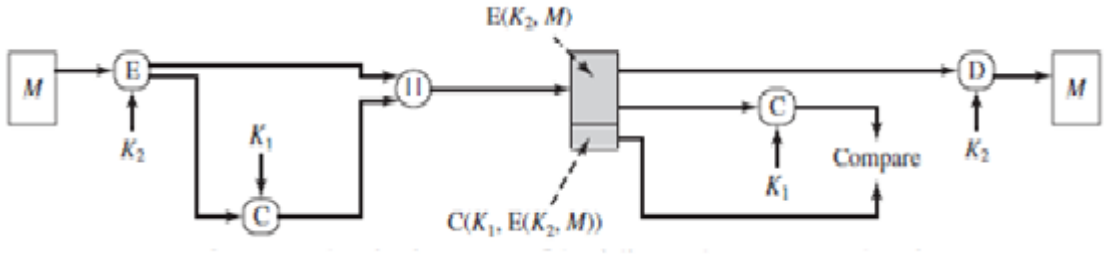


Figure 12.12: Message authentication and confidentiality; authentication tied to cipher text

The message plus MAC are transmitted to the intended recipient. The recipient performs the same calculation on the received message, using the same secret key, to generate a new MAC. The received MAC is compared to the calculated MAC (Figure 12.10). If we assume that only the receiver and the sender know the identity of the secret key, and if the received MAC matches the calculated MAC, then

1. The receiver is assured that the message has not been altered. If an attacker alters the message but does not alter the MAC, then the receiver's calculation of the MAC will differ from the received MAC. Because the attacker is assumed not to know the secret key, the attacker cannot alter the MAC to correspond to the alterations in the message.
2. The receiver is assured that the message is from the alleged sender. Because no one else knows the secret key, no one else could prepare a message with a proper MAC.
3. If the message includes a sequence number (such as is used with HDLC, X.25, and TCP), then the receiver can be assured of the proper sequence because an attacker cannot successfully alter the sequence number.

A MAC function is similar to encryption. One difference is that the MAC algorithm need not be reversible, as it must be for decryption. In general, the MAC function is a many-to-one function. The domain of the function consists of messages of some arbitrary length, whereas the range consists of all possible MACs and all possible keys. If an *n*-bit MAC is used, then there are 2^n possible MACs, whereas there are N possible messages with $N >> 2^n$. Furthermore, with a k-bit key, there are 2^k possible keys.

For example, suppose that we are using 100-bit messages and a 10-bit MAC. Then, there are a total of 2^{100} different messages but only 2^{10} different MACs. So, on average, each MAC value is

generated by a total of $2^{100}/2^{10} = 2^{90}$ different messages. If a 5-bit key is used, then there are 2^5=32 different mappings from the set of messages to the set of MAC values.

It turns out that, because of the mathematical properties of the authentication function, it is less vulnerable to being broken than encryption. The process depicted in Figure 12.10 provides authentication but not confidentiality, because the message as a whole is transmitted in the clear. Confidentiality can be provided by performing message encryption either after (Figure 12.11) or before (Figure 12.12) the MAC algorithm. In both these cases, two separate keys are needed, each of which is shared by the sender and the receiver. In the first case, the MAC is calculated with the message as input and is then concatenated to the message. The entire block is then encrypted. In the second case, the message is encrypted first. Then the MAC is calculated using the resulting cipher text and is concatenated to the cipher text to form the transmitted block. Typically, it is preferable to tie the authentication directly to the plaintext, so the method of Figure 12.11 is used.

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

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

attached to the program, it could be checked whenever assurance was required of the integrity of the program.

4. Three other rationales may be added.
5. For some applications, it may not be of concern to keep messages secret, but it is important to authenticate messages. An example is the Simple Network Management Protocol Version 3 (SNMPV3), which separates the functions of confidentiality and authentication. For this application, it is usually important for a managed system to authenticate incoming SNMP messages, particularly if the message contains a command to change parameters at the managed system. On the other hand, it may not be necessary to conceal the SNMP traffic.
6. Separation of authentication and confidentiality functions affords architectural flexibility. For example, it may be desired to perform authentication at the application level but to provide confidentiality at a lower level, such as the transport layer.
7. A user may wish to prolong the period of protection beyond the time of reception and yet allow processing of message contents. With message encryption, the protection is lost when the message is decrypted, so the message is protected against fraudulent modifications only in transit but not within the target system.

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

12.4 DIGITAL SIGNATURES

Properties

Message authentication protects messages exchanged between two parties. However, it does not protect the two parties against each other. Several forms of dispute between the two are possible.

For example, suppose that A sends an authenticated message to B, using one of the schemes of Figures 12.4 to 12.7. Consider the following disputes that could arise.

1. B may forge a different message and claim that it came from A. B would simply have to create a message and append an authentication code using the key that A and B share.
2. A can deny sending the message. Because it is possible for B to forge a message, there is no way to prove that A did in fact send the message.

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

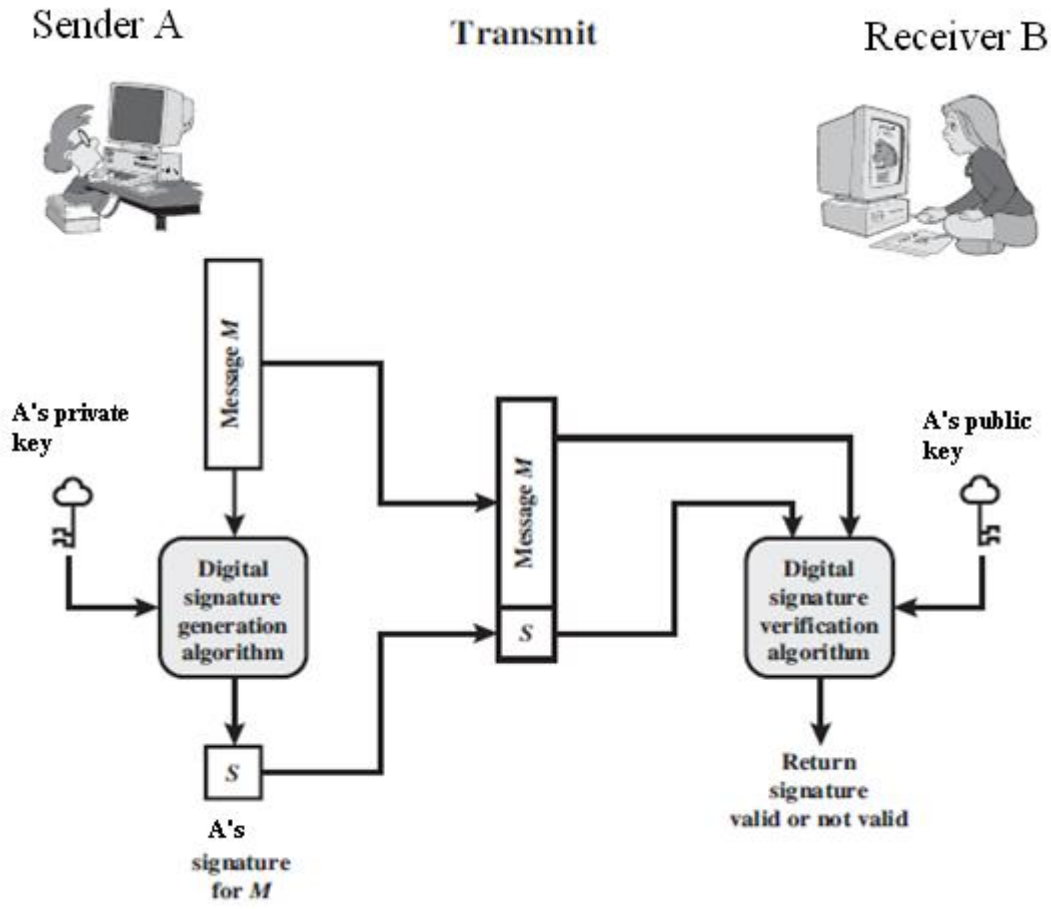


Figure 12.13: Generic model of digital signature process

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

- It must verify the author and the date and time of the signature.

• It must authenticate the contents at the time of the signature.

• It must be verifiable by third parties, to resolve disputes.

Thus, the digital signature function includes the authentication function.

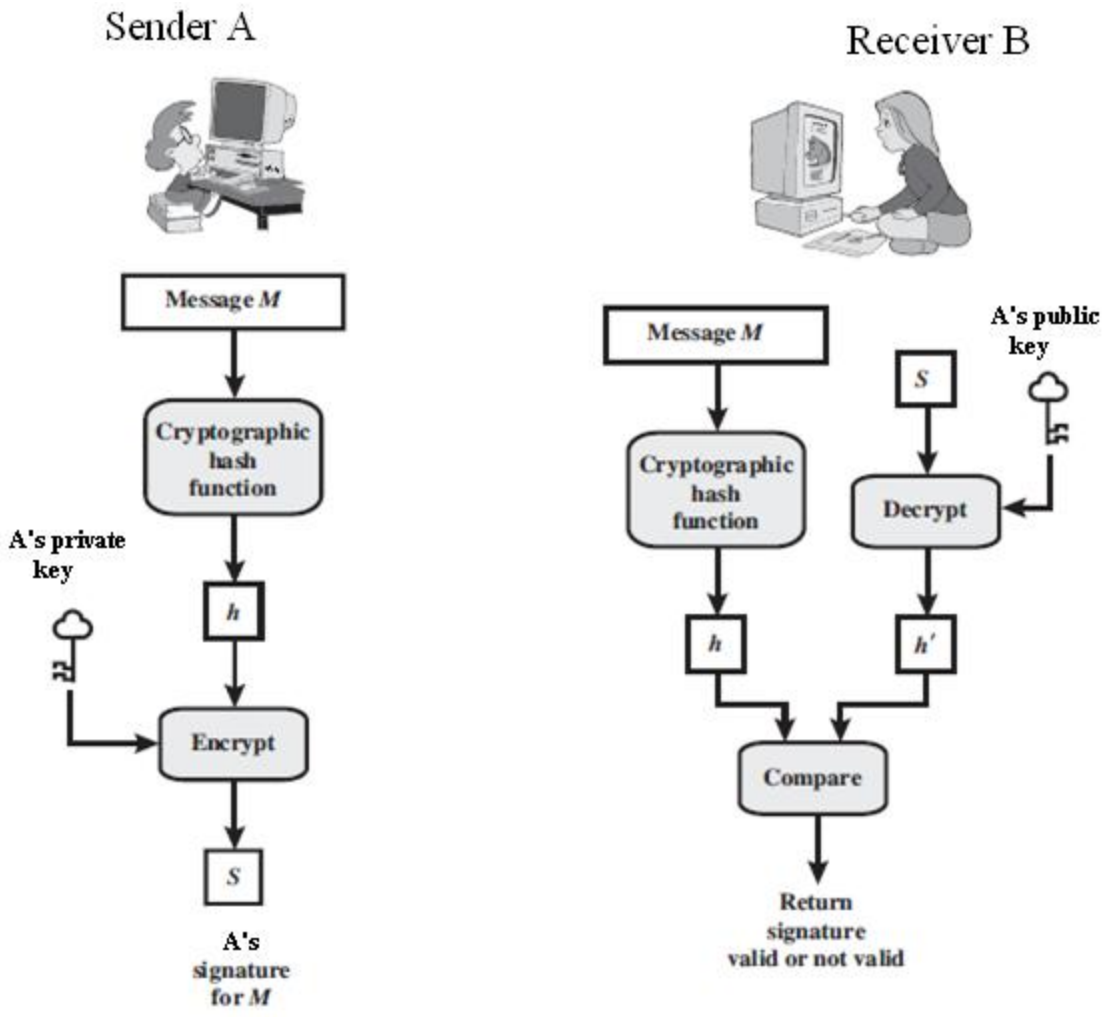


Figure 12.14: Simplified Depiction of Essential Elements of Digital Signature Process

Attacks and Forgeries:

Following is the list of the types of attacks, in order of increasing severity. Here A denotes the user whose signature method is being attacked, and C denotes the attacker.

- **Key-only attack:** C only knows A's public key.
- **Known message attack:** C is given access to a set of messages and their signatures.

- **Generic chosen message attack:** C chooses a list of messages before attempting to breaks A's signature scheme, independent of A's public key. C then obtains from A valid signatures for the chosen messages. The attack is generic, because it does not depend on A's public key; the same attack is used against everyone.
- **Directed chosen message attack:** Similar to the generic attack, except that the list of messages to be signed is chosen after C knows A's public key but before any signatures are seen.
- **Adaptive chosen message attack:** C is allowed to use A as an "oracle." This means the A may request signatures of messages that depend on previously obtained message–signature pairs.

The attack is said to be a success if C can do any one of the following:

- **Total break:** C determines A's private key.
- **Universal forgery:** C finds an efficient signing algorithm that provides an equivalent way of constructing signatures on arbitrary messages.
- **Selective forgery:** C forges a signature for a particular message chosen by C.
- **Existential forgery:** C forges a signature for at least one message. C has no control over the message. Consequently, this forgery may only be a minor nuisance to A.

Requirements of Digital Signature:

On the basis of the properties and attacks just discussed, we can formulate the following requirements for a digital signature.

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

A secure hash function, embedded in a scheme such as that of Figure 12.14 provides a basis for satisfying these requirements. However, care must be taken in the design of the details of the scheme.

Direct digital Signature:

The term direct digital signature refers to a digital signature scheme that involves only the communicating parties (source, destination). It is assumed that the destination knows the public key of the source.

Confidentiality can be provided by encrypting the entire message plus signature with a shared secret key (symmetric encryption). Note that it is important to perform the signature function first and then an outer confidentiality function. In case of dispute, some third party must view the message and its signature. If the signature is calculated on an encrypted message, then the third party also needs access to the decryption key to read the original message. However, if the signature is the inner operation, then the recipient can store the plaintext message and its signature for later use in dispute resolution.

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

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

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 latest version (2009) incorporates digital signature algorithms based on RSA and on elliptic curve cryptography.

The DSS Approach:

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

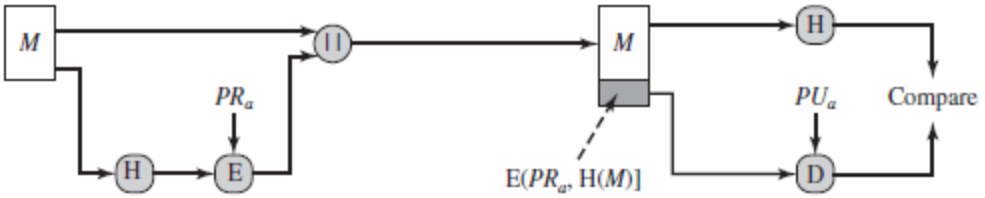


Figure 12.15: RSA approach

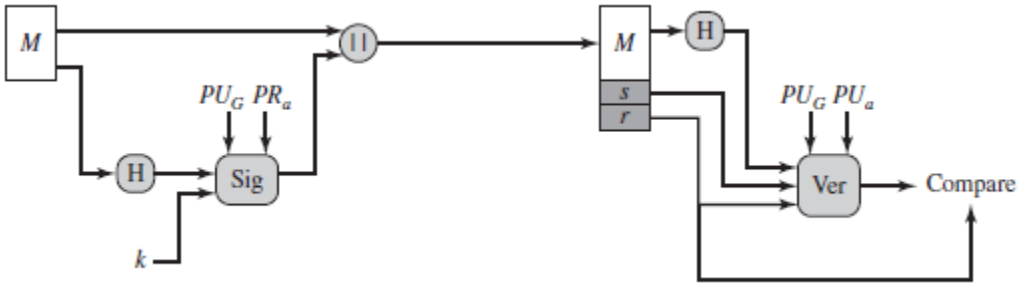


Figure 12.16: DSS approach

Figure 12.15 and 12.16 are two approaches for digital signature. 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 (PR_a) and a set of parameters known to a group of communicating principals. We can consider this set to constitute a global public key (PU_G). The result is a signature consisting of two components, labeled *s* and *r*.

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

12.5 SUMMARY

This unit introduced topics on data integrity. Section 1 is about secured key exchange method proposed by Diffie and Hellman. Section 2 discusses the use of hashing technique to secure data. In section 3 message authentication ways are described. Section 4 is about digital signature procedures.

12.6 KEYWORDS

Diffie Hellman Key exchange, Hash function, Message authentication function, Frame checksum, MAC, Internal and external error control, Digital signature, Direct digital signature, DSS.

12.7 QUESTIONS

1. Explain Diffie Hellman Key exchange protocol.
2. Illustrate Diffie Hellman Key exchange protocol.
3. What are applications of hash functions in cryptography?

4. Discuss using figures various ways of using hash function for message authentication?
5. Explain how hash functions are used for digital signatures.
6. Explain any two hash functions.
7. Mention the requirements of a message authentication function.
8. What are the various ways of authenticating messages? Explain using figures?
9. What do you mean by internal and external error control? Explain.
10. What is digital signature?
11. Discuss attacks and forgeries on digitally signed messages.
12. What are requirements of digital signature?
13. Explain direct digital signature.
14. Compare RSA and DSS signature methods.

12.8 REFERENCES

1. Atul Kahate, Cryptography and Network Security, Tata MCGrawHill
2. Behrouz A Forouzan, Cryptography and Network security, McGraw Hill
3. William Stallings, Cryptography and Network Security, Pearson

MODULE 4

NETWORK, SYSTEM SECURITY

UNIT -13: E-MAIL SECURITY

Structure

13.0 OBJECTIVES

After thorough understanding of the material introduced in this unit you will

- ✓ Understand the ways of securing e-mails and an important principle in email security called pretty good privacy
- ✓ Various authentication protocols

13.1 E-MAIL SECURITY

In virtually all distributed environments, electronic mail is the most heavily used network-based application. Users expect to be able to, and do, send e-mail to others who are connected directly or indirectly to the Internet, regardless of host operating system or communications suite. With

the explosively growing reliance on e-mail, there grows a demand for authentication and confidentiality services. Two schemes stand out as approaches that enjoy widespread use: Pretty Good Privacy (PGP) and S/MIME. We discuss in detail the PGP approach.

Pretty good privacy:

PGP is a remarkable phenomenon. Largely the effort of a single person, Phil Zimmermann, PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications. In essence, Zimmermann has done the following:

1. Selected the best available cryptographic algorithms as building blocks.
2. Integrated these algorithms into a general-purpose application that is independent of operating system and processor and that is based on a small set of easy-to-use commands.
3. Made the package and its documentation, including the source code, freely available via the Internet, bulletin boards, and commercial networks such as AOL (America On Line).
4. Entered into an agreement with a company (Viacrypt, now Network Associates) to provide a fully compatible, low-cost commercial version of PGP.

PGP has grown explosively and is now widely used. A number of reasons can be cited for this growth.

1. It is available free worldwide in versions that run on a variety of platforms, including Windows, UNIX, Macintosh, and many more. In addition, the commercial version satisfies users who want a product that comes with vendor support.
2. It is based on algorithms that have survived extensive public review and are considered extremely secure. Specifically, the package includes RSA, DSS, and Diffie-Hellman for public-key encryption; CAST-128, IDEA, and 3DES for symmetric encryption; and SHA-1 for hash coding.
3. It has a wide range of applicability, from corporations that wish to select and enforce a standardized scheme for encrypting files and messages to individuals who wish to communicate securely with others worldwide over the Internet and other networks.
4. It was not developed by, nor is it controlled by, any governmental or standards organization. For those with an instinctive distrust of "the establishment," this makes PGP attractive.
5. PGP is now on an Internet standards track. Nevertheless, PGP still has an aura of an antiestablishment endeavor.

We begin with an overall look at the operation of PGP. Next, we examine how cryptographic keys are created and stored. Then, we address the vital issue of public-key management.

Notation:

Most of the notation used in this chapter has been used before, but a few terms are new. It is perhaps best to summarize those at the beginning. The following symbols are used.

K_s = session key used in symmetric encryption scheme

PR_a = private key of user A, used in public-key encryption scheme

PU_a = public key of user A, used in public-key encryption scheme

EP=public-key encryption

DP= public-key decryption

EC=symmetric encryption

DC = symmetric decryption

H=Hash function

||=concatenation

Z=compression using ZIP algorithm

R64=conversion to radix 64 ASCII format

The PGP documentation often uses the term *secret key* to refer to a key paired with a public key in a public-key encryption scheme. As was mentioned earlier, this practice risks confusion with a secret key used for symmetric encryption. Hence, we use the term *private key* instead.

The figures below exhibit the three modes of PGP.

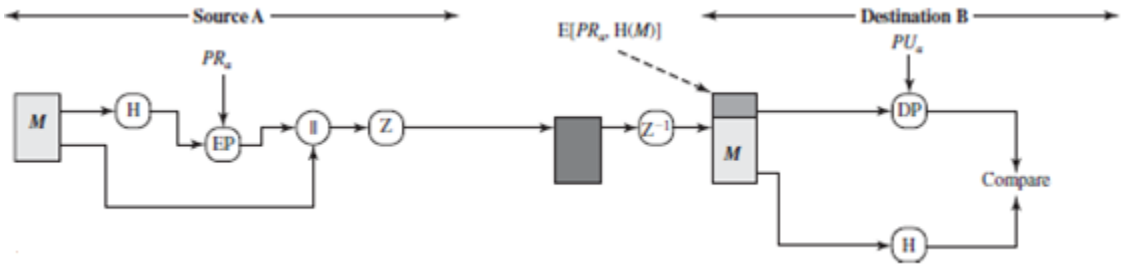


Figure 13.1: Authentication only

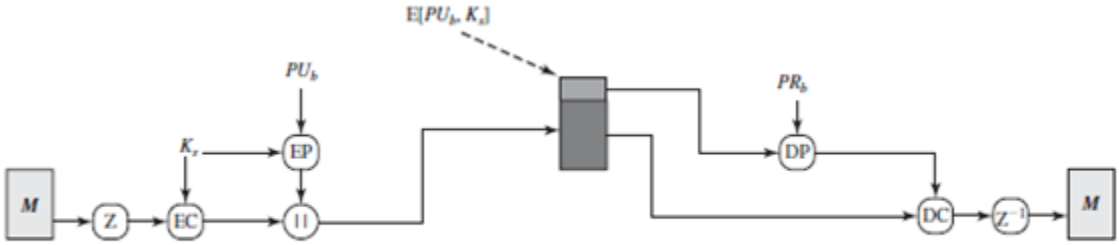


Figure 13.2: Confidentiality only

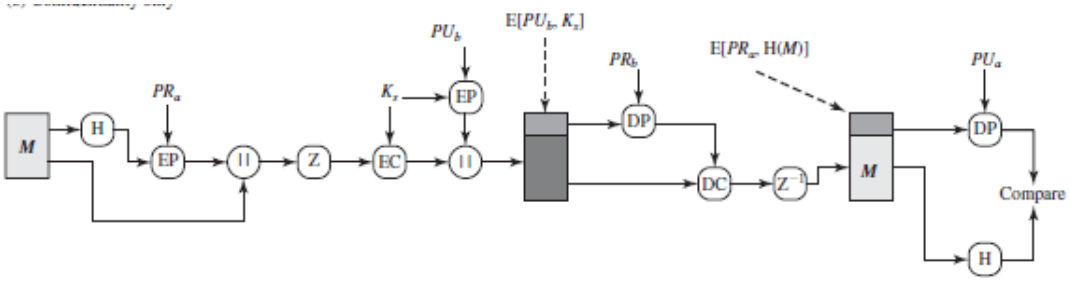


Figure 13.3: Confidentiality and authentication

Sequences of steps for the PGP service in figure 13.1 is as follows:

1. The sender creates a message.
2. SHA-1 is used to generate a 160-bit hash code of the message.
3. The hash code is encrypted with RSA using the sender's private key, and the result is prepended to the message.
4. The receiver uses RSA with the sender's public key to decrypt and recover the hash code.
5. The receiver generates a new hash code for the message and compares it with the decrypted hash code. If the two match, the message is accepted as authentic.

The PGP service in figure 13.2 has the following steps:

1. The sender generates a message and a random 128-bit number to be used as a session key for this message only.

2. The message is encrypted using CAST-128 (or IDEA or 3DES) with the session key- Message confidentiality.

3. The session key is encrypted with RSA using the recipient's public key and is prepended to the message- session key confidentiality.

4. The receiver uses RSA with its private key to decrypt and recover the session key.

5. The session key is used to decrypt the message.

Finally the PGP service in figure 13.3 offers both confidentiality and authentication.

First, a signature is generated for the plaintext message and prepended to the message. Then the plaintext message plus signature is encrypted using CAST-128 (or IDEA or 3DES), and the session key is encrypted using RSA (or ElGamal). This sequence is preferable to the opposite: encrypting the message and then generating a signature for the encrypted message. It is generally more convenient to store a signature with a plaintext version of a message. Furthermore, for purposes of third-party verification, if the signature is performed first, a third party need not be concerned with the symmetric key when verifying the signature.

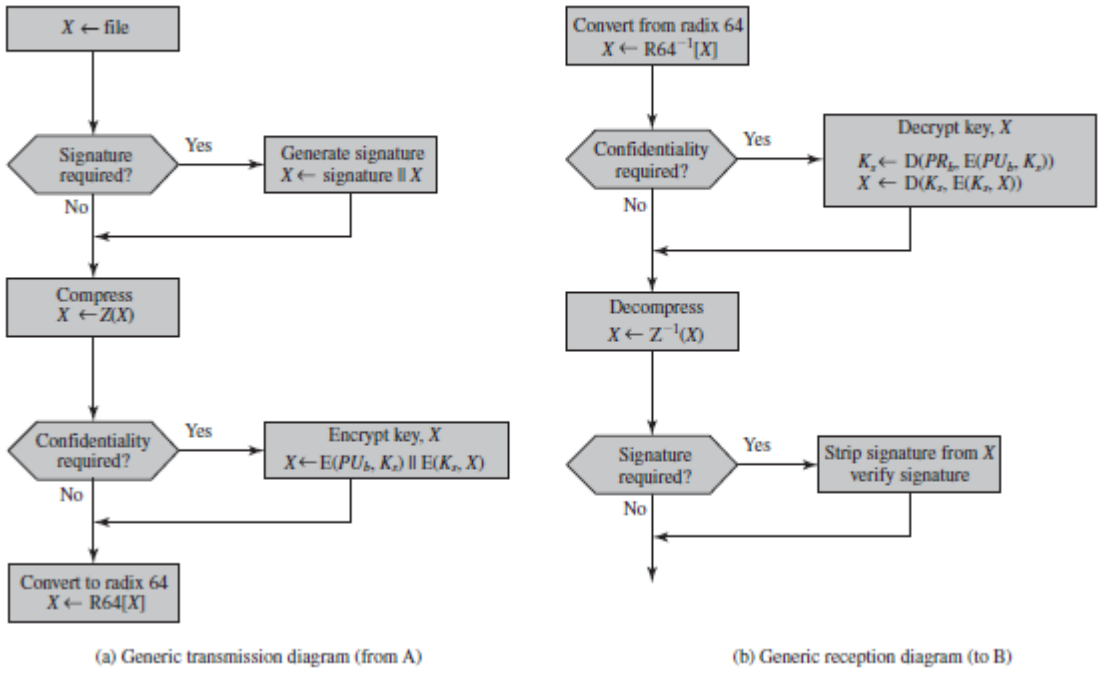
In summary, when both services are used, the sender first signs the message with its own private key, then encrypts the message with a session key, and finally encrypts the session key with the recipient's public key.

With a note on compatibility between end systems in E-mail communication, we close this section.

E-mail compatibility: When PGP is used, at least part of the block to be transmitted is encrypted. If only the signature service is used, then the message digest is encrypted (with the sender's private key). If the confidentiality service is used, the message plus signature (if present) are encrypted (with a one-time symmetric key). Thus, part or the entire resulting block consists of a stream of arbitrary 8-bit octets. However, many electronic mail systems only permit the use of blocks consisting of ASCII text. To accommodate this restriction, PGP provides the service of converting the raw 8-bit binary stream to a stream of printable ASCII characters. The scheme used for this purpose is radix-64 conversion. Each group of three octets of binary data is mapped into four ASCII characters. This format also appends a CRC to detect transmission errors.

Figure 13.4 shows the relationship among the four services so far discussed. On transmission (if it is required), a signature is generated using a hash code of the uncompressed plaintext. Then the plaintext (plus signature if present) is compressed. Next, if confidentiality is required, the block (compressed plaintext or compressed signature plus plaintext) is encrypted and prepended with the public-key encrypted symmetric encryption key. Finally, the entire block is converted to radix-64 format.

On reception, the incoming block is first converted back from radix-64 format to binary. Then, if the message is encrypted, the recipient recovers the session key and decrypts the message. The resulting block is then decompressed. If the message is signed, the recipient recovers the transmitted hash code and compares it to its own calculation of the hash code.



(a) Generic transmission diagram (from A) b) Generic reception diagram (to B)

Figure 13.4: Transmission and Reception of PGP Messages

13.2 AUTHENTICATION SERVICES

In most computer security contexts, user authentication is the fundamental building block and the primary line of defense. User authentication is the basis for most types of access control and for user accountability.

The process of verifying an identity claimed by or for a system entity. An authentication process consists of two steps:

- **Identification step:** Presenting an identifier to the security system. (Identifiers should be assigned carefully, because authenticated identities are the basis for other security services, such as access control service).
- **Verification step:** Presenting or generating authentication information that corroborates the binding between the entity and the identifier.

In essence, identification is the means by which a user provides a claimed identity to the system; user authentication is the means of establishing the validity of the claim.

There are four general means of authenticating a user's identity, which can be used alone or in combination:

- **Something the individual knows:** Examples include a password, a personal identification number (PIN), or answers to a prearranged set of questions.
- **Something the individual possesses:** Examples include cryptographic keys, electronic keycards, smart cards, and physical keys. This type of authenticator is referred to as a *token*.
- **Something the individual is (static biometrics):** Examples include recognition by fingerprint, retina, and face.
- **Something the individual does (dynamic biometrics):** Examples include recognition by voice pattern, handwriting characteristics, and typing rhythm.

All of these methods, properly implemented and used, can provide secure user authentication. However, each method has problems. An adversary may be able to guess or steal a password. Similarly, an adversary may be able to forge or steal a token. A user may forget a password or lose a token. Furthermore, there is a significant administrative overhead for managing password and token information on systems and securing such information on

systems. With respect to biometric authenticators, there are a variety of problems, including dealing with false positives and false negatives, user acceptance, cost, and convenience. For network-based user authentication, the most important methods involve cryptographic keys and something the individual knows, such as a password.

Authentication is of two kinds: Mutual authentication and one way authentication

Mutual Authentication:

An important application area is that of mutual authentication protocols. Such protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys.

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

Approaches for coping with replay attacks are:

1. **Attach sequence number with messages.**
   An attacker cannot attach new sequence number while replaying
2. **Include time stamps with messages.**
   This requires synchronized clocks between communicating parties. Only if messages are received within short time since it is sent, it is understood to be genuine.
3. **Using nonce with messages.**
   Communicating parties exchange nonce (could be random number) before the actual messages is sent.

One-Way Authentication:

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

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

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

We now give two protocols one for mutual authentication and one for one way authentication.

Mutual authentication protocol (for symmetric encryption):

$1. A \rightarrow KDC: \ ID_A \| ID_B \| N_1$

$2. KDC \rightarrow A: \ E(K_a, [K_s \| ID_B \| N_1 \| E(K_b, [K_s \| ID_A])])$

$3. A \rightarrow B: \quad E(K_b, [K_s \| ID_A])$

$4. B \rightarrow A: \quad E(K_s, N_2)$

$5. A \rightarrow B: \quad E(K_s, f(N_2))$

One way authentication protocol (for symmetric encryption):

$1. \ A \rightarrow KDC: \ ID_A \| ID_B \| N_1$

$2. \ KDC \rightarrow A: \ E(K_a, [K_s \| ID_B \| N_1 \| E(K_b, [K_s \| ID_A])])$

$3. \ A \rightarrow B: \quad E(K_b, [K_s \| ID_A]) \| E(K_s, M)$

Mutual authentication protocol (for asymmetric encryption):

$1. \ A \rightarrow AS: ID_A \| ID_B$

$2. \ AS \rightarrow A: E(PR_{as}, [ID_A \| PU_a \| T]) \| E(PR_{as}, [ID_B \| PU_b \| T])$

$3. \ A \rightarrow B: \ E(PR_{as}, [ID_A \| PU_a \| T]) \| E(PR_{as}, [ID_B \| PU_b \| T]) \| E(PU_b, E(PR_a, [K_s \| T]))$

One way authentication for asymmetric (for asymmetric encryption):

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

$$A \rightarrow B : E(PU_b, K_s) \parallel E(K_s, M)$$

If authentication is the primary concern, then a digital signature may suffice,

$$A \rightarrow B : M \parallel E(PR_a, H(M))$$

13.4 KERBORES

Kerberos is an authentication service developed as part of Project Athena at MIT. The problem that Kerberos addresses is this: Assume an open distributed environment in which users at workstations wish to access services on servers distributed throughout the network .We would like for servers to be able to restrict access to authorized users and to be able to authenticate requests for service. In this environment, a workstation cannot be trusted to identify its users correctly to network services. In particular, the following three threats exist:

1. A user may gain access to a particular workstation and pretend to be another user operating from that workstation.
2. A user may alter the network address of a workstation so that the requests sent from the altered workstation appear to come from the impersonated workstation.
3. A user may eavesdrop on exchanges and use a replay attack to gain entrance to a server or to disrupt operations.

In any of these cases, an unauthorized user may be able to gain access to services and data that he or she is not authorized to access. Rather than building in elaborate authentication protocols at each server, Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Unlike most other authentication schemes, Kerberos relies exclusively on symmetric encryption, making no use of public-key encryption.

Two versions of Kerberos are in common use. Version 4 implementations still exist. Version 5 corrects some of the security deficiencies of version 4 and has been issued as a proposed Internet Standard.

Today's environment is a distributed architecture consisting of dedicated user workstations (clients) and distributed or centralized servers. In this environment, three approaches to security can be envisioned.

1. Rely on each individual client workstation to assure the identity of its user or users and rely on each server to enforce a security policy based on user identification (ID).
2. Require that client systems authenticate themselves to servers, but trust the client system concerning the identity of its user.
3. Require the user to prove his or her identity for each service invoked. Also require that servers prove their identity to clients.

In a small, closed environment in which all systems are owned and operated by a single organization, the first or perhaps the second strategy may suffice. But in a more open environment in which network connections to other machines are supported, the third approach is needed to protect user information and resources housed at the server. Kerberos supports this third approach. Kerberos assumes distributed client/server architecture and employs one or more Kerberos servers to provide an authentication service.

Sequence of steps in authentication with Kerberos version is given here in table 13.1(a) to 13.1(c)

(a) Authentication Service Exchange to obtain ticket-granting ticket.

$$1.C \rightarrow AS : ID_C \,||\, ID_{tgs} \,||\, TS_1$$

$$2.AS \rightarrow C : E(K_c, [\mathrm{K}_{c,tgs} \,||\, ID_{tgs} \,||TS_2 \,||\, Lifetime_2 \,||\, \mathrm{Ti}\,cket_{tgs}])$$

$$Ticket_{tgs} = E(K_{tgs}, [\mathrm{K}_{c,tgs} \,||\, ID_C \,||\, \mathrm{AD}_C \,||\, \mathrm{ID}_{tgs} \,||TS_2 \,||\, Lifetime_2])$$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket.

$$3.\ C \rightarrow TGS : ID_v \,||\, ticket_{tgs} \,||\, Authenticator_c$$

$$4.\ TGS \rightarrow C : E(K_{c,tgs}, [\mathrm{K}_{c,\mathrm{v}} \,||\, ID_v \,||TS_4 \,||\, \mathrm{Ti}\,cket_v])$$

$$Ticket_{tgs} = E(K_{tgs}, [\mathrm{K}_{c,tgs} \,||\, ID_C \,||\, \mathrm{AD}_C \,||\, \mathrm{ID}_{tgs} \,||TS_2 \,||\, Lifetime_2])$$

$$Ticket_v = E(K_v, [\mathrm{K}_{c,\mathrm{v}} \,||\, ID_C \,||\, \mathrm{AD}_C \,||\, \mathrm{ID}_v \,||TS_4 \,||\, Lifetime_4])$$

$$Authenticator_c = E(K_{c,tgs}, [ID_C \,||\, \mathrm{AD}_C \,||TS_3])$$

(c) Client/Server Authentication Exchange to obtain service

5. $C \rightarrow V : \mathrm{T}icket_v \,||\, Authenticator_c$

6. $V \rightarrow C : E(K_{c,\mathrm{v}}, [TS_5 + 1])$ (for mutual authentication)

$Ticket_v = E(K_v, [\mathrm{K}_{c,\mathrm{v}} \,||\, ID_C \,||\, \mathrm{AD}_C \,||\, \mathrm{ID}_v \,||TS_4 \,||\, Lifetime_4])$

$Authenticator_c = E(K_{c,\mathrm{v}}, [ID_C \,||\, \mathrm{AD}_C \,||TS_5])$

Tables 13.1a to 13.1c: Summary of Kerberos Version4 Message Exchanges

The notations in the table are briefly explained here. The service TGS, issues tickets to users who have been authenticated to AS.

Kerberos protocol sequences of operations are described in tables 13.2 (a) to(c)

Message (1)	Client requests ticket-granting ticket.
ID_C	Tells AS identity of user from this client.
ID_{tgs}	Tells AS that user re quests access to TGS.
TS_1	Allows AS to verify that client's clock is synchronized with that of AS.
Message (2)	AS returns ticket-granting ticket.
$\mathrm{K_C}$	Encryption is based on user's password, enabling AS and client to verify Password and protecting contents of message (2).
$K_{c,tgs}$	Copy of session key accessible to client created by AS to permit secure exchange between client and TGS without requiring them to share a permanent key.
ID_{tgs}	Confirms that this ticket is for the TGS.
TS_2	Informs client of time this ticket was issued.
$Lifetime_2$	Informs client of the lifetime of this ticket.
$Ticket_{tgs}$	Ticket to be used by client to access TGS.

(a) Authentication Service Exchange

Message (3)	Client requests service-granting ticket.
ID_V	Tells TGS that user requests access to server V.
$Ticket_{tgs}$	Assures TGS that this user has been authenticated by AS.
$Authenticator_c$	Generated by client to validate ticket.
Message (4)	TGS returns service-granting ticket.
$K_{c,tgs}$	Key shared only by C and TGS protects contents of message (4).
$K_{c,v}$	Copy of session key accessible to client created by TGS to permit secure exchange between client and server without requiring them to share a permanent key.
ID_V	Confirms that this ticket is for server V.
TS_4	Informs client of time this ticket was issued.
$Ticket_V$	Ticket to be used by client to access server V.
$Ticket_{tgs}$	Reusable so that user does not have to reenter password.
K_{tgs}	Ticket is encrypted with key known only to AS and TGS, to prevent tampering.
$K_{c,tgs}$	Copy of session key accessible to TGS used to decrypt authenticator, thereby authenticating ticket.
ID_C	Indicates the rightful owner of this ticket.
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket.
ID_{tgs}	Assures server that it has decrypted ticket properly.
TS_2	Informs TGS of time this ticket was issued.
$Lifetime_2$	Prevents replay after ticket has expired.
$Authenticator_c$	Assures TGS that the ticket presenter is the same as the client for whom the ticket was issued has very short lifetime to prevent replay.
$K_{c,tgs}$	Authenticator is encrypted with key known only to client and TGS, to prevent tampering.
ID_C	Must match ID in ticket to authenticate ticket.

AD_C	Must match address in ticket to authenticate ticket.
TS_3	Informs TGS of time this authenticator was generated.

(b) Ticket-Granting Service Exchange

Message (5)	Client requests service.
$Ticket_V$	Assures server that this user has been authenticated by AS.
$Authenticator_c$	Generated by client to validate ticket.
Message (6)	Optional authentication of server to client.
$K_{c,v}$	Assures C that this message is from V.
TS_5+1	Assures C that this is not a replay of an old reply.
$Ticket_V$	Reusable so that client does not need to request a new ticket from TGS for each access to the same server.
K_v	Ticket is encrypted with key known only to TGS and server, to prevent tampering.
$K_{c,v}$	Copy of session key accessible to client; used to decrypt authenticator, thereby authenticating ticket.
ID_C	Indicates the rightful owner of this ticket.
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket.
ID_V	Assures server that it has decrypted ticket properly.
TS_4	Informs server of time this ticket was issued.
$Lifetime_4$	Prevents replay after ticket has expired.
$Authenticator_c$	Assures server that the ticket presenter is the same as the client for whom the ticket was issued; has very short lifetime to prevent replay.
$K_{c,v}$	Authenticator is encrypted with key known only to client and server, to prevent tampering.
ID_C	Must match ID in ticket to authenticate ticket.

AD_C	Must match address in ticket to authenticate ticket.
TS_5	Informs server of time this authenticator was generated.

(c) Client/Server Authentication Exchange

Table 13.2: Rationale for the Elements of the Kerberos Version 4 Protocol

13.5 SUMMARY

In this unit, review of internet is given in section 13.1. E-mail security is the topic in section 13.2. Pretty good privacy (PGP) is widely used method for security of E-mails. User authentication protocols are described in section 13.3. Section 13.4 introduces the standard authentication service Kerberos.

13.6 KEYWORDS

IP security, OSI, TCP-IP, E-mail security, PGP, mutual authentication, one way authentication

13.7 QUESTIONS

1. Give an overview of PGP of Zimmerman.
2. Mention the reasons for popularity of PGP.
3. Describe with figures the three forms of PGP.
4. Write about E-mail compatibility.
5. Differentiate mutual and one way authentication.
6. Write a protocol for mutual and one way authentication using symmetric and asymmetric encryption.
7. Write a note on Kerberos.

13.8 REFERENCES

1. Atul Kahate, Cryptography and Network Security, Tata MCGrawHill
2. Behrouz A Forouzan, Cryptography and Network security, McGraw Hill
3. Charlie Kauffman, Radia Perlman, Mike Spciner, Network Security, Pearson Education
4. William Stallings, Cryptography and Network Security, Pearson

UNIT -14: IP AND WEB SECURITY

Structure

14.0 OBJECTIVES

After going through the contents discussed in this unit you will

- ✓ Learn about challenges in securing web
- ✓ Come to know about potential threats
- ✓ Learn about relationship between layers and kind of security
- ✓ Know applications and benefits of securing IP
- ✓ Services provided by IP security

14.1 WEB SECURITY

The World Wide Web is nothing but client/server application running on the TCP/IP intranets. All the security tools discussed so far are useful for web also. But web poses new challenges which have not been addressed in computer and network security. We highlight here these new challenges exclusively to be addressed in securing web.

1. Unlike traditional environments, internet is two way. Hence web is vulnerable to attacks on the web servers over the internet.
2. Web is more and more used by Corporate for product information and business transactions. Reputation can be damaged and money can be lost if web servers are subverted.

3. Web browsers are easy to use; web servers are easy to manage, content on the web can be developed easily. Underlying software which makes these tasks easy are highly complex and possibly having security flaws. The brief history of web is filled with many examples of security attacks in spite of upgraded systems properly installed.
4. Web servers are often used as launching pad for an organization's computing systems. If this server is subverted, an attacker can have access to organization's secure data, although this is not part of the web.
5. Casual and untrained users are common clients for web based services. Such people may not be aware of security risks and quiet often do not have tools or knowledge to take effective counter measures.

Web security threats is summarized in table 14.1

	Threats	**Consequences**	**Countermeasures**
Integrity	• Modification of user data • Trojan horse browser • Modification of memory • Modification of message traffic in transit	• Loss of information • Compromise of machine • Vulnerability to all other threats	Cryptographic checksums
Confidentiality	• Eavesdropping on the net • Theft of info from server • Theft of data from client • Info about network configuration • Info about which client talks to server	• Loss of information • Loss of privacy	Encryption, Web proxies
Denial of Service	• Killing of user threads • Flooding machine with bogus requests • Filling up disk or memory • Isolating machine by DNS Attacks	• Disruptive • Annoying • Prevent user from getting work done	Difficult to prevent
Authentication	• Impersonation of legitimate users • Data forgery	• Misrepresentation of user • Belief that false information is valid	Cryptographic techniques

Table 14.1: A Comparison of Threats on the Web

Another way to classify Web security threats is in terms of the location of the threat: Web server, Web browser, and network traffic between browser and server. Issues of server and browser security fall into the category of computer system security.

Traffic security will be addressed in this unit. There are a number of approaches to provide Web security. The various approaches that are discussed are similar in services they provide and, to some extent, in the mechanisms that they use, but they differ with respect to their scope of applicability and their relative location within the TCP/IP protocol stack.

Figure 14.1 illustrates this difference. One way to provide Web security is to use IP security (IPsec) (Figure 14.1a).The advantage of using IPsec is that it is transparent to end users and applications and provides a general-purpose solution. Furthermore, IPsec includes a filtering capability so that only selected traffic need incur the overhead of IPsec processing.

Another relatively general-purpose solution is to implement security just above TCP (Figure 14.1b). The foremost example of this approach is the Secure Sockets Layer (SSL) and the follow-on Internet standard known as Transport Layer Security (TLS). At this level, there are two implementation choices. For full generality, SSL (or TLS) could be provided as part of the underlying protocol suite and therefore be transparent to applications. Alternatively, SSL can be embedded in specific packages. For example, Netscape and Microsoft Explorer browsers come equipped with SSL, and most Web servers have implemented the protocol.

Application-specific security services are embedded within the particular application. Figure 14.1c shows examples of this architecture. The advantage of this approach is that the service can be tailored to the specific needs of a given application.

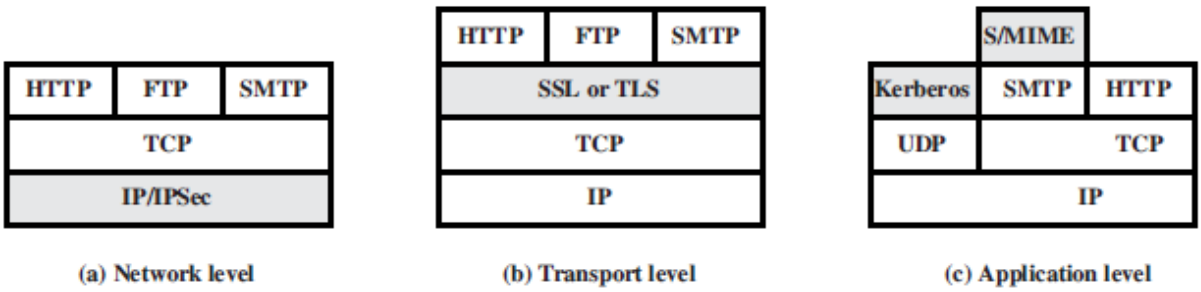


Figure 14.1: Relative Location of Security Facilities in the TCP/IP Protocol Stack

SSL Architecture

SSL is designed to make use of TCP to provide a reliable end-to-end secure service. SSL is not a single protocol but rather two layers of protocols, as illustrated in Figure 14.2.

The SSL Record Protocol provides basic security services to various higher layer protocols. In particular, the Hypertext Transfer Protocol (HTTP), which provides the transfer service for Web client/server interaction, can operate on top of SSL. Three higher-layer protocols are defined as part of SSL: the Handshake Protocol, the Change Cipher Spec Protocol, and the Alert Protocol.

Two important SSL concepts are the SSL session and the SSL connection, which are defined in the specification as follows.

- **Connection:** A connection is a transport (in the OSI layering model definition) that provides a suitable type of service. For SSL, such connections are peer-to-peer relationships. The connections are transient. Every connection is associated with one session.
- **Session:** An SSL session is an association between a client and a server. Sessions are created by the Handshake Protocol. Sessions define a set of cryptographic security parameters which can be shared among multiple connections. Sessions are used to avoid the expensive negotiation of new security parameters for each connection.

Between any pair of parties (applications such as HTTP on client and server), there may be multiple secure connections. In theory, there may also be multiple simultaneous sessions between parties, but this feature is not used in practice.

There are a number of states associated with each session. Once a session is established, there is a current operating state for both read and write (i.e., receive and send). In addition, during the Handshake Protocol, pending read and write states are created. Upon successful conclusion of the Handshake Protocol, the pending states become the current states.

Transport layer security (TLS)

TLS is an IETF standardization initiative whose goal is to produce an Internet standard version of SSL. TLS is defined as a Proposed Internet Standard in RFC 5246. RFC 5246 is very similar to SSLv3. In this section, we highlight the differences between TLS and SSL.

There are two differences between the SSLv3 and TLS MAC schemes: the actual algorithm and the scope of the MAC calculation. TLS makes use of the HMAC algorithm defined in RFC 2104.

TLS makes use of a pseudorandom function referred to as PRF to expand secrets into blocks of data for purposes of key generation or validation. The objective is to make use of a relatively small shared secret value but to generate longer blocks of data in a way that is secure from the kinds of attacks made on hash functions and MACs. The PRF is based on the data expansion function as given in the following steps.

P_hash(secret, seed)= HMAC_hash (secret, A(1) || seed) ||
HMAC_hash (secret, A(2) || seed) ||
HMAC_hash (secret, A(3) || seed) || . . .

where A( ) is defined as

A(0) = seed

A(i) = HMAC_hash (secret, A(i – 1))

14.2 IP SECURITY

To provide security, the IAB (Internet Architecture Board) included authentication and encryption as necessary security features in the next-generation IP, which has been issued as IPv6. Fortunately, these security capabilities were designed to be usable both with the current IPv4 and the future IPv6. This means that vendors can begin offering these features now, and many vendors now do have some IPsec capability in their products. The IPsec specification now exists as a set of Internet standards.

Applications of IP security

IPsec provides the capability to secure communications across a LAN, across private and public WANs, and across the Internet. Examples of its use include:

- **Secure branch office connectivity over the Internet:** A company can build a secure virtual private network over the Internet or over a public WAN. This enables a business

to rely heavily on the Internet and reduce its need for private networks, saving costs and network management overhead.

- **Secure remote access over the Internet:** An end user whose system is equipped with IP security protocols can make a local call to an Internet Service Provider (ISP) and gain secure access to a company network. This reduces the cost of toll charges for traveling employees and telecommuters.
- **Establishing extranet and intranet connectivity with partners:** IPsec can be used to secure communication with other organizations, ensuring authentication and confidentiality and providing a key exchange mechanism.
- **Enhancing electronic commerce security:** Even though some Web and electronic commerce applications have built-in security protocols, the use of IPsec enhances that security. IPsec guarantees that all traffic designated by the network administrator is both encrypted and authenticated, adding an additional layer of security to whatever is provided at the application layer.

The principal feature of IPsec that enables it to support these varied applications is that it can encrypt and/or authenticate *all* traffic at the IP level. Thus, all distributed applications (including remote logon, client/server, e-mail, file transfer, Web access, and so on) can be secured.

Figure 14.2 is a typical scenario of IPsec usage. An organization maintains LANs at dispersed locations. Non-secure IP traffic is conducted on each LAN. For traffic offsite, through some sort of private or public WAN, IPsec protocols are used. These protocols operate in networking devices, such as a router or a firewall, that connect each LAN to the outside world. The IPsec networking device will typically encrypt and compress all traffic going into the WAN and decrypt and decompress traffic coming from the WAN; these operations are transparent to workstations and servers on the LAN. Secure transmission is also possible with individual users who dial into the WAN. Such user workstations must implement the IPsec protocols to provide security.

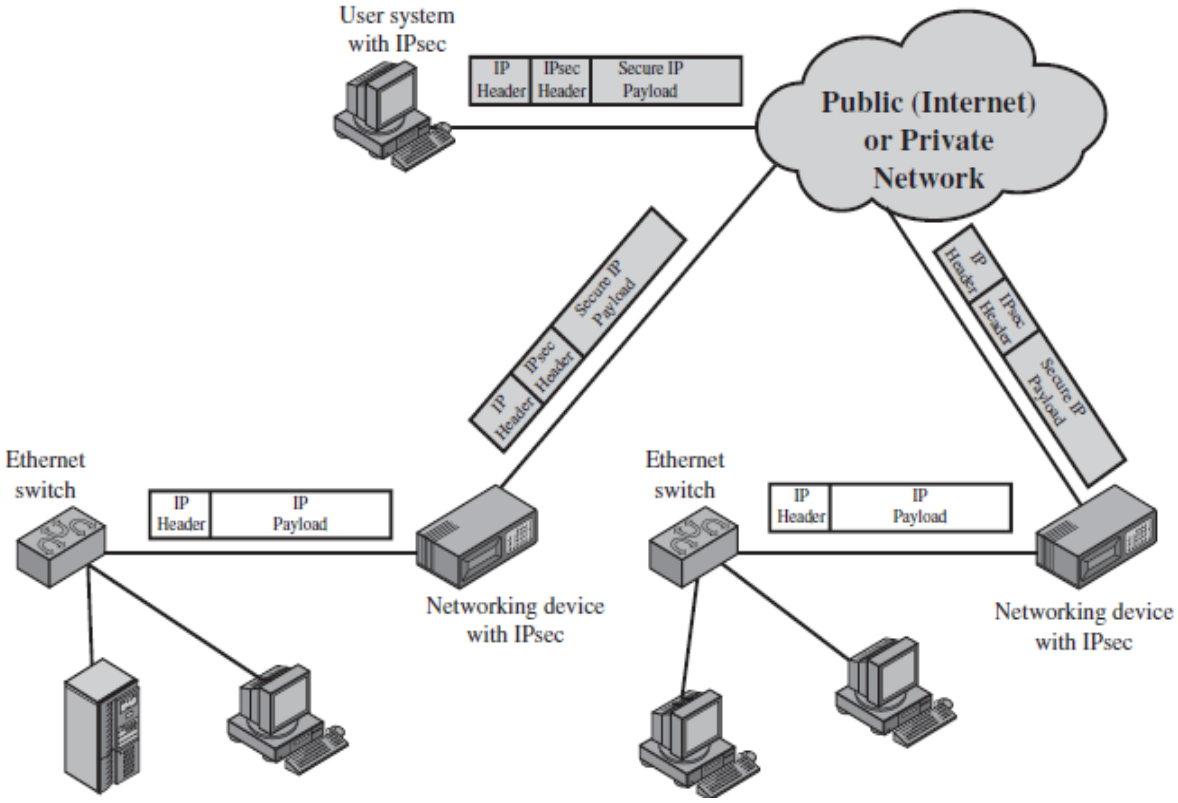


Figure 14.2: An IP Security Scenario

Benefits of IP security

- When IPsec is implemented in a firewall or router, it provides strong security that can be applied to all traffic crossing the perimeter. Traffic within a company or workgroup does not incur the overhead of security-related processing.
- IPsec in a firewall is resistant to bypass if all traffic from the outside must use IP and the firewall is the only means of entrance from the Internet into the organization.
- IPsec is below the transport layer (TCP, UDP) and so is transparent to applications. There is no need to change software on a user or server system when IPsec is implemented in the firewall or router. Even if IPsec is implemented in end systems, upper-layer software, including applications, is not affected.
- IPsec can be transparent to end users. There is no need to train users on security mechanisms, issue keying material on a per-user basis, or revoke keying material when users leave the organization.

- IPsec can provide security for individual users if needed. This is useful for offsite workers and for setting up a secure virtual subnetwork within an organization for sensitive applications.

IP security services

IPsec provides security services at the IP layer by enabling a system to select required security protocols, determine the algorithm(s) to use for the service(s), and put in place any cryptographic keys required to provide the requested services. Two protocols are used to provide security: an authentication protocol designated by the header of the protocol, Authentication Header (AH); and a combined encryption/ authentication protocol designated by the format of the packet for that protocol, Encapsulating Security Payload (ESP). RFC 4301 lists the following services:

1. Access control
2. Connectionless integrity
3. Data origin authentication
4. Rejection of replayed packets (a form of partial sequence integrity)
5. Confidentiality (encryption)
6. Limited traffic flow confidentiality

14.3 SUMMARY

In this unit a detailed description of web security is given in section 14.1. Threats on web security, secure socket layer and transport layer protocols are given in brief. In section 14.2 IP Security is discussed. Applications and benefits of IP security, and services of IP security are explained here briefly.

14.4 KEYWORDS

IP security – applications and benefits, RFC, IKS, ESP, AH [web security – threats, approaches, SSL , TSL]

14.5 QUESTIONS

1. Classify threats on security of web.

2. Explain approaches for security of web.
3. Discuss architecture of SSL.
4. Mention and explain operations in SSL protocol.
5. What are the applications of IP security?
6. Point out the benefits of IP security.
7. Mention the services of IP security.

14.6 REFERENCES

1. Atul Kahate, Cryptography and Network Security, Tata MCGrawHill
2. Charlie Kauffman, Radia Perlman, Mike Spciner, Network Security, Pearson Education
3. Wade Trappe, Lawrence Washington, Introduction to Cryptography with Coding Theory, Pearson International
4. William Stallings, Cryptography and Network Security, Pearson

UNIT -15: INTRUDER DETECTION AND PASSWORD MANAGEMENT

Structure

15.0 OBJECTIVES

When you finish reading this unit you will come to know about

- ✓ Classes of intruders
- ✓ Techniques intruders use for hacking
- ✓ Detection mechanism
- ✓ Threats to password based systems
- ✓ Efficient management of passwords

15.1 INTRUDERS

Unauthorized intrusion into a computer system or network is one of the most serious threats to computer security. User trespass can take the form of unauthorized logon to a machine or, in the case of an authorized user, acquisition of privileges or performance of actions beyond those that have been authorized. Software trespass can take the form of a virus, worm, or Trojan horse.

All these attacks relate to network security because system entry can be achieved by means of a network. However, these attacks are not confined to network-based attacks. A user with access to a local terminal may attempt trespass without using an intermediate network. A virus or Trojan horse may be introduced into a system by means of a diskette. Only the worm is a uniquely network phenomenon. Thus, system trespass is an area in which the concerns of network security and computer security overlap.

At first, we examine the nature of attacks to systems. One of the common threats to security of a system is intruder, also referred to as hacker or cracker. Intruders are classified into three classes.

i. Masquerader: Unauthorized individual who penetrates a system's access control to exploit an authorized user's account.
ii. Misfeasor: Genuine user accesses data for which he is not authorized or he is authorized for access but misuses his or her privileges.
iii. Clandestine user: A person who seizes supervisory control of the system and uses this control to evade auditing and access controls or to suppress audit collection.

Masquerader is mostly an outsider whereas misfeasor is generally an insider. Clandestine user can be outsider or insider.

Intruder attacks range from benign to the serious. Sometimes people explore internets and see what is out there. This is benign type. There are cases where individuals attempt to read/modify data in the system, though unauthorized. This is serious attack. These attacks are still not under total control. We point out some techniques for intrusion.

Intrusion techniques

The objective of the intruder is to gain access to a system or to increase the range of privileges accessible on a system. Generally this requires the intruder to acquire information that is protected. Often this information is in the form of a user password.

With knowledge of some other user's password, an intruder can log in to a system and exercise all the privileges accorded to the legitimate user. Typically, a system must maintain a file that associates a password with each authorized user. If such a file is stored with no

protection, then it is an easy matter to gain access to it and learn passwords. The password file can be protected in one of two ways:

- **One-way function:** The system stores only the value of a function based on the user's password. When the user presents a password, the system transforms that password and compares it with the stored value. In practice, the system usually performs a one-way transformation (not reversible) in which the password is used to generate a key for the one-way function and in which a fixed-length output is produced.
- **Access control:** Access to the password file is limited to one or a very few accounts.

If one or both of these countermeasures are in place, some effort is needed for a potential intruder to learn passwords. On the basis of a survey of the literature and interviews with a number of password crackers, the techniques for learning passwords are:

1. Try default passwords used with standard accounts that are shipped with the system. Many administrators do not bother to change these defaults.
2. Exhaustively try all short passwords (those of one to three characters).
3. Try words in the system's online dictionary or a list of likely passwords. Examples of the latter are readily available on hacker bulletin boards.
4. Collect information about users, such as their full names, the names of their spouse and children, pictures in their office, and books in their office that are related to hobbies.
5. Try users' phone numbers, Social Security numbers, and room numbers.
6. Try all legitimate license plate numbers for this state.
7. Use a Trojan horse (described in next unit) to bypass restrictions on access.
8. Tap the line between a remote user and the host system.

The first six methods are various ways of guessing a password. If an intruder has to verify the guess by attempting to log in, it is a tedious and easily countered means of attack. For example, a system can simply reject any login after three password attempts, thus requiring the intruder to reconnect to the host to try again. Under these circumstances, it is not practical to try more than a handful of passwords. However, the intruder is unlikely to try such crude methods. For example, if an intruder can gain access with a low level of privileges to an encrypted password file, then the strategy would be to capture that file and then use the encryption mechanism of that

particular system at leisure until a valid password that provided greater privileges was discovered.

Guessing attacks are feasible, and indeed highly effective, when a large number of guesses can be attempted automatically and each guess verified, without the guessing process being detectable.

The seventh method of attack, the Trojan horse, can be particularly difficult to counter. An example of a program that bypasses access controls was cited in the literature. A low-privilege user produced a game program and invited the system operator to use it in his or her spare time. The program did indeed play a game, but in the background it also contained code to copy the password file, which was unencrypted but access protected, into the user's file. Because the game was running under the operator's high-privilege mode, it was able to gain access to the password file.

The eighth attack listed namely line tapping, is a matter of physical security. It can be countered with link encryption techniques. Other intrusion techniques do not require learning a password. Intruders can get access to a system by exploiting attacks such as buffer overflows on a program that runs with certain privileges. Privilege escalation can be done this way as well.

We turn now to a discussion of the two principal countermeasures: detection and prevention. Detection is concerned with learning of an attack, either before or after its success. Prevention is a challenging security goal and an uphill battle at all times. The difficulty stems from the fact that the defender must attempt to thwart all possible attacks, whereas the attacker is free to try to find the weakest link in the defense chain and attack at that point.

15.2 INTRUSION DETECTION

Inevitably, the best intrusion prevention system will fail. A system's second line of defense is intrusion detection, and this has been the focus of much research in recent years. This interest is motivated by a number of considerations, including the following:

1. If an intrusion is detected quickly enough, the intruder can be identified and ejected from the system before any damage is done or any data are compromised. Even if the detection

is not sufficiently timely to preempt the intruder, the sooner that the intrusion is detected, the less the amount of damage and the more quickly that recovery can be achieved.

2. An effective intrusion detection system can serve as a deterrent, so acting to prevent intrusions.
3. Intrusion detection enables the collection of information about intrusion techniques that can be used to strengthen the intrusion prevention facility.

Intrusion detection is based on the assumption that the behavior of the intruder differs from that of a legitimate user in ways that can be quantified. Of course, we cannot expect that there will be a crisp, exact distinction between an attack by an intruder and the normal use of resources by an authorized user. Rather, we must expect that there will be some overlap.

Figure 15.1 suggests, in very abstract terms, the nature of the task confronting the designer of an intrusion detection system. Although the typical behavior of an intruder differs from the typical behavior of an authorized user, there is an overlap in these behaviors. Thus, a loose interpretation of intruder behavior, which will catch more intruders, will also lead to a number of "false positives," or authorized users identified as intruders. On the other hand, an attempt to limit false positives by a tight interpretation of intruder behavior will lead to an increase in false negatives, or intruders not identified as intruders. Thus, there is an element of compromise and art in the practice of intrusion detection.

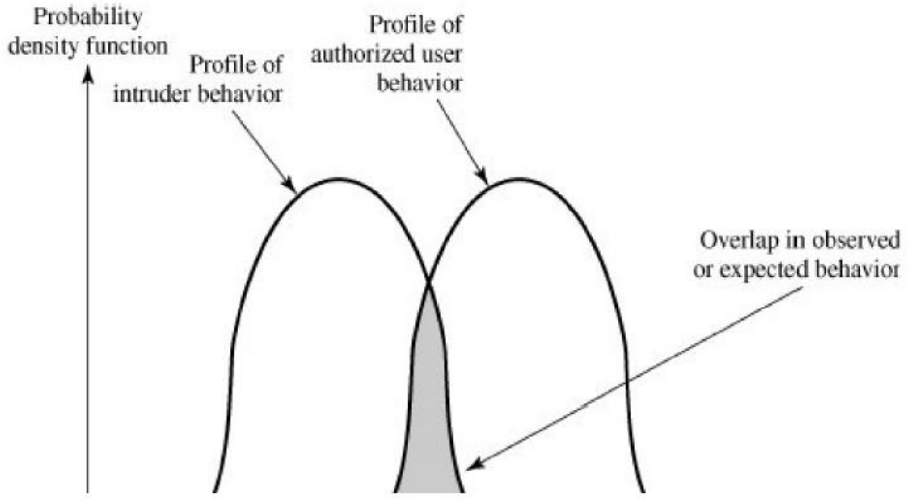


Figure 15.1: Profiles of Behavior of Intruders and Authorized User

There are studies postulating that one could, with reasonable confidence, distinguish between a masquerader and a legitimate user. Patterns of legitimate user behavior can be established by observing past history, and significant deviation from such patterns can be detected. The task of detecting a misfeasor (legitimate user performing in an unauthorized fashion) is more difficult, in that the distinction between abnormal and normal behavior may be small. Such violations would be undetectable solely through the search for anomalous behavior. However, misfeasor behavior might nevertheless be detectable by intelligent definition of the class of conditions that suggest unauthorized use. Finally, the detection of the clandestine user is still beyond the scope of purely automated techniques.

Approaches for intrusion detection are

1. Statistical anomaly detection: Involves the collection of data relating to the behavior of legitimate users over a period of time. Then statistical tests are applied to observed behavior to determine with a high level of confidence whether that behavior is not legitimate user behavior.

a. Threshold detection: This approach involves defining thresholds, independent of user, for the frequency of occurrence of various events.

b. Profile based: A profile of the activity of each user is developed and used to detect changes in the behavior of individual accounts.

2. Rule-based detection: Involves an attempt to define a set of rules that can be used to decide that a given behavior is that of an intruder.

a. Anomaly detection: Rules are developed to detect deviation from previous usage patterns.

b. Penetration identification: An expert system approach that searches for suspicious behavior.

In a nutshell, statistical approaches attempt to define normal or expected behavior, whereas rule-based approaches attempt to define proper behavior.

In terms of the types of attackers listed earlier, statistical anomaly detection is effective against masqueraders, who are unlikely to mimic the behavior patterns of the accounts they appropriate. On the other hand, such techniques may be unable to deal with misfeasors. For such attacks, rule-based approaches may be able to recognize events and sequences that, in context,

reveal penetration. In practice, a system may exhibit a combination of both approaches to be effective against a broad range of attacks.

Audit Records

A fundamental tool for intrusion detection is the audit record. Some record of ongoing activity by users must be maintained as input to an intrusion detection system. Basically, two plans are used:

- **Native audit records:** Virtually all multiuser operating systems include accounting software that collects information on user activity. The advantage of using this information is that no additional collection software is needed. The disadvantage is that the native audit records may not contain the needed information or may not contain it in a convenient form.
- **Detection-specific audit records:** A collection facility can be implemented that generates audit records containing only that information required by the intrusion detection system. One advantage of such an approach is that it could be made vendor independent and ported to a variety of systems. The disadvantage is the extra overhead involved in having, in effect, two accounting packages running on a machine.

A good example of detection-specific audit records reported in literature contains the following fields:

- **Subject:** Initiators of actions. A subject is typically a terminal user but might also be a process acting on behalf of users or groups of users. All activity arises through commands issued by subjects. Subjects may be grouped into different access classes, and these classes may overlap.
- **Action:** Operation performed by the subject on or with an object; for example, login, read, perform I/O, execute.
- **Object:** Receptors of actions. Examples include files, programs, messages, records, terminals, printers, and user- or program-created structures. When a subject is the recipient of an action, such as electronic mail, then that subject is considered an object. Objects may be grouped by type. Object granularity may vary by object type and by environment. For example, database actions may be audited for the database as a whole or at the record level.

- **Exception-Condition:** Denotes which, if any, exception condition is raised on return.
- **Resource-Usage:** A list of quantitative elements in which each element gives the amount used of some resource (e.g., number of lines printed or displayed, number of records read or written, processor time, I/O units used, session elapsed time).
- **Time-Stamp:** Unique time-and-date stamp identifying when the action took place.

The audit records provide input to the intrusion detection using statistical anomaly detection in two ways. First, the designer must decide on a number of quantitative metrics that can be used to measure user behavior. An analysis of audit records over a period of time can be used to determine the activity profile of the average user. Thus, the audit records serve to define typical behavior. Second, current audit records are the input used to detect intrusion. That is, the intrusion detection model analyzes incoming audit records to determine deviation from average behavior.

As far as rule based intrusion detection, audit records are examined as they are generated, and they are matched against the rule base. If a match is found, then the user's *suspicion rating* is increased. If enough rules are matched, then the rating will pass a threshold that results in the reporting of an anomaly.

Distributed Intrusion Detection

Until recently, work on intrusion detection systems focused on single-system stand-alone facilities. The typical organization, however, needs to defend a distributed collection of hosts supported by a LAN or internetwork. Although it is possible to mount a defense by using stand-alone intrusion detection systems on each host, a more effective defense can be achieved by coordination and cooperation among intrusion detection systems across the network.

The major issues in the design of a distributed intrusion detection system are:

- A distributed intrusion detection system may need to deal with different audit record formats. In a heterogeneous environment, different systems will employ different native audit collection systems and, if using intrusion detection, may employ different formats for security-related audit records.
- One or more nodes in the network will serve as collection and analysis points for the data from the systems on the network. Thus, either raw audit data or summary data must be

transmitted across the network. Therefore, there is a requirement to assure the integrity and confidentiality of these data. Integrity is required to prevent an intruder from masking his or her activities by altering the transmitted audit information. Confidentiality is required because the transmitted audit information could be valuable.

- Either a centralized or decentralized architecture can be used. With a centralized architecture, there is a single central point of collection and analysis of all audit data. This eases the task of correlating incoming reports but creates a potential bottleneck and single point of failure. With a decentralized architecture, there are more than one analysis centers, but these must coordinate their activities and exchange information.

15.3 PASSWORD PROTECTION AND MANAGEMENT

One important element of intrusion prevention is password management, with the goal of preventing unauthorized users from having access to the passwords of others.

The front line of defense against intruders is the password system. Virtually all multiuser systems require that a user provide not only a name or identifier (ID) but also a password. The password serves to authenticate the ID of the individual logging on to the system. In turn, the ID provides security in the following ways:

- The ID determines whether the user is authorized to gain access to a system. In some systems, only those who already have an ID filed on the system are allowed to gain access.
- The ID determines the privileges accorded to the user. A few users may have supervisory or "superuser" status that enables them to read files and perform functions that are especially protected by the operating system. Some systems have guest or anonymous accounts, and users of these accounts have more limited privileges than others.
- The ID is used in what is referred to as discretionary access control. For example, by listing the IDs of the other users, a user may grant permission to them to read files owned by that user.

The Vulnerability of Passwords

To understand the nature of the threat to password-based systems, let us consider a scheme that is widely used on UNIX, in which passwords are never stored in the clear. Rather, the following

procedure is employed. Each user selects a password of up to eight printable characters in length. This is converted into a 56-bit value (using 7-bit ASCII) that serves as the key input to an encryption routine. The encryption routine, known as crypt(3), is based on DES. The DES algorithm is modified using a 12-bit "salt" value. Typically, this value is related to the time at which the password is assigned to the user. The modified DES algorithm is exercised with a data input consisting of a 64-bit block of zeros. The output of the algorithm then serves as input for a second encryption. This process is repeated for a total of 25 encryptions. The resulting 64-bit output is then translated into an 11-character sequence. The hashed password is then stored, together with a plaintext copy of the salt, in the password file for the corresponding user ID. This method has been shown to be secure against a variety of cryptanalytic attacks.

The salt serves three purposes:

- It prevents duplicate passwords from being visible in the password file. Even if two users choose the same password, those passwords will be assigned at different times. Hence, the "extended" passwords of the two users will differ.
- It effectively increases the length of the password without requiring the user to remember two additional characters. Hence, the number of possible passwords is increased by a factor of 4096, increasing the difficulty of guessing a password.
- It prevents the use of a hardware implementation of DES, which would ease the difficulty of a brute-force guessing attack.

When a user attempts to log on to a UNIX system, the user provides an ID and a password. The operating system uses the ID to index into the password file and retrieve the plaintext salt and the encrypted password. The salt and user-supplied passwords are used as input to the encryption routine. If the result matches the stored value, the password is accepted.

The encryption routine is designed to discourage guessing attacks. Software implementations of DES are slow compared to hardware versions, and the use of 25 iterations multiplies the time required by 25. However, since the original design of this algorithm, two changes have occurred. First, newer implementations of the algorithm itself have resulted in speedups. Second, hardware performance continues to increase, so that any software algorithm executes more quickly.

Thus, there are two threats to the UNIX password scheme. First, a user can gain access on a machine using a guest account or by some other means and then run a password guessing program, called a password cracker, on that machine. The attacker should be able to check hundreds and perhaps thousands of possible passwords with little resource consumption. In addition, if an opponent is able to obtain a copy of the password file, then a cracker program can be run on another machine at leisure. This enables the opponent to run through many thousands of possible passwords in a reasonable period.

Even stupendous guessing rates do not yet make it feasible for an attacker to use a dumb brute-force technique of trying all possible combinations of characters to discover a password. Instead, password crackers rely on the fact that some people choose easily guessable passwords.

Some users, when permitted to choose their own password, pick one that is absurdly short. This makes password cracking very easy.

Password length is only part of the problem. Many people, when permitted to choose their own password, pick a password that is guessable, such as their own name, their street name, a common dictionary word, and so forth. This makes the job of password cracking straightforward. The cracker simply has to test the password file against lists of likely passwords. Because many people use guessable passwords, such a strategy should succeed on virtually all systems.

Password cracking programs could use one or all of the following strategies.

1. Try the user's name, initials, account name, and other relevant personal information. In all, 130 different permutations for each user were tried.
2. Try words from various dictionaries. The author compiled a dictionary of over 60,000 words, including the online dictionary on the system itself, and various other lists as shown.
3. Try various permutations on the words from step 2. This included making the first letter uppercase or a control character, making the entire word uppercase, reversing the word, changing the letter "o" to the digit "zero," and so on. These permutations added another 1 million words to the list.

4. Try various capitalization permutations on the words from step 2 that were not considered in step 3. This added almost 2 million additional words to the list.

Access Control

One way to thwart a password attack is to deny the opponent access to the password file. If the encrypted password portion of the file is accessible only by a privileged user, then the opponent cannot read it without already knowing the password of a privileged user. There are flaws in this strategy too.

- Many systems, including most UNIX systems, are susceptible to unanticipated break-ins. Once an attacker has gained access by some means, he or she may wish to obtain a collection of passwords in order to use different accounts for different logon sessions to decrease the risk of detection. Or a user with an account may desire another user's account to access privileged data or to sabotage the system.
- An accident of protection might render the password file readable, thus compromising all the accounts.
- Some of the users have accounts on other machines in other protection domains, and they use the same password. Thus, if the passwords could be read by anyone on one machine, a machine in another location might be compromised.

Thus, a more effective strategy would be to force users to select passwords that are difficult to guess.

Password Selection Strategies

The lesson from the two experiments just described is that, left to their own devices, many users choose a password that is too short or too easy to guess. At the other extreme, if users are assigned passwords consisting of eight randomly selected printable characters, password cracking is effectively impossible. But it would be almost as impossible for most users to remember their passwords. Fortunately, even if we limit the password universe to strings of characters that are reasonably memorable, the size of the universe is still too large to permit practical cracking. Our goal, then, is to eliminate guessable passwords while allowing the user to select a password that is memorable. Four basic techniques are in use:

- User education
- Computer-generated passwords

- Reactive password checking
- Proactive password checking

Users can be told the importance of using hard-to-guess passwords and can be provided with guidelines for selecting strong passwords. This **user education** strategy is unlikely to succeed at most installations, particularly where there is a large user population or a lot of turnover. Many users will simply ignore the guidelines. Others may not be good judges of what is a strong password. For example, many users (mistakenly) believe that reversing a word or capitalizing the last letter makes a password unguessable.

Computer-generated passwords also have problems. If the passwords are quite random in nature, users will not be able to remember them. Even if the password is pronounceable, the user may have difficulty remembering it and so be tempted to write it down. In general, computer-generated password schemes have a history of poor acceptance by users. FIPS PUB 181 defines one of the best-designed automated password generators. The standard includes not only a description of the approach but also a complete listing of the C source code of the algorithm. The algorithm generates words by forming pronounceable syllables and concatenating them to form a word. A random number generator produces a random stream of characters used to construct the syllables and words.

A **reactive password checking** strategy is one in which the system periodically runs its own password cracker to find guessable passwords. The system cancels any passwords that are guessed and notifies the user. This tactic has a number of drawbacks. First, it is resource intensive if the job is done right. Because a determined opponent who is able to steal a password file can devote full CPU time to the task for hours or even days, an effective reactive password checker is at a distinct disadvantage. Furthermore, any existing passwords remain vulnerable until the reactive password checker finds them.

The most promising approach to improved password security is a **proactive password checker**. In this scheme, a user is allowed to select his or her own password. However, at the time of selection, the system checks to see if the password is allowable and, if not, rejects it. Such checkers are based on the philosophy that, with sufficient guidance from the system, users

can select memorable passwords from a fairly large password space that are not likely to be guessed in a dictionary attack.

The trick with a proactive password checker is to strike a balance between user acceptability and strength. If the system rejects too many passwords, users will complain that it is too hard to select a password. If the system uses some simple algorithm to define what is acceptable, this provides guidance to password crackers to refine their guessing technique.

15.4 SUMMARY

In this unit we discussed issues relating to unauthorized intrusion into a system. In section 15.1 we defined and discussed in length, types of intruders, attack classification and techniques for intruding a system. The section that followed is about detection of intrusion. Two basic types of detection namely statistical and rule based methods are discussed here. The basic ingredients for these methods are audit records of users. Description of audit records is found in this section. Finally the unit closes with section 15.3 where in we described ways to secure passwords and prompting/guiding users to select a strong password.

15.5 KEYWORDS

Intruders, intrusion attacks, intrusion techniques, intrusion detection- standalone and distributed system, password protection, password management, password selection

15.6 QUESTIONS

1. Explain the classification of intruders.
2. How do intruders achieve their goals?
3. Differentiate intrusion detection and prevention? Which is easier to do why?
4. Discuss thoroughly
    i. Statistical anomaly detection
    ii. Rule based detection
5. What is audit record? Explain. How is this useful for detection intruders?
6. Explain detection of intrusion in distributed system environment.

7. Discuss UNIX way of maintaining passwords. Discuss the strengths and weaknesses of this method.
8. Write about password crackers.
9. Write a note on password selection strategies.

15.7 REFERENCES

1. Atul Kahate, Cryptography and Network Security, Tata MCGrawHill
2. Charlie Kauffman, Radia Perlman, Mike Spciner, Network Security, Pearson Education
3. William Stallings, Cryptography and Network Security, Pearson

UNIT -16: MALICIOUS SOFTWARE AND FIREWALLS

Structure

16.0 OBJECTIVES

When you have gone through the material discussed in this unit you will know

- ✓ What a malicious software is and its categories
- ✓ The basic differences between logic bomb, Trojan horses, viruses, worms
- ✓ About the solution to threat of malicious software
- ✓ Antivirus techniques working principle
- ✓ Ways in which firewall protects the system

16.1 MALICIOUS SOFTWARE

Malicious software is software that is intentionally included or inserted in a system for a harmful purpose. Perhaps the most sophisticated types of threats to computer systems are presented by programs that exploit vulnerabilities in computing systems. Such threats are referred to as **malicious software**, or **malware**. In this context, we are concerned with application programs as well as utility programs, such as editors and compilers.

We begin this section with an overview of the spectrum of such software threats. Malicious software can be divided into two categories: those that need a host program, and those

that are independent. The former are essentially fragments of programs that cannot exist independently of some actual application program, utility, or system program. Viruses, logic bombs, and backdoors are examples. The latter are self-contained programs that can be scheduled and run by the operating system. Worms and zombie programs are examples. Table 16.1 summarizes the various malicious software which are prevalent today.

Name	Description
Virus	Malware that, when executed, tries to replicate itself into other executable code; when it succeeds the code is said to be infected. When the infected code is executed, the virus also executes.
Worm	A computer program that can run independently and can propagate a complete working version of itself onto other hosts on a network.
Logic bomb	A program inserted into software by an intruder. A logic bomb lies dormant until a predefined condition is met; the program then triggers an unauthorized act.
Trojan horse	A computer program that appears to have a useful function, but also has a hidden and potentially malicious function that evades security mechanisms, sometimes by exploiting legitimate authorizations of a system entity that invokes the Trojan horse program.
Backdoor (trapdoor)	Any mechanism that bypasses a normal security check; it may allow unauthorized access to functionality.
Mobile code	Software (e.g., script, macro, or other portable instruction) that can be shipped unchanged to a heterogeneous collection of platforms and execute with identical semantics.
Exploits	Code specific to a single vulnerability or set of vulnerabilities.
Downloaders	Program that installs other items on a machine that is under attack. Usually, a downloader is sent in an e-mail.
Auto-rooter	Malicious hacker tools used to break into new machines remotely.
Kit (virus generator)	Set of tools for generating new viruses automatically.
Spammer programs	Used to send large volumes of unwanted e-mail.
Flooders	Used to attack networked computer systems with a large volume of traffic to carry out a denial-of-service (DoS) attack.
Keyloggers	Captures keystrokes on a compromised system.
Rootkit	Set of hacker tools used after attacker has broken into a computer system and gained root-level access.
Zombie, bot	Program activated on an infected machine that is activated to launch attacks on other machines.
Spyware	Software that collects information from a computer and transmits it to another system.
Adware	Advertising that is integrated into software. It can result in pop-up ads or redirection of a browser to a commercial site.

Table 16.1: Terminology of Malicious Programs

We can also differentiate between those software threats that do not replicate and those that do. The former are programs or fragments of programs that are activated by a trigger. Examples are logic bombs, backdoors, and zombie programs. The latter consists of either a program fragment

or an independent program that, when executed may produce one or more copies of itself to be activated later on the system. Viruses and worms are examples.

A survey of some malicious software is given here.

Backdoor: A **backdoor**, also known as a **trapdoor**, is a secret entry point into a program that allows someone who is aware of the backdoor to gain access without going through the usual security access procedures. Programmers have used backdoors legitimately for many years to debug and test programs; such a backdoor is called a **maintenance hook**. This usually is done when the programmer is developing an application that has an authentication procedure, or a long setup, requiring the user to enter many different values to run the application. To debug the program, the developer may wish to gain special privileges or to avoid all the necessary setup and authentication. The programmer may also want to ensure that there is a method of activating the program should something be wrong with the authentication procedure that is being built into the application. The backdoor is code that recognizes some special sequence of input or is triggered by being run from a certain user ID or by an unlikely sequence of events.

Backdoors become threats when unscrupulous programmers use them to gain unauthorized access.

Logic bomb: One of the oldest types of program threat, predating viruses and worms, is the logic bomb. The logic bomb is code embedded in some legitimate program that is set to "explode" when certain conditions are met. Examples of conditions that can be used as triggers for a logic bomb are the presence or absence of certain files, a particular day of the week or date, or a particular user running the application. Once triggered, a bomb may alter or delete data or entire files, cause a machine halt, or do some other damage.

Trojan Horses: A Trojan horse is a useful, or apparently useful, program or command procedure containing hidden code that, when invoked, performs some unwanted or harmful function.

Trojan horse programs can be used to accomplish functions indirectly that an unauthorized user could not accomplish directly. For example, to gain access to the files of another user on a shared system, a user could create a Trojan horse program that, when executed, changed the invoking user's file permissions so that the files are readable by any user. The author

could then induce users to run the program by placing it in a common directory and naming it such that it appears to be a useful utility. An example is a program that ostensibly produces a listing of the user's files in a desirable format. After another user has run the program, the author can then access the information in the user's files. An example of a Trojan horse program that would be difficult to detect is a compiler that has been modified to insert additional code into certain programs as they are compiled, such as a system login program. The code creates a backdoor in the login program that permits the author to log on to the system using a special password. This Trojan horse can never be discovered by reading the source code of the login program.

Another common motivation for the Trojan horse is data destruction. The program appears to be performing a useful function (e.g., a calculator program), but it may also be quietly deleting the user's files.

Zombie: A zombie is a program that secretly takes over another Internet-attached computer and then uses that computer to launch attacks that are difficult to trace to the zombie's creator. Zombies are used in denial-of-service attacks, typically against targeted Web sites. The zombie is planted on hundreds of computers belonging to unsuspecting third parties, and then used to overwhelm the target Web site by launching an overwhelming onslaught of Internet traffic.

16.2 VIRUSES

A virus is a piece of software that can "infect" other programs by modifying them; the modification includes a copy of the virus program, which can then go on to infect other programs.

Biological viruses are tiny scraps of genetic code-DNA or RNA-that can take over the machinery of a living cell and trick it into making thousands of flawless replicas of the original virus. Like its biological counterpart, a computer virus carries in its instructional code the recipe for making perfect copies of itself. The typical virus becomes embedded in a program on a computer. Then, whenever the infected computer comes into contact with an uninfected piece of software, a fresh copy of the virus passes into the new program. Thus, the infection can be spread from computer to computer by unsuspecting users who either swap disks or send programs to

one another over a network. In a network environment, the ability to access applications and system services on other computers provides a perfect culture for the spread of a virus.

A virus can do anything that other programs do. The only difference is that it attaches itself to another program and executes secretly when the host program is run. Once a virus is executing, it can perform any function, such as erasing files and programs.

During its lifetime, a typical virus goes through the following four phases:

- **Dormant phase:** The virus is idle. The virus will eventually be activated by some event, such as a date, the presence of another program or file, or the capacity of the disk exceeding some limit. Not all viruses have this stage.
- **Propagation phase:** The virus places an identical copy of itself into other programs or into certain system areas on the disk. Each infected program will now contain a clone of the virus, which will itself enter a propagation phase.
- **Triggering phase:** The virus is activated to perform the function for which it was intended. As with the dormant phase, the triggering phase can be caused by a variety of system events, including a count of the number of times that this copy of the virus has made copies of itself.
- **Execution phase:** The function is performed. The function may be harmless, such as a message on the screen, or damaging, such as the destruction of programs and data files.

Most viruses carry out their work in a manner that is specific to a particular operating system and, in some cases, specific to a particular hardware platform. Thus, they are designed to take advantage of the details and weaknesses of particular systems.

Virus Structure

A virus can be prepended or postpended to an executable program, or it can be embedded in some other fashion. The key to its operation is that the infected program, when invoked, will first execute the virus code and then execute the original code of the program.

A very general depiction of virus structure is shown in figure 16.1. In this case, the virus code, V, is pretended to infected programs, and it is assumed that the entry point to the program, when invoked, is the first line of the program.

```
     program V :=

{goto main;
     1234567;

     subroutine infect-executable :=
          {loop:
          file := get-random-executable-file;
          if (first-line-of-file = 1234567)
             then goto loop
             else prepend V to file; }

     subroutine do-damage :=
          {whatever damage is to be done}

     subroutine trigger-pulled :=
          {return true if some condition holds}

main:    main-program :=
          {infect-executable;
          if trigger-pulled then do-damage;
          goto next;}
next:

}
```

Figure 16.1: A simple virus

An infected program begins with the virus code and works as follows. The first line of code is a jump to the main virus program. The second line is a special marker that is used by the virus to determine whether or not a potential victim program has already been infected with this virus. When the program is invoked, control is immediately transferred to the main virus program. The virus program first seeks out uninfected executable files and infects them. Next, the virus may perform some action, usually detrimental to the system. This action could be performed every time the program is invoked, or it could be a logic bomb that triggers only under certain conditions. Finally, the virus transfers control to the original program. If the infection phase of the program is reasonably rapid, a user is unlikely to notice any difference between the execution of an infected and uninfected program.

A virus such as the one just described is easily detected because an infected version of a program is longer than the corresponding uninfected one. A way to thwart such a simple means of detecting a virus is to compress the executable file.

A virus program may be written to compress all uninfected executable programs and append the same to original program when user executes this infected file, the compressed version of original program is uncompressed and executed. Thus change in size of the file is not observed. Also running time is not longer. The virus spreads to other programs. Once virus gets entry into a system, it can affect all other executable files. Thus viral infection can be completely prevented only by preventing its entry. Unfortunately this is extremely difficult task, because a virus can be part of any program outside a system.

Types of Viruses

There has been a continuous arms race between virus writers and writers of antivirus software since viruses first appeared. As effective countermeasures have been developed for existing types of viruses, new types have been developed. The following categories are the most significant types of viruses:

- **Parasitic virus:** The traditional and still most common form of virus. A parasitic virus attaches itself to executable files and replicates, when the infected program is executed, by finding other executable files to infect.
- **Memory-resident virus:** Lodges in main memory as part of a resident system program. From that point on, the virus infects every program that executes.
- **Boot sector virus:** Infects a master boot record or boot record and spreads when a system is booted from the disk containing the virus.
- **Stealth virus:** A form of virus explicitly designed to hide itself from detection by antivirus software.
- **Polymorphic virus:** A virus that mutates with every infection, making detection by the "signature" of the virus impossible.
- **Metamorphic virus:** As with a polymorphic virus, a metamorphic virus mutates with every infection. The difference is that a metamorphic virus rewrites itself completely at each iteration, increasing the difficulty of detection. Metamorphic viruses my change their behavior as well as their appearance.

One example of a **stealth virus** was discussed earlier: a virus that uses compression so that the infected program is exactly the same length as an uninfected version. Far more sophisticated techniques are possible. For example, a virus can place intercept logic in disk I/O routines, so

that when there is an attempt to read suspected portions of the disk using these routines, the virus will present back the original, uninfected program. Thus, *stealth* is not a term that applies to a virus as such but, rather, is a technique used by a virus to evade detection.

A **polymorphic virus** creates copies during replication that are functionally equivalent but have distinctly different bit patterns. As with a stealth virus, the purpose is to defeat programs that scan for viruses. In this case, the "signature" of the virus will vary with each copy. To achieve this variation, the virus may randomly insert superfluous instructions or interchange the order of independent instructions. A more effective approach is to use encryption. A portion of the virus, generally called a *mutation engine*, creates a random encryption key to encrypt the remainder of the virus. The key is stored with the virus, and the mutation engine itself is altered. When an infected program is invoked, the virus uses the stored random key to decrypt the virus. When the virus replicates, a different random key is selected.

Another weapon in the virus writers' armory is the virus-creation toolkit. Such a toolkit enables a relative novice to create quickly a number of different viruses. Although viruses created with toolkits tend to be less sophisticated than viruses designed from scratch, the sheer number of new viruses that can be generated creates a problem for antivirus schemes.

Macro Viruses

In the mid-1990s, macro viruses became by far the most prevalent type of virus. Macro viruses are particularly threatening for a number of reasons:

1. A macro virus is platform independent. Virtually all of the macro viruses infect Microsoft Word documents. Any hardware platform and operating system that supports Word can be infected.
2. Macro viruses infect documents, not executable portions of code. Most of the information introduced onto a computer system is in the form of a document rather than a program.
3. Macro viruses are easily spread. A very common method is by electronic mail.

Recent releases of Word provide increased protection against macro viruses. For example, Microsoft offers an optional Macro Virus Protection tool that detects suspicious Word files and alerts the customer to the potential risk of opening a file with macros. Various antivirus product vendors have also developed tools to detect and correct macro viruses.

E-mail Viruses

A more recent development in malicious software is the e-mail virus. The first rapidly spreading e-mail viruses, such as Melissa, made use of a Microsoft Word macro embedded in an attachment. If the recipient opens the e-mail attachment, the Word macro is activated. Then

1. The e-mail virus sends itself to everyone on the mailing list in the user's e-mail package.
2. The virus does local damage.

At the end of 1999, a more powerful version of the e-mail virus appeared. This newer version can be activated merely by opening an e-mail that contains the virus rather than opening an attachment. The virus uses the Visual Basic scripting language supported by the e-mail package.

Thus we see a new generation of malware that arrives via e-mail and uses e-mail software features to replicate itself across the Internet. The virus propagates itself as soon as activated (either by opening an e-mail attachment of by opening the e-mail) to all of the e-mail addresses known to the infected host. As a result, whereas viruses used to take months or years to propagate, they now do so in hours. This makes it very difficult for antivirus software to respond before much damage is done. Ultimately, a greater degree of security must be built into Internet utility and application software on PCs to counter the growing threat.

16.3 WORMS

A worm is a program that can replicate itself and send copies from computer to computer across network connections. Upon arrival, the worm may be activated to replicate and propagate again. In addition to propagation, the worm usually performs some unwanted function. An e-mail virus has some of the characteristics of a worm, because it propagates itself from system to system. However, we can still classify it as a virus because it requires a human to move it forward. A worm actively seeks out more machines to infect and each machine that is infected serves as an automated launching pad for attacks on other machines.

Network worm programs use network connections to spread from system to system. Once active within a system, a network worm can behave as a computer virus or bacteria, or it could implant Trojan horse programs or perform any number of disruptive or destructive actions.

To replicate itself, a network worm uses some sort of network vehicle. Examples include the following:

- **Electronic mail facility:** A worm mails a copy of itself to other systems.
- **Remote execution capability:** A worm executes a copy of itself on another system.
- **Remote login capability:** A worm logs onto a remote system as a user and then uses commands to copy itself from one system to the other.

The new copy of the worm program is then run on the remote system where, in addition to any functions that it performs at that system, it continues to spread in the same fashion.

A network worm exhibits the same characteristics as a computer virus: a dormant phase, a propagation phase, a triggering phase, and an execution phase. The propagation phase generally performs the following functions:

1. Search for other systems to infect by examining host tables or similar repositories of remote system addresses.
2. Establish a connection with a remote system.
3. Copy itself to the remote system and cause the copy to be run.

The network worm may also attempt to determine whether a system has previously been infected before copying itself to the system. In a multiprogramming system, it may also disguise its presence by naming itself as a system process or using some other name that may not be noticed by a system operator. As with viruses, network worms are difficult to counter.

Morris worm (1998) was designed to spread on UNIX systems and used a number of different techniques for propagation. When Morris worm program is executed, its first task was to discover other hosts that would allow entry from this host. This is done by examining variety of lists and tables, including system tables that declared other trusted hosts, email forwarding files etc. More recent worms are code Red (2001), SQL Slammer (2003) Sobig.f(2003) and Mydoom (2004). Code red exploits security holes in Microsoft Internet Information server to generate and spread. It is capable of initiating denial of service attacks against a Government website by flooding the site with packets from numerous hosts. The worm then suspends activities and reactivates periodically. Code Red II is a variant of Code Red, which installs a backdoor allowing hacker to direct activities of victim computers. This worm modifies .htm,

.html, .asp files. SQL Slammer worm exploited buffer overflow vulnerability in Microsoft SQL server. Sobig.f exploited open proxy servers. It was reported that at its peak, one in every 17 messages was affected by this worm. Mydoom is a mass-mailing email worm. It was capable of installing a backdoor in infected computers allowing entry for hackers.

For each discovered host, the worm tried a number of methods for gaining access:

1. It attempted to log on to a remote host as a legitimate user. In this method, the worm first attempted to crack the local password file, and then used the discovered passwords and corresponding user IDs. The assumption was that many users would use the same password on different systems. To obtain the passwords, the worm ran a password-cracking program that tried.
   a) Each user's account name and simple permutations of it
   b) A list of 432 built-in passwords that Morris thought to be likely candidates
   c) All the words in the local system directory
2. It exploited a bug in the finger protocol, which reports the whereabouts of a remote user.
3. It exploited a trapdoor in the debug option of the remote process that receives and sends mail.

If any of these attacks succeeded, the worm achieved communication with the operating system command interpreter. It then sent this interpreter a short bootstrap program, issued a command to execute that program, and then logged off. The bootstrap program then called back the parent program and downloaded the remainder of the worm. The new worm was then executed.

State of Worm Technology

The state of the art in worm technology includes the following:

- **Multiplatform:** Newer worms are not limited to Windows machines but can attack a variety of platforms, especially the popular varieties of UNIX.
- **Multiexploit:** New worms penetrate systems in a variety of ways, using exploits against Web servers, browsers, e-mail, file sharing, and other network-based applications.
- **Ultrafast spreading:** One technique to accelerate the spread of a worm is to conduct a prior Internet scan to accumulate Internet addresses of vulnerable machines.

- **Polymorphic:** To evade detection, skip past filters, and foil real-time analysis, worms adopt the virus polymorphic technique. Each copy of the worm has new code generated on the fly using functionally equivalent instructions and encryption techniques.
- **Metamorphic:** In addition to changing their appearance, metamorphic worms have a repertoire of behavior patterns that are unleashed at different stages of propagation.
- **Transport vehicles:** Because worms can rapidly compromise a large number of systems, they are ideal for spreading other distributed attack tools, such as distributed denial of service zombies.
- **Zero-day exploit:** To achieve maximum surprise and distribution, a worm should exploit an unknown vulnerability that is only discovered by the general network community when the worm is launched.

16.4 VIRUS COUNTER MEASURES

We discuss general approaches of antivirus software and advanced techniques.

Antivirus Approaches

The ideal solution to the threat of viruses is prevention: Do not allow a virus to get into the system in the first place. This goal is, in general, impossible to achieve, although prevention can reduce the number of successful viral attacks. The next best approach is to be able to do the following:

- **Detection:** Once the infection has occurred, determine that it has occurred and locate the virus.
- **Identification:** Once detection has been achieved, identify the specific virus that has infected a program.
- **Removal:** Once the specific virus has been identified, remove all traces of the virus from the infected program and restore it to its original state. Remove the virus from all infected systems so that the disease cannot spread further.

If detection succeeds but either identification or removal is not possible, then the alternative is to discard the infected program and reload a clean backup version.

Advances in virus and antivirus technology go hand in hand. Early viruses were relatively simple code fragments and could be identified and purged with relatively simple antivirus software packages. As the virus arms race has evolved, both viruses and, necessarily, antivirus software have grown more complex and sophisticated.

The four generations of antivirus software are:

- First generation: simple scanners
- Second generation: heuristic scanners
- Third generation: activity traps
- Fourth generation: full-featured protection

A **first-generation** scanner requires a virus signature to identify a virus. The virus may contain "wildcards" but has essentially the same structure and bit pattern in all copies. Such signature-specific scanners are limited to the detection of known viruses. Another type of first-generation scanner maintains a record of the length of programs and looks for changes in length.

A **second-generation** scanner does not rely on a specific signature. Rather, the scanner uses heuristic rules to search for probable virus infection. One class of such scanners looks for fragments of code that are often associated with viruses. For example, a scanner may look for the beginning of an encryption loop used in a polymorphic virus and discover the encryption key. Once the key is discovered, the scanner can decrypt the virus to identify it, then remove the infection and return the program to service.

Another second-generation approach is integrity checking. A checksum can be appended to each program. If a virus infects the program without changing the checksum, then an integrity check will catch the change. To counter a virus that is sophisticated enough to change the checksum when it infects a program, an encrypted hash function can be used. The encryption key is stored separately from the program so that the virus cannot generate a new hash code and encrypt that. By using a hash function rather than a simpler checksum, the virus is prevented from adjusting the program to produce the same hash code as before.

Third-generation programs are memory-resident programs that identify a virus by its actions rather than its structure in an infected program. Such programs have the advantage that it

is not necessary to develop signatures and heuristics for a wide array of viruses. Rather, it is necessary only to identify the small set of actions that indicate an infection is being attempted and then to intervene.

Fourth-generation products are packages consisting of a variety of antivirus techniques used in conjunction. These include scanning and activity trap components. In addition, such a package includes access control capability, which limits the ability of viruses to penetrate a system and then limits the ability of a virus to update files in order to pass on the infection.

The arms race continues. With fourth-generation packages, a more comprehensive defense strategy is employed, broadening the scope of defense to more general-purpose computer security measures.

Advanced Antivirus Techniques

More sophisticated antivirus approaches and products continue to appear. In this subsection, we highlight two of the most important.

Generic Decryption: Generic decryption (GD) technology enables the antivirus program to easily detect even the most complex polymorphic viruses, while maintaining fast scanning speeds. Recall that when a file containing a polymorphic virus is executed, the virus must decrypt itself to activate. In order to detect such a structure, executable files are run through a GD scanner, which contains the following elements:

- **CPU emulator:** A software-based virtual computer. Instructions in an executable file are interpreted by the emulator rather than executed on the underlying processor. The emulator includes software versions of all registers and other processor hardware, so that the underlying processor is unaffected by programs interpreted on the emulator.
- **Virus signature scanner:** A module that scans the target code looking for known virus signatures.
- **Emulation control module:** Controls the execution of the target code.

At the start of each simulation, the emulator begins interpreting instructions in the target code, one at a time. Thus, if the code includes a decryption routine that decrypts and hence exposes the virus, that code is interpreted. In effect, the virus does the work for the antivirus

program by exposing the virus. Periodically, the control module interrupts interpretation to scan the target code for virus signatures.

During interpretation, the target code can cause no damage to the actual personal computer environment, because it is being interpreted in a completely controlled environment.

The most difficult design issue with a GD scanner is to determine how long to run each interpretation. Typically, virus elements are activated soon after a program begins executing, but this need not be the case. The longer the scanner emulates a particular program, the more likely it is to catch any hidden viruses. However, the antivirus program can take up only a limited amount of time and resources before users complain.

16.5 FIREWALLS

Firewalls can be an effective means of protecting a local system or network of systems from network-based security threats while at the same time affording access to the outside world via wide area networks and the Internet.

Internet connectivity is no longer optional for organizations. The information and services available are essential to the organization. Moreover, individual users within the organization want and need Internet access, and if this is not provided via their LAN, they will use dial-up capability from their PC to an Internet service provider (ISP). However, while Internet access provides benefits to the organization, it enables the outside world to reach and interact with local network assets. This creates a threat to the organization. While it is possible to equip each workstation and server on the premises network with strong security features, such as intrusion protection, this is not a practical approach. Consider a network with hundreds or even thousands of systems, running a mix of various versions of UNIX, plus Windows. When a security flaw is discovered, each potentially affected system must be upgraded to fix that flaw. The alternative, increasingly accepted, is the firewall. The firewall is inserted between the premises network and the Internet to establish a controlled link and to erect an outer security wall or perimeter. The aim of this perimeter is to protect the premises network from Internet-based attacks and to provide a single choke point where security and audit can be imposed. The

firewall may be a single computer system or a set of two or more systems that cooperate to perform the firewall function.

Firewall characteristics

The following are the design goals of a firewall

1. All traffic from inside to outside, and vice versa, must pass through the firewall. This is achieved by physically blocking all access to the local network except via the firewall. Various configurations are possible.
2. Only authorized traffic, as defined by the local security policy, will be allowed to pass. Various types of firewalls are used, which implement various types of security policies, as explained later in this section.
3. The firewall itself is immune to penetration. This implies that use of a trusted system with a secure operating system.

There are four essential controls exercised by a firewall

- **Service control:** Determines the types of Internet services that can be accessed, inbound or outbound. The firewall may filter traffic on the basis of IP address and TCP port number; may provide proxy software that receives and interprets each service request before passing it on; or may host the server software itself, such as a Web or mail service.
- **Direction control:** Determines the direction in which particular service requests may be initiated and allowed to flow through the firewall.
- **User control:** Controls access to a service according to which user is attempting to access it. This feature is typically applied to users inside the firewall perimeter (local users). It may also be applied to incoming traffic from external users; the latter requires some form of secure authentication technology, such as is provided in IPSec .
- **Behavior control:** Controls how particular services are used. For example, the firewall may filter e-mail to eliminate spam, or it may enable external access to only a portion of the information on a local Web server.

Firewalls have their limitations, including the following:

1. The firewall cannot protect against attacks that bypass the firewall. Internal systems may have dial-out capability to connect than ISP. An internal LAN may support a modem pool that provides dial-in capability for traveling employees and telecommuters.

2. The firewall does not protect against internal threats, such as a disgruntled employee or an employee who unwittingly cooperates with an external attacker.
3. The firewall cannot protect against the transfer of virus-infected programs or files. Because of the variety of operating systems and applications supported inside the perimeter, it would be impractical and perhaps impossible for the firewall to scan all incoming files, e-mail, and messages for viruses.

Types of Firewalls

Figure 16.2 illustrates the three common types of firewalls: packet filters, application-level gateways, and circuit-level gateways. We examine each of these in turn.

Packet-Filtering Router

A packet-filtering router applies a set of rules to each incoming and outgoing IP packet and then forwards or discards the packet. The router is typically configured to filter packets going in both directions (from and to the internal network). Filtering rules are based on information contained in a network packet:

- **Source IP address:** The IP address of the system that originated the IP packet (e.g., 192.178.1.1)
- **Destination IP address:** The IP address of the system the IP packet is trying to reach (e.g., 192.168.1.2)
- **Source and destination transport-level address:** The transport level (e.g., TCP or UDP) port number, which defines applications such as SNMP or TELNET
- **IP protocol field:** Defines the transport protocol
- **Interface:** For a router with three or more ports, which interface of the router the packet came from or which interface of the router the packet is destined for.

The packet filter is typically set up as a list of rules based on matches to fields in the IP or TCP header. If there is a match to one of the rules, that rule is invoked to determine whether to forward or discard the packet. If there is no match to any rule, then a default action is taken. Two default policies are possible:

- **Default** = *discard*: That which is not expressly permitted is prohibited.
- **Default** = *forward*: That which is not expressly prohibited is permitted.

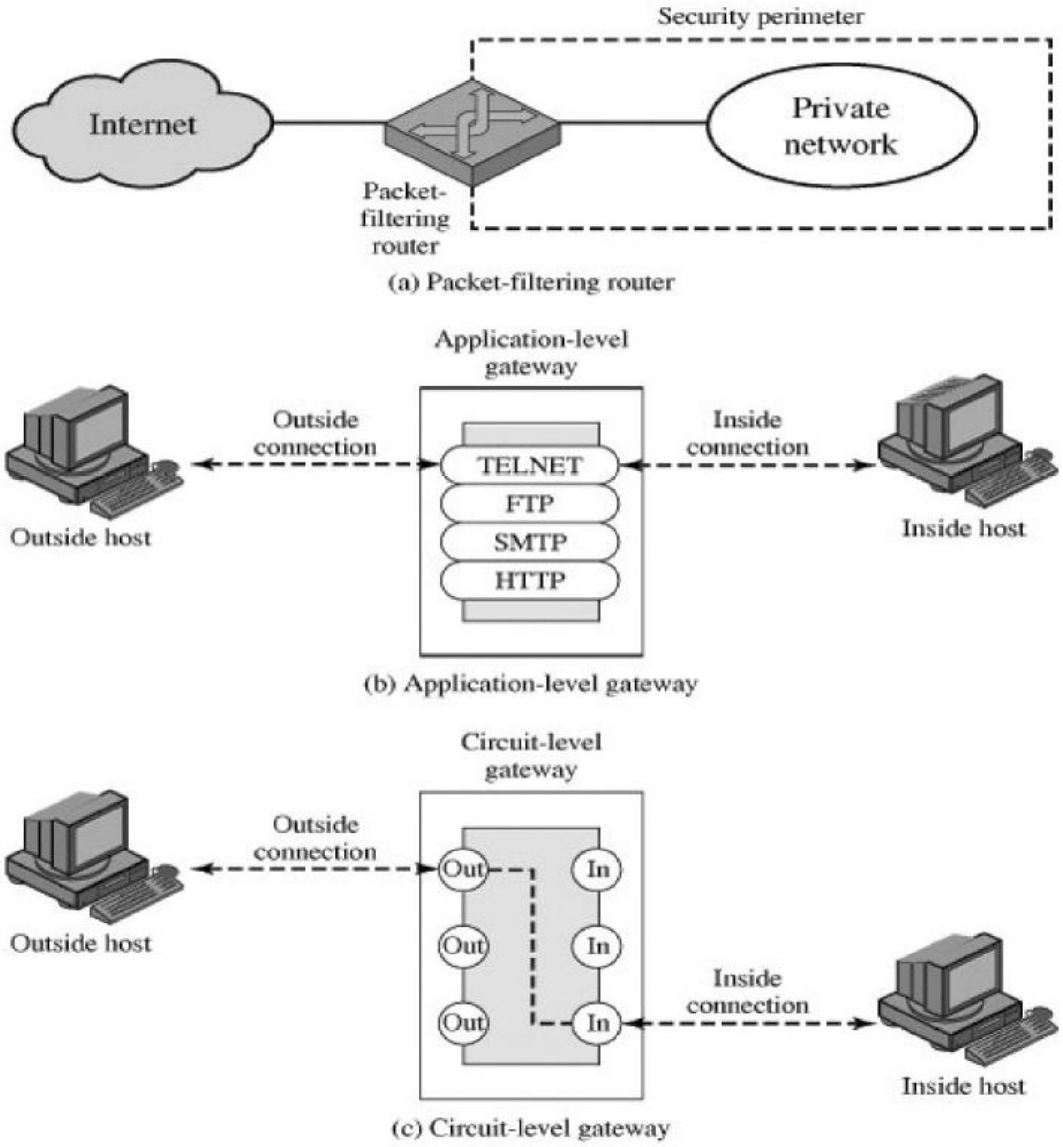


Figure 16.2: Firewall types

The default discard policy is more conservative. Initially, everything is blocked, and services must be added on a case-by-case basis. This policy is more visible to users, who are more likely to see the firewall as a hindrance. The default forward policy increases ease of use for end users

but provides reduced security; the security administrator must, in essence, react to each new security threat as it becomes known.

Application-Level Gateway

An application-level gateway, also called a proxy server, acts as a relay of application-level traffic (Figure 16.2). The user contacts the gateway using a TCP/IP application, such as Telnet or FTP, and the gateway asks the user for the name of the remote host to be accessed. When the user responds and provides a valid user ID and authentication information, the gateway contacts the application on the remote host and relays TCP segments containing the application data between the two endpoints. If the gateway does not implement the proxy code for a specific application, the service is not supported and cannot be forwarded across the firewall. Further, the gateway can be configured to support only specific features of an application that the network administrator considers acceptable while denying all other features.

Application-level gateways tend to be more secure than packet filters. Rather than trying to deal with the numerous possible combinations that are to be allowed and forbidden at the TCP and IP level, the application-level gateway need only scrutinize a few allowable applications. In addition, it is easy to log and audit all incoming traffic at the application level.

A prime disadvantage of this type of gateway is the additional processing overhead on each connection. In effect, there are two spliced connections between the end users, with the gateway at the splice point, and the gateway must examine and forward all traffic in both directions.

Circuit-Level Gateway

A third type of firewall is the circuit-level gateway (Figure 16.2). This can be a stand-alone system or it can be a specialized function performed by an application-level gateway for certain applications. A circuit-level gateway does not permit an end-to-end TCP connection; rather, the gateway sets up two TCP connections, one between itself and a TCP user on an inner host and one between itself and a TCP user on an outside host. Once the two connections are established, the gateway typically relays TCP segments from one connection to the other without examining the contents. The security function consists of determining which connections will be allowed.

A typical use of circuit-level gateways is a situation in which the system administrator trusts the internal users. The gateway can be configured to support application-level or proxy service on inbound connections and circuit-level functions for outbound connections. In this configuration, the gateway can incur the processing overhead of examining incoming application data for forbidden functions but does not incur that overhead on outgoing data.

16.6 SUMMARY

This unit began with discussions on malicious software viruses, worms, Trojan horse, logic bomb, zombie. Phases of a virus in its life cycle, types of viruses are discussed in section 16.2. The behavior programs and latest in worm technology are listed in section 16.3. Counter measures possible prevent and remedy virus attacks are given the section next. The unit concludes with a discussion on firewalls.

16.7 KEYWORDS

Malicious software, Trojan horse, zombie, virus, worms, virus counter measures, firewalls.

16.8 QUESTIONS

1. Write about logic bomb, Trojan horse, and zombie.
2. What are the phases of virus?
3. What is the structure of a virus?
4. What is the role of compression in the operation of a virus?
5. What is the role of encryption in the operation of a virus?
6. Explain propagation of worms.
7. What are the types of viruses?
8. How can counter measures be devised for virus attacks?
9. List the design goals of firewalls.
10. Explain the techniques used for controlling access with a firewall.
11. Explain firewall design mechanisms
    a) Packet filter

b) Application level gateway

c) Circuit level gateway.

16.9 REFERENCES

1. Atul Kahate, Cryptography and Network Security, Tata MCGrawHill
2. Charlie Kauffman, Radia Perlman, Mike Spciner, Network Security, Pearson Education
3. William Stallings, Cryptography and Network Security, Pearson